Appendix A

Library Functions

The functions within the standard compiler library are listed in this chapter. Each entry begins with the name of the function. This is followed by information decomposed into the following categories.

Synopsis the C declaration of the function, and the header file in which it is declared.

Description a narrative description of the function and its purpose.

Example an example of the use of the function. It is usually a complete small program that illustrates the function.

Data types any special data types (structures etc.) defined for use with the function. These data types will be defined in the header file named under **Synopsis**.

See also any allied functions.

Return value the type and nature of the return value of the function, if any. Information on error returns is also included

Only those categories which are relevant to each function are used.

__CONFIG

Synopsis

```
#include <htc.h>

__CONFIG(data)
```

Description

This macro is used to program the configuration fuses that set the device into various modes of operation.

The macro accepts the 16-bit value it is to update it with.

16-Bit masks have been defined to describe each programmable attribute available on each device. These attribute masks can be found tabulated in this manual in the Features and Runtime Environment section.

Multiple attributes can be selected by ANDing them together.

Example

```
#include <htc.h>

__CONFIG(RC & UNPROTECT)

void
main (void)
{
}
```

See also

__EEPROM_DATA(), __IDLOC(), __IDLOC7()

__EEPROM_DATA

Synopsis

```
#include <htc.h>

__EEPROM_DATA(a,b,c,d,e,f,g,h)
```

Description

This macro is used to store initial values into the device's EEPROM registers at the time of programming.

The macro must be given blocks of 8 bytes to write each time it is called, and can be called repeatedly to store multiple blocks.

__EEPROM_DATA() will begin writing to EEPROM address zero, and will auto-increment the address written to by 8, each time it is used.

Example

```
#include <htc.h>

__EEPROM_DATA(0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07)
__EEPROM_DATA(0x08,0x09,0x0A,0x0B,0x0C,0x0D,0x0E,0x0F)

void
main (void)
{
}
```

See also

__CONFIG()

__IDLOC

Synopsis

```
#include <htc.h>

__IDLOC(x)
```

Description

This macro places data into the device's special locations outside of addressable memory reserved for ID. This would be useful for storage of serial numbers etc.

The macro will attempt to write 4 nibbles of data to the 4 locations reserved for ID purposes.

Example

```
#include <htc.h>

__IDLOC(15F0);
/* will store 1, 5, F and 0 in the ID registers*/

void
main (void)
{
}
```

See also

__IDLOC7(), __CONFIG()

__IDLOC7

Synopsis

```
#include <htc.h>

__IDLOC7(a,b,c,d)
```

Description

This macro places data into the device's special locations outside of addressable memory reserved for ID. This would be useful for storage of serial numbers etc.

The macro will attempt to write 7 bits of data to each of the 4 locations reserved for ID purposes.

Example

```
#include <htc.h>

__IDLOC(0x7F,70,1,0x5A);
/* will store 7Fh, 70, 1 and 5Ah in the ID registers */

void
main (void)
{
}
```

Note

Not all devices permit 7 bit programming of the ID locations. Refer to the device datasheet to see whether this macro can be used on your particular device.

See also

__IDLOC(), __CONFIG()

_DELAY, __DELAY_MS, __DELAY_US

Synopsis

```
__delay_ms(x)  // request a delay in milliseconds
__delay_us(x)  // request a delay in microseconds

// request a delay for a number of instruction cycles
void _delay_ms(unsigned long n)
```

Description

The when code calls `_delay(n)`, the code generator will customize and in-lined sequence of code to facilitate a delay of *n* instruction cycles. As this routine is customized for the parameter given, the resultant code produced may differ significantly based on the magnitude of the requested delay.

As it is often more convenient request a delay in time-based terms rather than in cycle counts, the macros `__delay_ms(x)` and `__delay_us(x)` are provided. These macros simply wrap around `_delay(n)` and convert the time based request into instruction cycles based on the system frequency. In order to achieve this, these macros require the prior definition of preprocessor symbol `_XTAL_FREQ`. This symbol should be defined as the oscillator frequency (in Hertz) used by the system.

An error will result if these macros are used without defining this symbol or if the delay period requested is too large.

ABS

Synopsis

```
#include <stdlib.h>

int abs (int j)
```

Description

The **abs()** function returns the absolute value of **j**.

Example

```
#include <stdio.h>
#include <stdlib.h>

void
main (void)
{
    int a = -5;

    printf("The absolute value of %d is %d\n", a, abs(a));
}
```

See Also

labs(), fabs()

Return Value

The absolute value of **j**.

ACOS

Synopsis

```
#include <math.h>

double acos (double f)
```

Description

The **acos()** function implements the inverse of cos(), i.e. it is passed a value in the range -1 to +1, and returns an angle in radians whose cosine is equal to that value.

Example

```
#include <math.h>
#include <stdio.h>

/* Print acos() values for -1 to 1 in degrees. */

void
main (void)
{
    float i, a;

    for(i = -1.0; i < 1.0 ; i += 0.1) {
        a = acos(i)*180.0/3.141592;
        printf("acos(%f) = %f degrees\n", i, a);
    }
}
```

See Also

sin(), cos(), tan(), asin(), atan(), atan2()

Return Value

An angle in radians, in the range 0 to π

ASCTIME

Synopsis

```
#include <time.h>

char * asctime (struct tm * t)
```

Description

The **asctime()** function takes the time broken down into the **struct tm** structure, pointed to by its argument, and returns a 26 character string describing the current date and time in the format:

Sun Sep 16 01:03:52 1973\n\0

Note the *newline* at the end of the string. The width of each field in the string is fixed. The example gets the current time, converts it to a **struct tm** pointer with localtime(), it then converts this to ASCII and prints it. The time() function will need to be provided by the user (see time() for details).

Example

```
#include <stdio.h>
#include <time.h>

void
main (void)
{
    time_t clock;
    struct tm * tp;

    time(&clock);
    tp = localtime(&clock);
    printf("%s", asctime(tp));
}
```

See Also

ctime(), gmtime(), localtime(), time()

Return Value

A pointer to the string.

Note

The example will require the user to provide the time() routine as it cannot be supplied with the compiler.. See time() for more details.

ASIN

Synopsis

```
#include <math.h>

double asin (double f)
```

Description

The **asin()** function implements the converse of sin(), i.e. it is passed a value in the range -1 to +1, and returns an angle in radians whose sine is equal to that value.

Example

```
#include <math.h>
#include <stdio.h>

void
main (void)
{
    float i, a;

    for(i = -1.0; i < 1.0 ; i += 0.1) {
        a = asin(i)*180.0/3.141592;
        printf("asin(%f) = %f degrees\n", i, a);
    }
}
```

See Also

sin(), cos(), tan(), acos(), atan(), atan2()

Return Value

An angle in radians, in the range - π

ASSERT

Synopsis

```
#include <assert.h>

void assert (int e)
```

Description

This macro is used for debugging purposes; the basic method of usage is to place assertions liberally throughout your code at points where correct operation of the code depends upon certain conditions being true initially. An **assert()** routine may be used to ensure at run time that an assumption holds true. For example, the following statement asserts that the pointer tp is not equal to NULL:

assert(tp);

If at run time the expression evaluates to false, the program will abort with a message identifying the source file and line number of the assertion, and the expression used as an argument to it. A fuller discussion of the uses of **assert()** is impossible in limited space, but it is closely linked to methods of proving program correctness.

Example

```
void
ptrfunc (struct xyz * tp)
{
    assert(tp != 0);
}
```

Note

When required for ROM based systems, the underlying routine _fassert(...) will need to be implemented by the user.

ATAN

Synopsis

```
#include <math.h>

double atan (double x)
```

Description

This function returns the arc tangent of its argument, i.e. it returns an angle e in the range - π

Example

```
#include <stdio.h>
#include <math.h>

void
main (void)
{
    printf("%f\n", atan(1.5));
}
```

See Also

sin(), cos(), tan(), asin(), acos(), atan2()

Return Value

The arc tangent of its argument.

ATAN2

Synopsis

```
#include <math.h>

double atan2 (double x, double x)
```

Description

This function returns the arc tangent of `y/x`.

Example

```
#include <stdio.h>
#include <math.h>

void
main (void)
{
    printf("%f\n", atan2(10.0, -10.0));
}
```

See Also

sin(), cos(), tan(), asin(), acos(), atan()

Return Value

The arc tangent of `y/x`.

ATOF

Synopsis

```
#include <stdlib.h>

double atof (const char * s)
```

Description

The **atof()** function scans the character string passed to it, skipping leading blanks. It then converts an ASCII representation of a number to a double. The number may be in decimal, normal floating point or scientific notation.

Example

```
#include <stdlib.h>
#include <stdio.h>

void
main (void)
{
    char buf[80];
    double i;

    gets(buf);
    i = atof(buf);
    printf("Read %s: converted to %f\n", buf, i);
}
```

See Also

atoi(), atol(), strtod()

Return Value

A double precision floating point number. If no number is found in the string, 0.0 will be returned.

ATOI

Synopsis

```
#include <stdlib.h>

int atoi (const char * s)
```

Description

The **atoi()** function scans the character string passed to it, skipping leading blanks and reading an optional sign. It then converts an ASCII representation of a decimal number to an integer.

Example

```
#include <stdlib.h>
#include <stdio.h>

void
main (void)
{
    char buf[80];
    int i;

    gets(buf);
    i = atoi(buf);
    printf("Read %s: converted to %d\n", buf, i);
}
```

See Also

xtoi(), atof(), atol()

Return Value

A signed integer. If no number is found in the string, 0 will be returned.

ATOL

Synopsis

```
#include <stdlib.h>

long atol (const char * s)
```

Description

The **atol()** function scans the character string passed to it, skipping leading blanks. It then converts an ASCII representation of a decimal number to a long integer.

Example

```
#include <stdlib.h>
#include <stdio.h>

void
main (void)
{
    char buf[80];
    long i;

    gets(buf);
    i = atol(buf);
    printf("Read %s: converted to %ld\n", buf, i);
}
```

See Also

atoi(), atof()

Return Value

A long integer. If no number is found in the string, 0 will be returned.

BSEARCH

Synopsis

```
#include <stdlib.h>

void * bsearch (const void * key, void * base, size_t n_memb,
size_t size, int (*compar)(const void *, const void *))
```

Description

The **bsearch()** function searches a sorted array for an element matching a particular key. It uses a binary search algorithm, calling the function pointed to by **compar** to compare elements in the array.

Example

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

struct value {
    char name[40];
    int value;
} values[100];

int
val_cmp (const void * p1, const void * p2)
{
    return strcmp(((const struct value *)p1)->name,
                  ((const struct value *)p2)->name);
}

void
main (void)
{
    char inbuf[80];
    int i;
    struct value * vp;
```

```
        i = 0;
        while(gets(inbuf)) {
            sscanf(inbuf,"%s %d", values[i].name, &values[i].value);
            i++;
        }
        qsort(values, i, sizeof values[0], val_cmp);
        vp = bsearch("fred", values, i, sizeof values[0], val_cmp);
        if(!vp)
            printf("Item 'fred' was not found\n");
        else
            printf("Item 'fred' has value %d\n", vp->value);
    }
```

See Also

qsort()

Return Value

A pointer to the matched array element (if there is more than one matching element, any of these may be returned). If no match is found, a null pointer is returned.

Note

The comparison function must have the correct prototype.

CEIL

Synopsis

```
#include <math.h>

double ceil (double f)
```

Description

This routine returns the smallest whole number not less than **f**.

Example

```
#include <stdio.h>
#include <math.h>

void
main (void)
{
    double j;

    scanf("%lf", &j);
    printf("The ceiling of %lf is %lf\n", j, ceil(j));
}
```

CGETS

Synopsis

```
#include <conio.h>

char * cgets (char * s)
```

Description

The **cgets()** function will read one line of input from the console into the buffer passed as an argument. It does so by repeated calls to getche(). As characters are read, they are buffered, with *backspace* deleting the previously typed character, and *ctrl-U* deleting the entire line typed so far. Other characters are placed in the buffer, with a *carriage return* or *line feed (newline)* terminating the function. The collected string is null terminated.

Example

```
#include <conio.h>
#include <string.h>

char buffer[80];

void
main (void)
{
    for(;;) {
        cgets(buffer);
        if(strcmp(buffer, "exit") == 0)
            break;
        cputs("Type 'exit' to finish\n");
    }
}
```

See Also

getch(), getche(), putch(), cputs()

Return Value

The return value is the character pointer passed as the sole argument.

CLRWDT

Synopsis

```
#include <htc.h>

CLRWDT();
```

Description

This macro is used to clear the device's internal watchdog timer.

Example

```
#include <htc.h>

void
main (void)
{
    WDTCON=1;
      /* enable the WDT */

    CLRWDT();
}
```

COS

Synopsis

```
#include <math.h>

double cos (double f)
```

Description

This function yields the cosine of its argument, which is an angle in radians. The cosine is calculated by expansion of a polynomial series approximation.

Example

```
#include <math.h>
#include <stdio.h>

#define C 3.141592/180.0

void
main (void)
{
    double i;

    for(i = 0 ; i <= 180.0 ; i += 10)
        printf("sin(%3.0f) = %f, cos = %f\n", i, sin(i*C), cos(i*C));
}
```

See Also

sin(), tan(), asin(), acos(), atan(), atan2()

Return Value

A double in the range -1 to +1.

COSH, SINH, TANH

Synopsis

```
#include <math.h>

double cosh (double f)
double sinh (double f)
double tanh (double f)
```

Description

These functions are the implement hyperbolic equivalents of the trigonometric functions; cos(), sin() and tan().

Example

```
#include <stdio.h>
#include <math.h>

void
main (void)
{
    printf("%f\n", cosh(1.5));
    printf("%f\n", sinh(1.5));
    printf("%f\n", tanh(1.5));
}
```

Return Value

The function **cosh()** returns the hyperbolic cosine value.
The function **sinh()** returns the hyperbolic sine value.
The function **tanh()** returns the hyperbolic tangent value.

CPUTS

Synopsis

```
#include <conio.h>

void cputs (const char * s)
```

Description

The **cputs()** function writes its argument string to the console, outputting *carriage returns* before each *newline* in the string. It calls putch() repeatedly. On a hosted system **cputs()** differs from puts() in that it writes to the console directly, rather than using file I/O. In an embedded system **cputs()** and puts() are equivalent.

Example

```
#include <conio.h>
#include <string.h>

char buffer[80];

void
main (void)
{
    for(;;) {
        cgets(buffer);
        if(strcmp(buffer, "exit") == 0)
            break;
        cputs("Type 'exit' to finish\n");
    }
}
```

See Also

cputs(), puts(), putch()

CTIME

Synopsis

```
#include <time.h>

char * ctime (time_t * t)
```

Description

The **ctime()** function converts the time in seconds pointed to by its argument to a string of the same form as described for asctime(). Thus the example program prints the current time and date.

Example

```
#include <stdio.h>
#include <time.h>

void
main (void)
{
    time_t clock;

    time(&clock);
    printf("%s", ctime(&clock));
}
```

See Also

gmtime(), localtime(), asctime(), time()

Return Value

A pointer to the string.

Note

The example will require the user to provide the time() routine as one cannot be supplied with the compiler. See time() for more detail.

DI, EI

Synopsis

```
#include <htc.h>

void ei (void)
void di (void)
```

Description

The **di()** and **ei()** routines disable and re-enable interrupts respectively. These are implemented as macros defined in **PIC.h**. The example shows the use of **ei()** and **di()** around access to a long variable that is modified during an interrupt. If this was not done, it would be possible to return an incorrect value, if the interrupt occurred between accesses to successive words of the count value.

Example

```
#include <htc.h>

long count;

void
interrupt tick (void)
{
    count++;
}

long
getticks (void)
{
    long val;    /* Disable interrupts around access
                    to count, to ensure consistency.*/
    di();
    val = count;
    ei();
    return val;
}
```

DIV

Synopsis

```
#include <stdlib.h>

div_t div (int numer, int demon)
```

Description

The **div()** function computes the quotient and remainder of the numerator divided by the denominator.

Example

```
#include <stdlib.h>
#include <stdio.h>

void
main (void)
{
    div_t x;

    x = div(12345, 66);
    printf("quotient = %d, remainder = %d\n", x.quot, x.rem);
}
```

See Also

udiv(), ldiv(), uldiv()

Return Value

Returns the quotient and remainder into the **div_t** structure.

EEPROM_READ, EEPROM_WRITE

Synopsis

```
#include <htc.h>

unsigned char eeprom_read (unsigned char addr);
void eeprom_write (unsigned char addr, unsigned char value);
```

Description

These function allow access to the on-chip eeprom (when present). The eeprom is not in the directly-accessible memory space and a special byte sequence is loaded to the eeprom control registers to access the device. Writing a value to the eeprom is a slow process and the **eeprom_write()** function polls the appropriate registers to ensure that any previous writes have completed before writing the next datum. Reading data is completed in the one cycle and no polling is necessary to check for a read completion.

Example

```
#include <htc.h>

void
main (void)
{
    unsigned char data;
    unsigned char address;

    address = 0x10;
    data = eeprom_read(address);
}
```

Note

It may be necessary to poll the eeprom registers to ensure that the write has completed if an **eeprom_write()** call is immediately followed by an **eeprom_read()**. The global interrupt enable bit (GIE) is now restored by the **eeprom_write()** routine. The EEIF interrupt flag is not reset by this function.

EVAL_POLY

Synopsis

```
#include <math.h>

double eval_poly (double x, const double * d, int n)
```

Description

The **eval_poly()** function evaluates a polynomial, whose coefficients are contained in the array **d**, at **x**, for example:

```
y = x*x*d2 + x*d1 + d0.
```

The order of the polynomial is passed in **n**.

Example

```
#include <stdio.h>
#include <math.h>

void
main (void)
{
    double x, y;
    double d[3] = {1.1, 3.5, 2.7};

    x = 2.2;
    y = eval_poly(x, d, 2);
    printf("The polynomial evaluated at %f is %f\n", x, y);
}
```

Return Value

A double value, being the polynomial evaluated at **x**.

EXP

Synopsis

```
#include <math.h>

double exp (double f)
```

Description

The **exp()** routine returns the exponential function of its argument, i.e. e to the power of **f**.

Example

```
#include <math.h>
#include <stdio.h>

void
main (void)
{
    double f;

    for(f = 0.0 ; f <= 5 ; f += 1.0)
        printf("e to %1.0f = %f\n", f, exp(f));
}
```

See Also

log(), log10(), pow()

FABS

Synopsis

```
#include <math.h>

double fabs (double f)
```

Description

This routine returns the absolute value of its double argument.

Example

```
#include <stdio.h>
#include <math.h>

void
main (void)
{
    printf("%f %f\n", fabs(1.5), fabs(-1.5));
}
```

See Also

abs(), labs()

FLASH_COPY

Synopsis

```
#include <htc.h>

void flash_copy(const unsigned char * source_addr,
     unsigned char length, unsigned short dest_addr);
```

Description

This utility function is useful for copying a large section of memory to a new location in flash memory.

Note it is only applicable to those devices which have an internal set of flash buffer registers.

When the function is called, it needs to be supplied with a **const pointer** to the source address of the data to copy. The pointer may point to a valid address in either RAM or flash memory.

A length parameter must be specified to indicate the number of words of the data to be copied.

Finally the flash address where this data is destined must be specified.

Example

```
#include <htc.h>

const unsigned char ROMSTRING[] = "0123456789ABCDEF";

void
main (void){
    const unsigned char * ptr = &ROMSTRING[0];
    flash_copy( ptr, 5, 0x70 );
}
```

See Also

EEPROM_READ, EEPROM_WRITE, FLASH_READ, FLASH_WRITE

Note

This function is only applicable to those devices which use internal buffer registers when writing to flash.

Ensure that the function does not attempt to overwrite the section of program memory from which it is currently executing, and extreme caution must be exercised if modifying code at the device's reset or interrupt vectors. A reset or interrupt must not be triggered while this sector is in erasure.

FLASH_ERASE(), FLASH_READ()

Synopsis

```
#include <htc.h>

void flash_erase (unsigned short addr);
unsigned int flash_read (unsigned short addr);
```

Description

These functions allow access to the flash memory of the microcontroller (if supported).

Reading from the flash memory can be done one word at a time with use of the **flash_read()** function. **flash_read()** returns the data value found at the specified word address in flash memory.

Entire sectors of 32 words can be restored to an unprogrammed state (value=**FF**) with use of the **flash_erase()** function. Specifying an address to the **flash_erase()** function, will erase all 32 words in the sector that contains the given address.

Example

```
#include <htc.h>

void
main (void)
{
    unsigned int data;
    unsigned short address=0x1000;

    data = flash_read(address);

    flash_erase(address);

}
```

Return Value

flash_read() returns the data found at the given address, as an unsigned int.

Note

The functions **flash_erase()** and **flash_read()** are only available on those devices that support such functionality.

FMOD

Synopsis

```
#include <math.h>

double fmod (double x, double y)
```

Description

The function **fmod** returns the remainder of **x/y** as a floating point quantity.

Example

```
#include <math.h>

void
main (void)
{
    double rem, x;

    x = 12.34;
    rem = fmod(x, 2.1);
}
```

Return Value

The floating-point remainder of x/y.

FLOOR

Synopsis

```
#include <math.h>

double floor (double f)
```

Description

This routine returns the largest whole number not greater than **f**.

Example

```
#include <stdio.h>
#include <math.h>

void
main (void)
{
    printf("%f\n", floor( 1.5 ));
    printf("%f\n", floor( -1.5));
}
```

FREXP

Synopsis

```
#include <math.h>

double frexp (double f, int * p)
```

Description

The **frexp()** function breaks a floating point number into a normalized fraction and an integral power of 2. The integer is stored into the **int** object pointed to by **p**. Its return value x is in the interval (0.5, 1.0) or zero, and **f** equals x times 2 raised to the power stored in ***p**. If **f** is zero, both parts of the result are zero.

Example

```
#include <math.h>
#include <stdio.h>

void
main (void)
{
    double f;
    int i;

    f = frexp(23456.34, &i);
    printf("23456.34 = %f * 2^%d\n", f, i);
}
```

See Also

ldexp()

FTOA

Synopsis

```
#include <stdlib.h>

char * ftoa (float f, int * status)
```

Description

The function **ftoa** converts the contents of **f** into a string which is stored into a buffer which is then return.

Example

```
#include <stdlib.h>
#include <stdio.h>

void
main (void)
{
    char * buf;
    float input = 12.34;
    int status;
    buf = ftoa(input, &status);
    printf("The buffer holds %s\n", buf);
}
```

See Also

strtol(), itoa(), utoa(), ultoa()

Return Value

This routine returns a reference to the buffer into which the result is written.

GETCH, GETCHE

Synopsis

```
#include <conio.h>

char getch (void)
char getche (void)
```

Description

The **getch()** function reads a single character from the console keyboard and returns it without echoing. The **getche()** function is similar but does echo the character typed.

In an embedded system, the source of characters is defined by the particular routines supplied. By default, the library contains a version of **getch()** that will interface to the Lucifer Debugger. The user should supply an appropriate routine if another source is desired, e.g. a serial port.

The module *getch.c* in the SOURCES directory contains model versions of all the console I/O routines. Other modules may also be supplied, e.g. *ser180.c* has routines for the serial port in a Z180.

Example

```
#include <conio.h>

void
main (void)
{
    char c;

    while((c = getche()) != '\n')
        continue;
}
```

See Also

cgets(), cputs(), ungetch()

GETCHAR

Synopsis

```
#include <stdio.h>

int getchar (void)
```

Description

The **getchar()** routine is a getc(stdin) operation. It is a macro defined in **stdio.h**. Note that under normal circumstances **getchar()** will NOT return unless a *carriage return* has been typed on the console. To get a single character immediately from the console, use the function getch().

Example

```
#include <stdio.h>

void
main (void)
{
    int c;

    while((c = getchar()) != EOF)
        putchar(c);
}
```

See Also

getc(), fgetc(), freopen(), fclose()

Note

This routine is not usable in a ROM based system.

GETS

Synopsis

```
#include <stdio.h>

char * gets (char * s)
```

Description

The **gets()** function reads a line from standard input into the buffer at **s**, deleting the *newline* (cf. fgets()). The buffer is null terminated. In an embedded system, **gets()** is equivalent to cgets(), and results in getche() being called repeatedly to get characters. Editing (with *backspace*) is available.

Example

```
#include <stdio.h>

void
main (void)
{
    char buf[80];

    printf("Type a line: ");
    if(gets(buf))
        puts(buf);
}
```

See Also

fgets(), freopen(), puts()

Return Value

It returns its argument, or NULL on end-of-file.

GET_CAL_DATA

Synopsis

```
#include <htc.h>

double get_cal_data (const unsigned char * code_ptr)
```

Description

This function returns the 32-bit floating point calibration data from the PIC 14000 calibration space. Only use this function to access KREF, KBG, VHTHERM and KTC (that is, the 32-bit floating point parameters). FOSC and TWDT can be accessed directly as they are bytes.

Example

```
#include <htc.h>

void
main (void)
{
    double x;
    unsigned char y;

        /* Get the slope reference ratio. */
    x = get_cal_data(KREF);

        /* Get the WDT time-out. */
    y =TWDT;
}
```

Return Value

The value of the calibration parameter

Note

This function can only be used on the PIC 14000.

GMTIME

Synopsis

```
#include <time.h>

struct tm * gmtime (time_t * t)
```

Description

This function converts the time pointed to by **t** which is in seconds since 00:00:00 on Jan 1, 1970, into a broken down time stored in a structure as defined in **time.h**. The structure is defined in the 'Data Types' section.

Example

```
#include <stdio.h>
#include <time.h>

void
main (void)
{
    time_t clock;
    struct tm * tp;

    time(&clock);
    tp = gmtime(&clock);
    printf("It's %d in London\n", tp->tm_year+1900);
}
```

See Also

ctime(), asctime(), time(), localtime()

Return Value

Returns a structure of type **tm**.

Note

The example will require the user to provide the time() routine as one cannot be supplied with the compiler. See time() for more detail.

ISALNUM, ISALPHA, ISDIGIT, ISLOWER et. al.

Synopsis

```
#include <ctype.h>

int isalnum (char c)
int isalpha (char c)
int isascii (char c)
int iscntrl (char c)
int isdigit (char c)
int islower (char c)
int isprint (char c)
int isgraph (char c)
int ispunct (char c)
int isspace (char c)
int isupper (char c)
int isxdigit(char c)
```

Description

These macros, defined in **ctype.h**, test the supplied character for membership in one of several overlapping groups of characters. Note that all except **isascii()** are defined for **c**, if **isascii(c)** is true or if c = EOF.

isalnum(c)	c is in 0-9 or a-z or A-Z
isalpha(c)	c is in A-Z or a-z
isascii(c)	c is a 7 bit ascii character
iscntrl(c)	c is a control character
isdigit(c)	c is a decimal digit
islower(c)	c is in a-z
isprint(c)	c is a printing char
isgraph(c)	c is a non-space printable character
ispunct(c)	c is not alphanumeric
isspace(c)	c is a space, tab or newline
isupper(c)	c is in A-Z
isxdigit(c)	c is in 0-9 or a-f or A-F

Example

```
#include <ctype.h>
#include <stdio.h>

void
main (void)
{
    char buf[80];
    int i;

    gets(buf);
    i = 0;
    while(isalnum(buf[i]))
        i++;
    buf[i] = 0;
    printf("'%s' is the word\n", buf);
}
```

See Also

toupper(), tolower(), toascii()

ISDIG

Synopsis

```
#include <ctype.h>

int isdig (int c)
```

Description

The **isdig**() function tests the input character `c` to see if is a decimal digit (0 – 9) and returns true is this is the case; false otherwise.

Example

```
#include <ctype.h>

void
main (void)
{
    char buf[] = "1998a";
    if(isdig(buf[0]))
        printf("valid type detected\n");
}
```

See Also

isdigit() (listed un isalnum())

Return Value

Zero if the character is a decimal digit; a non-zero value otherwise.

ITOA

Synopsis

```
#include <stdlib.h>

char * itoa (char * buf, int val, int base)
```

Description

The function **itoa** converts the contents of **val** into a string which is stored into **buf**. The conversion is performed according to the radix specified in **base**. **buf** is assumed to reference a buffer which has sufficent space allocated to it.

Example

```
#include <stdlib.h>
#include <stdio.h>

void
main (void)
{
    char buf[10];
    itoa(buf, 1234, 16);
    printf("The buffer holds %s\n", buf);
}
```

See Also

strtol(), utoa(), ltoa(), ultoa()

Return Value

This routine returns a copy of the buffer into which the result is written.

LABS

Synopsis

```
#include <stdlib.h>

int labs (long int j)
```

Description

The **labs()** function returns the absolute value of long value **j**.

Example

```
#include <stdio.h>
#include <stdlib.h>

void
main (void)
{
    long int a = -5;

    printf("The absolute value of %ld is %ld\n", a, labs(a));
}
```

See Also

abs()

Return Value

The absolute value of **j**.

LDEXP

Synopsis

```
#include <math.h>

double ldexp (double f, int i)
```

Description

The **ldexp()** function performs the inverse of frexp() operation; the integer **i** is added to the exponent of the floating point **f** and the resultant returned.

Example

```
#include <math.h>
#include <stdio.h>

void
main (void)
{
    double f;

    f = ldexp(1.0, 10);
    printf("1.0 * 2^10 = %f\n", f);
}
```

See Also

frexp()

Return Value

The return value is the integer **i** added to the exponent of the floating point value **f**.

LDIV

Synopsis

```
#include <stdlib.h>

ldiv_t ldiv (long number, long denom)
```

Description

The **ldiv()** routine divides the numerator by the denominator, computing the quotient and the remainder. The sign of the quotient is the same as that of the mathematical quotient. Its absolute value is the largest integer which is less than the absolute value of the mathematical quotient.

The **ldiv()** function is similar to the div() function, the difference being that the arguments and the members of the returned structure are all of type **long int**.

Example

```
#include <stdlib.h>
#include <stdio.h>

void
main (void)
{
    ldiv_t lt;

    lt = ldiv(1234567, 12345);
    printf("Quotient = %ld, remainder = %ld\n", lt.quot, lt.rem);
}
```

See Also

div(), uldiv(), udiv()

Return Value

Returns a structure of type **ldiv_t**

LOCALTIME

Synopsis

```
#include <time.h>

struct tm * localtime (time_t * t)
```

Description

The **localtime()** function converts the time pointed to by **t** which is in seconds since 00:00:00 on Jan 1, 1970, into a broken down time stored in a structure as defined in **time.h**. The routine **localtime()** takes into account the contents of the global integer time_zone. This should contain the number of minutes that the local time zone is *westward* of Greenwich. On systems where it is not possible to predetermine this value, **localtime()** will return the same result as **gmtime()**.

Example

```
#include <stdio.h>
#include <time.h>

    char * wday[] = {
        "Sunday", "Monday", "Tuesday", "Wednesday",
        "Thursday", "Friday", "Saturday"
};

void
main (void)
{
    time_t clock;
    struct tm * tp;

    time(&clock);
    tp = localtime(&clock);
    printf("Today is %s\n", wday[tp->tm_wday]);
}
```

See Also

ctime(), asctime(), time()

Return Value

Returns a structure of type **tm**.

Note

The example will require the user to provide the time() routine as one cannot be supplied with the compiler. See time() for more detail.

LOG, LOG10

Synopsis

```
#include <math.h>

double log (double f)
double log10 (double f)
```

Description

The **log()** function returns the natural logarithm of **f**. The function **log10()** returns the logarithm to base 10 of **f**.

Example

```
#include <math.h>
#include <stdio.h>

void
main (void)
{
    double f;

    for(f = 1.0 ; f <= 10.0 ; f += 1.0)
        printf("log(%1.0f) = %f\n", f, log(f));
}
```

See Also

exp(), pow()

Return Value

Zero if the argument is negative.

LONGJMP

Synopsis

```
#include <setjmp.h>

void longjmp (jmp_buf buf, int val)
```

Description

The **longjmp()** function, in conjunction with setjmp(), provides a mechanism for non-local goto's. To use this facility, setjmp() should be called with a **jmp_buf** argument in some outer level function. The call from setjmp() will return 0.

To return to this level of execution, **longjmp()** may be called with the same **jmp_buf** argument from an inner level of execution. ***Note*** however that the function which called setjmp() must still be active when **longjmp()** is called. Breach of this rule will cause disaster, due to the use of a stack containing invalid data. The **val** argument to **longjmp()** will be the value apparently returned from the setjmp(). This should normally be non-zero, to distinguish it from the genuine setjmp() call.

Example

```
#include <stdio.h>
#include <setjmp.h>
#include <stdlib.h>

jmp_buf jb;

void
inner (void)
{
    longjmp(jb, 5);
}

void
main (void)
{
    int i;
```

```
        if(i = setjmp(jb)) {
            printf("setjmp returned %d\n", i);
            exit(0);
        }
        printf("setjmp returned 0 - good\n");
        printf("calling inner...\n");
        inner();
        printf("inner returned - bad!\n");
    }
```

See Also

setjmp()

Return Value

The **longjmp()** routine never returns.

Note

The function which called setjmp() must still be active when **longjmp()** is called. Breach of this rule will cause disaster, due to the use of a stack containing invalid data.

LTOA

Synopsis

```
#include <stdlib.h>

char * ltoa (char * buf, long val, int base)
```

Description

The function **itoa** converts the contents of **val** into a string which is stored into **buf**. The conversion is performed according to the radix specified in **base**. **buf** is assumed to reference a buffer which has sufficent space allocated to it.

Example

```
#include <stdlib.h>
#include <stdio.h>

void
main (void)
{
    char buf[10];
    utoi(buf, 12345678L, 16);
    printf("The buffer holds %s\n", buf);
}
```

See Also

strtol(), itoa(), utoa(), ultoa()

Return Value

This routine returns a copy of the buffer into which the result is written.

MEMCHR

Synopsis

```
#include <string.h>

/* For baseline and midrange processors */
const void * memchr (const void * block, int val, size_t length)

/* For high-end processors */
void * memchr (const void * block, int val, size_t length)
```

Description

The **memchr()** function is similar to strchr() except that instead of searching null terminated strings, it searches a block of memory specified by length for a particular byte. Its arguments are a pointer to the memory to be searched, the value of the byte to be searched for, and the length of the block. A pointer to the first occurrence of that byte in the block is returned.

Example

```
#include <string.h>
#include <stdio.h>

unsigned int ary[] = {1, 5, 0x6789, 0x23};

void
main (void)
{
    char * cp;

    cp = memchr(ary, 0x89, sizeof ary);
    if(!cp)
        printf("not found\n");
    else
        printf("Found at offset %u\n", cp - (char *)ary);
}
```

See Also

strchr()

Return Value

A pointer to the first byte matching the argument if one exists; NULL otherwise.

MEMCMP

Synopsis

```
#include <string.h>

int memcmp (const void * s1, const void * s2, size_t n)
```

Description

The **memcmp()** function compares two blocks of memory, of length **n**, and returns a signed value similar to strncmp(). Unlike strncmp() the comparison does not stop on a null character.

Example

```
#include <stdio.h>
#include <string.h>

void
main (void)
{
    int buf[10], cow[10], i;

    buf[0] = 1;
    buf[2] = 4;
    cow[0] = 1;
    cow[2] = 5;
    buf[1] = 3;
    cow[1] = 3;
    i = memcmp(buf, cow, 3*sizeof(int));
    if(i < 0)
        printf("less than\n");
    else if(i > 0)
        printf("Greater than\n");
    else
        printf("Equal\n");
}
```

See Also

strncpy(), strncmp(), strchr(), memset(), memchr()

Return Value

Returns negative one, zero or one, depending on whether **s1** points to string which is less than, equal to or greater than the string pointed to by **s2** in the collating sequence.

MEMCPY

Synopsis

```
#include <string.h>

/* For baseline and midrange processors */
void * memcpy (void * d, const void * s, size_t n)

/* For high-end processors */
far void * memcpy (far void * d, const void * s, size_t n)
```

Description

The **memcpy()** function copies **n** bytes of memory starting from the location pointed to by **s** to the block of memory pointed to by **d**. The result of copying overlapping blocks is undefined. The **memcpy()** function differs from strcpy() in that it copies a specified number of bytes, rather than all bytes up to a null terminator.

Example

```
#include <string.h>
#include <stdio.h>

void
main (void)
{
    char buf[80];

    memset(buf, 0, sizeof buf);
    memcpy(buf, "a partial string", 10);
    printf("buf = '%s'\n", buf);
}
```

See Also

strncpy(), strncmp(), strchr(), memset()

Return Value

The **memcpy()** routine returns its first argument.

MEMMOVE

Synopsis

```
#include <string.h>

/* For baseline and midrange processors */
void * memmove (void * s1, const void * s2, size_t n)

/* For high-end processors */
far void * memmove (far void * s1, const void * s2, size_t n)
```

Description

The **memmove()** function is similar to the function memcpy() except copying of overlapping blocks is handled correctly. That is, it will copy forwards or backwards as appropriate to correctly copy one block to another that overlaps it.

See Also

strncpy(), strncmp(), strchr(), memcpy()

Return Value

The function **memmove()** returns its first argument.

MEMSET

Synopsis

```
#include <string.h>

/* For baseline and midrange processors */
void * memset (void * s, int c, size_t n)

/* For high-end processors */
far void * memset (far void * s, int c, size_t n)
```

Description

The **memset()** function fills **n** bytes of memory starting at the location pointed to by **s** with the byte **c**.

Example

```
#include <string.h>
#include <stdio.h>

void
main (void)
{
    char abuf[20];

    strcpy(abuf, "This is a string");
    memset(abuf, 'x', 5);
    printf("buf = '%s'\n", abuf);
}
```

See Also

strncpy(), strncmp(), strchr(), memcpy(), memchr()

MKTIME

Synopsis

```
#include <time.h>

time_t mktime (struct tm * tmptr)
```

Description

The **mktime()** function converts the local calendar time referenced by the tm structure pointer **tmptr** into a time being the number of seconds passed since Jan 1st 1970, or -1 if the time cannot be represented.

Example

```
#include <time.h>
#include <stdio.h>

void
main (void)
{
    struct tm birthday;

    birthday.tm_year = 1955;
    birthday.tm_mon = 2;
    birthday.tm_mday = 24;
    birthday.tm_hour = birthday.tm_min = birthday.tm_sec = 0;
    printf("you have been alive approximately %ld seconds\n",
 mktime(&birthday));
}
```

See Also

ctime(), asctime()

Return Value

The time contained in the **tm** structure represented as the number of seconds since the 1970 Epoch, or -1 if this time cannot be represented.

MODF

Synopsis

```
#include <math.h>

double modf (double value, double * iptr)
```

Description

The **modf()** function splits the argument **value** into integral and fractional parts, each having the same sign as **value**. For example, -3.17 would be split into the integral part (-3) and the fractional part (-0.17).

The integral part is stored as a double in the object pointed to by **iptr**.

Example

```
#include <math.h>
#include <stdio.h>

void
main (void)
{
    double i_val, f_val;

    f_val = modf( -3.17, &i_val);
}
```

Return Value

The signed fractional part of **value**.

PERSIST_CHECK, PERSIST_VALIDATE

Synopsis

```
#include <sys.h>

int persist_check (int flag)
void persist_validate (void)
```

Description

The **persist_check()** function is used with non-volatile RAM variables, declared with the persistent qualifier. It tests the nvram area, using a magic number stored in a hidden variable by a previous call to **persist_validate()** and a checksum also calculated by **persist_validate()**. If the magic number and checksum are correct, it returns true (non-zero). If either are incorrect, it returns zero. In this case it will optionally zero out and re-validate the non-volatile RAM area (by calling **persist_validate()**). This is done if the flag argument is true.

The **persist_validate()** routine should be called after each change to a persistent variable. It will set up the magic number and recalculate the checksum.

Example

```
#include <sys.h>
#include <stdio.h>

persistent long reset_count;

void
main (void)
{
    if(!persist_check(1))
        printf("Reset count invalid - zeroed\n");
    else
        printf("Reset number %ld\n", reset_count);
    reset_count++;          /* update count */
    persist_validate();     /* and checksum */
    for(;;)
        continue;       /* sleep until next reset */
```

```
}
```

Return Value

FALSE (zero) if the NVRAM area is invalid; TRUE (non-zero) if the NVRAM area is valid.

POW

Synopsis

```
#include <math.h>

double pow (double f, double p)
```

Description

The **pow()** function raises its first argument, **f**, to the power **p**.

Example

```
#include <math.h>
#include <stdio.h>

void
main (void)
{
    double f;

    for(f = 1.0 ; f <= 10.0 ; f += 1.0)
        printf("pow(2, %1.0f) = %f\n", f, pow(2, f));
}
```

See Also

log(), log10(), exp()

Return Value

f to the power of **p**.

PRINTF

Synopsis

```
#include <stdio.h>

unsigned char printf (const char * fmt, ...)
```

Description

The **printf()** function is a formatted output routine, operating on stdout. There are corresponding routines operating into a string buffer (sprintf()). The **printf()** routine is passed a format string, followed by a list of zero or more arguments. In the format string are conversion specifications, each of which is used to print out one of the argument list values.

Each conversion specification is of the form **%m.nc** where the percent symbol **%** introduces a conversion, followed by an optional width specification **m**. The **n** specification is an optional precision specification (introduced by the dot) and **c** is a letter specifying the type of the conversion. Field widths and precision are only supported on the midrange and high-end processors, with the precision specification only applicable to **%s**.

If the character ***** is used in place of a decimal constant, e.g. in the format **%*d**, then one integer argument will be taken from the list to provide that value. The types of conversion for the Baseline series are:

o x X u d

Integer conversion - in radices 8, 16, 16, 10 and 10 respectively. The conversion is signed in the case of **d**, unsigned otherwise. The precision value is the total number of digits to print, and may be used to force leading zeroes. E.g. **%8.4x** will print at least 4 hex digits in an 8 wide field. The letter **X** prints out hexadecimal numbers using the upper case letters *A-F* rather than *a-f* as would be printed when using **x**. When the alternate format is specified, a leading zero will be supplied for the octal format, and a leading 0x or 0X for the hex format.

s

Print a string - the value argument is assumed to be a character pointer. At most **n** characters from the string will be printed, in a field **m** characters wide.

c

The argument is assumed to be a single character and is printed literally.

Any other characters used as conversion specifications will be printed. Thus **%** will produce a single percent sign.

For the Midrange and High-end series, the types of conversions are as for the Baseline with the addition of:

l

Long integer conversion - Preceding the integer conversion key letter with an **l** indicates that the argument list is long.

f

Floating point - **m** is the total width and **n** is the number of digits after the decimal point. If **n** is omitted it defaults to 6. If the precision is zero, the decimal point will be omitted unless the alternate format is specified.

Example

```
printf("Total = %4d%", 23)
       yields 'Total =   23%'

printf("Size is %lx" , size)
       where size is a long, prints size
       as hexadecimal.

Note: precision number is only available when using Midrange
and High-end processors when using the %s  placeholder.
printf("Name = %.8s", "a1234567890")
       yields 'Name = a1234567'

Note: the variable width number is only available when using
Midrange and High-end processors placeholder.
printf("xx%*d", 3, 4)
       yields 'xx  4'

/* vprintf example */

#include <stdio.h>

int
error (char * s, ...)
{
    va_list ap;

    va_start(ap, s);
    printf("Error: ");
    vprintf(s, ap);
```

```
        putchar('\n');
        va_end(ap);
    }

    void
    main (void)
    {
        int i;

        i = 3;
        error("testing 1 2 %d", i);
    }
```

See Also

sprintf()

Return Value

The **printf()** routine returns the number of characters written to stdout.
NB The return value is a char, NOT an int.

Note

Certain features of printf are only available for the midrange and high-end processors. Read the description for details. Printing floating point numbers requires that the float to be printed be no larger than the largest possible long integer. In order to use long or float formats, the appropriate supplemental library must be included. See the description on the PICC -L option and the HPDPIC Options/Long formats in printf menu for more details.

PUTCH

Synopsis

```
#include <conio.h>

void putch (char c)
```

Description

The **putch()** function outputs the character **c** to the console screen, prepending a *carriage return* if the character is a *newline*. In a CP/M or MS-DOS system this will use one of the system I/O calls. In an embedded system this routine, and associated others, will be defined in a hardware dependent way. The standard **putch()** routines in the embedded library interface either to a serial port or to the Lucifer Debugger.

Example

```
#include <conio.h>

char * x = "This is a string";

void
main (void)
{
    char * cp;

    cp = x;
    while(*x)
        putch(*x++);
    putch('\n');
}
```

See Also

cgets(), cputs(), getch(), getche()

PUTCHAR

Synopsis

```
#include <stdio.h>

int putchar (int c)
```

Description

The **putchar()** function is a putc() operation on stdout, defined in **stdio.h**.

Example

```
#include <stdio.h>

char * x = "This is a string";

void
main (void)
{
    char * cp;

    cp = x;
    while(*x)
        putchar(*x++);
    putchar('\n');
}
```

See Also

putc(), getc(), freopen(), fclose()

Return Value

The character passed as argument, or EOF if an error occurred.

Note

This routine is not usable in a ROM based system.

PUTS

Synopsis

```
#include <stdio.h>

int puts (const char * s)
```

Description

The **puts()** function writes the string **s** to the *stdout stream*, appending a *newline*. The null character terminating the string is not copied.

Example

```
#include <stdio.h>

void
main (void)
{
    puts("Hello, world!");
}
```

See Also

fputs(), gets(), freopen(), fclose()

Return Value

EOF is returned on error; zero otherwise.

QSORT

Synopsis

```
#include <stdlib.h>

void qsort (void * base, size_t nel, size_t width,
int (*func)(const void *, const void *))
```

Description

The **qsort()** function is an implementation of the quicksort algorithm. It sorts an array of **nel** items, each of length **width** bytes, located contiguously in memory at **base**. The argument **func** is a pointer to a function used by **qsort()** to compare items. It calls **func** with pointers to two items to be compared. If the first item is considered to be greater than, equal to or less than the second then **func** should return a value greater than zero, equal to zero or less than zero respectively.

Example

```
#include <stdio.h>
#include <stdlib.h>

int aray[] = {
    567, 23, 456, 1024, 17, 567, 66
};

int
sortem (const void * p1, const void * p2)
{
    return *(int *)p1 - *(int *)p2;
}

void
main (void)
{
    register int i;
```

```
        qsort(aray, sizeof aray/sizeof aray[0],
            sizeof aray[0], sortem);
        for(i = 0 ; i != sizeof aray/sizeof aray[0] ; i++)
            printf("%d\t", aray[i]);
        putchar('\n');
    }
```

Note

The function parameter must be a pointer to a function of type similar to:

```
int func (const void *, const void *)
```

i.e. it must accept two const void * parameters, and must be prototyped.

RAM_TEST_FAILED

Synopsis

```
void ram_test_failed (unsigned char errcode)
```

Description

The **ram_test_failed()** function is not intended to be called from within the general execution of the program. This routine is called during execution of the generated runtime startup code if the program is using a compiler generated RAM integrity test and the integrity test detects a bad cell.

Upon entry to this function, the working register contains an error code, the address that failed can be determined from the FSR register and IRP bit. The failed value will still be accessable through the INDF register. The default operation of this routine will halt program execution if a bad cell is detected, however the user is free to enhance this functionality if required.

See Also

__ram_cell_test

Note

This routine is intended to be replaced by an equivalent routine to suit the user's implementation. Possible enhancements include logging the location of the dead cell and continuing to test if there are any more more dead cells, or alerting the outside world that the device has a memory problem.

RAND

Synopsis

```
#include <stdlib.h>

int rand (void)
```

Description

The **rand()** function is a pseudo-random number generator. It returns an integer in the range 0 to 32767, which changes in a pseudo-random fashion on each call. The algorithm will produce a deterministic sequence if started from the same point. The starting point is set using the **srand()** call. The example shows use of the **time()** function to generate a different starting point for the sequence each time.

Example

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

void
main (void)
{
    time_t toc;
    int i;

    time(&toc);
    srand((int)toc);
    for(i = 0 ; i != 10 ; i++)
        printf("%d\t", rand());
    putchar('\n');
}
```

See Also

srand()

Note

The example will require the user to provide the time() routine as one cannot be supplied with the compiler. See time() for more detail.

ROUND

Synopsis

```
#include <math.h>

double round (double x)
```

Description

The **round** function round the argument to the nearest integer value, but in floating-point format. Values midway between integer values are rounded up.

Example

```
#include <math.h>

void
main (void)
{
    double input, rounded;
    input = 1234.5678;
    rounded = round(input);
}
```

See Also

trunc()

SCANF, VSCANF

Synopsis

```
#include <stdio.h>

int scanf (const char * fmt, ...)

#include <stdio.h>
#include <stdarg.h>

int vscanf (const char *, va_list ap)
```

Description

The **scanf()** function performs formatted input ("de-editing") from the *stdin stream*. Similar functions are available for streams in general, and for strings. The function **vscanf()** is similar, but takes a pointer to an argument list rather than a series of additional arguments. This pointer should have been initialised with va_start().

The input conversions are performed according to the **fmt** string; in general a character in the format string must match a character in the input; however a space character in the format string will match zero or more "white space" characters in the input, i.e. *spaces, tabs or newlines*.

A conversion specification takes the form of the character **%**, optionally followed by an assignment suppression character ('*'), optionally followed by a numerical maximum field width, followed by a conversion specification character. Each conversion specification, unless it incorporates the assignment suppression character, will assign a value to the variable pointed at by the next argument. Thus if there are two conversion specifications in the **fmt** string, there should be two additional pointer arguments.

The conversion characters are as follows:

o x d

Skip white space, then convert a number in base 8, 16 or 10 radix respectively. If a field width was supplied, take at most that many characters from the input. A leading minus sign will be recognized.

f

Skip white space, then convert a floating number in either conventional or scientific notation. The field width applies as above.

s

Skip white space, then copy a maximal length sequence of non-white-space characters. The pointer

argument must be a pointer to char. The field width will limit the number of characters copied. The resultant string will be null terminated.

c

Copy the next character from the input. The pointer argument is assumed to be a pointer to char. If a field width is specified, then copy that many characters. This differs from the **s** format in that white space does not terminate the character sequence.

The conversion characters **o**, **x**, **u**, **d** and **f** may be preceded by an **l** to indicate that the corresponding pointer argument is a pointer to long or double as appropriate. A preceding **h** will indicate that the pointer argument is a pointer to short rather than int.

Example

```
scanf("%d %s", &a, &c)
     with input " 12s"
     will assign 12 to a, and "s" to s.

scanf("%3cd %lf", &c, &f)
     with input " abcd -3.5"
     will assign " abc" to c, and -3.5 to f.
```

See Also

fscanf(), sscanf(), printf(), va_arg()

Return Value

The **scanf()** function returns the number of successful conversions; EOF is returned if end-of-file was seen before any conversions were performed.

SETJMP

Synopsis

```
#include <setjmp.h>

int setjmp (jmp_buf buf)
```

Description

The **setjmp()** function is used with longjmp() for non-local goto's. See longjmp() for further information.

Example

```
#include <stdio.h>
#include <setjmp.h>
#include <stdlib.h>

jmp_buf jb;

void
inner (void)
{
    longjmp(jb, 5);
}

void
main (void)
{
    int i;

    if(i = setjmp(jb)) {
        printf("setjmp returned %d\n", i);
        exit(0);
    }
    printf("setjmp returned 0 - good\n");
    printf("calling inner...\n");
```

```
        inner();
        printf("inner returned - bad!\n");
    }
```

See Also

longjmp()

Return Value

The **setjmp()** function returns zero after the real call, and non-zero if it apparently returns after a call to longjmp().

SIN

Synopsis

```
#include <math.h>

double sin (double f)
```

Description

This function returns the sine function of its argument.

Example

```
#include <math.h>
#include <stdio.h>

#define C 3.141592/180.0

void
main (void)
{
    double i;

    for(i = 0 ; i <= 180.0 ; i += 10)
        printf("sin(%3.0f) = %f\n", i, sin(i*C));
        printf("cos(%3.0f) = %f\n", i, cos(i*C));
}
```

See Also

cos(), tan(), asin(), acos(), atan(), atan2()

Return Value

Sine vale of **f**.

SPRINTF

Synopsis

```
#include <stdio.h>

/* For baseline and midrange processors */
unsigned char sprintf (char *buf, const char * fmt, ...)

/* For high-end processors */
unsigned char sprintf (far char *buf, const char * fmt, ...)
```

Description

The **sprintf()** function operates in a similar fashion to printf(), except that instead of placing the converted output on the *stdout stream*, the characters are placed in the buffer at **buf**. The resultant string will be null terminated, and the number of characters in the buffer will be returned.

See Also

printf()

Return Value

The **sprintf()** routine returns the number of characters placed into the buffer.
NB: The return value is a char not an int.

Note

For High-end processors the buffer is accessed via a far pointer.

SQRT

Synopsis

```
#include <math.h>

double sqrt (double f)
```

Description

The function **sqrt()**, implements a square root routine using Newton's approximation.

Example

```
#include <math.h>
#include <stdio.h>

void
main (void)
{
    double i;

    for(i = 0 ; i <= 20.0 ; i += 1.0)
        printf("square root of %.1f = %f\n", i, sqrt(i));
}
```

See Also

exp()

Return Value

Returns the value of the square root.

Note

A domain error occurs if the argument is negative.

SRAND

Synopsis

```
#include <stdlib.h>

void srand (unsigned int seed)
```

Description

The **srand()** function initializes the random number generator accessed by rand() with the given **seed**. This provides a mechanism for varying the starting point of the pseudo-random sequence yielded by rand(). On the Z80, a good place to get a truly random seed is from the refresh register. Otherwise timing a response from the console will do, or just using the system time.

Example

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

void
main (void)
{
    time_t toc;
    int i;

    time(&toc);
    srand((int)toc);
    for(i = 0 ; i != 10 ; i++)
        printf("%d\t", rand());
    putchar('\n');
}
```

See Also

rand()

STRCAT

Synopsis

```
#include <string.h>

char * strcat (char * s1, const char * s2)
```

Description

This function appends (concatenates) string **s2** to the end of string **s1**. The result will be null terminated. The argument **s1** must point to a character array big enough to hold the resultant string.

Example

```
#include <string.h>
#include <stdio.h>

void
main (void)
{
    char buffer[256];
    char * s1, * s2;

    strcpy(buffer, "Start of line");
    s1 = buffer;
    s2 = " ... end of line";
    strcat(s1, s2);
    printf("Length = %d\n", strlen(buffer));
    printf("string = \"%s\"\n", buffer);
}
```

See Also

strcpy(), strcmp(), strncat(), strlen()

Return Value

The value of **s1** is returned.

STRCAT

Synopsis

```
#include <string.h>

/* For baseline and midrange processors */
char * strcat (char * s1, const char * s2)

/* For high-end processors */
far char * strcat (far char * s1, const char * s2)
```

Description

This function appends (contcatenates) string **s2** to the end of string **s1**. The result will be null terminated. The argument **s1** must point to a character array big enough to hold the resultant string.

Example

```
#include <string.h>
#include <stdio.h>

void
main (void)
{
    char buffer[256];
    char * s1, * s2;

    strcpy(buffer, "Start of line");
    s1 = buffer;
    s2 = " ... end of line";
    strcat(s1, s2);
    printf("Length = %d\n", strlen(buffer));
    printf("string = \"%s\"\n", buffer);
}
```

See Also

strcpy(), strcmp(), strncat(), strlen()

Return Value

The value of **s1** is returned.

STRCHR, STRICHR

Synopsis

```
#include <string.h>

char * strchr (const char * s, int c)
char * strichr (const char * s, int c)
```

Description

The **strchr()** function searches the string **s** for an occurrence of the character **c**. If one is found, a pointer to that character is returned, otherwise NULL is returned.

The **strichr()** function is the case-insensitive version of this function.

Example

```
#include <strings.h>
#include <stdio.h>

void
main (void)
{
    static char temp[] = "Here it is...";
    char c = 's';

    if(strchr(temp, c))
        printf("Character %c was found in string\n", c);
    else
        printf("No character was found in string");
}
```

See Also

strrchr(), strlen(), strcmp()

Return Value

A pointer to the first match found, or NULL if the character does not exist in the string.

Note

Although the function takes an integer argument for the character, only the lower 8 bits of the value are used.

STRCHR, STRICHR

Synopsis

```
#include <string.h>

/* For baseline and midrange processors */
const char * strchr (const char * s, int c)
const char * strichr (const char * s, int c)

/* For high-end processors */
char * strchr (const char * s, int c)
char * strichr (const char * s, int c)
```

Description

The **strchr()** function searches the string **s** for an occurrence of the character **c**. If one is found, a pointer to that character is returned, otherwise NULL is returned.

The **strichr()** function is the case-insensitive version of this function.

Example

```
#include <strings.h>
#include <stdio.h>

void
main (void)
{
    static char temp[] = "Here it is...";
    char c = 's';

    if(strchr(temp, c))
        printf("Character %c was found in string\n", c);
    else
        printf("No character was found in string");
}
```

See Also

strrchr(), strlen(), strcmp()

Return Value

A pointer to the first match found, or NULL if the character does not exist in the string.

Note

The functions takes an integer argument for the character, only the lower 8 bits of the value are used.

STRCMP, STRICMP

Synopsis

```
#include <string.h>

int strcmp (const char * s1, const char * s2)
int stricmp (const char * s1, const char * s2)
```

Description

The **strcmp()** function compares its two, null terminated, string arguments and returns a signed integer to indicate whether **s1** is less than, equal to or greater than **s2**. The comparison is done with the standard collating sequence, which is that of the ASCII character set.

The **stricmp()** function is the case-insensitive version of this function.

Example

```
#include <string.h>
#include <stdio.h>

void
main (void)
{
    int i;

    if((i = strcmp("ABC", "ABc")) < 0)
        printf("ABC is less than ABc\n");
    else if(i > 0)
        printf("ABC is greater than ABc\n");
    else
        printf("ABC is equal to ABc\n");
}
```

See Also

strlen(), strncmp(), strcpy(), strcat()

Return Value

A signed integer less than, equal to or greater than zero.

Note

Other C implementations may use a different collating sequence; the return value is negative, zero or positive, i.e. do not test explicitly for negative one (-1) or one (1).

STRCPY

Synopsis

```
#include <string.h>

char * strcpy (char * s1, const char * s2)
```

Description

This function copies a null terminated string **s2** to a character array pointed to by **s1**. The destination array must be large enough to hold the entire string, including the null terminator.

Example

```
#include <string.h>
#include <stdio.h>

void
main (void)
{
    char buffer[256];
    char * s1, * s2;

    strcpy(buffer, "Start of line");
    s1 = buffer;
    s2 = " ... end of line";
    strcat(s1, s2);
    printf("Length = %d\n", strlen(buffer));
    printf("string = \"%s\"\n", buffer);
}
```

See Also

strncpy(), strlen(), strcat(), strlen()

Return Value

The destination buffer pointer **s1** is returned.

STRCPY

Synopsis

```
#include <string.h>

/* For baseline and midrange processors */
char * strcpy (char * s1, const char * s2)

/* For high-end processors */
far char * strcpy (far char * s1, const char * s2)
```

Description

This function copies a null terminated string **s2** to a character array pointed to by **s1**. The destination array must be large enough to hold the entire string, including the null terminator.

Example

```
#include <string.h>
#include <stdio.h>

void
main (void)
{
    char buffer[256];
    char * s1, * s2;

    strcpy(buffer, "Start of line");
    s1 = buffer;
    s2 = " ... end of line";
    strcat(s1, s2);
    printf("Length = %d\n", strlen(buffer));
    printf("string = \"%s\"\n", buffer);
}
```

See Also

strncpy(), strlen(), strcat(), strlen()

Return Value

The destination buffer pointer **s1** is returned.

STRCSPN

Synopsis

```
#include <string.h>

size_t strcspn (const char * s1, const char * s2)
```

Description

The **strcspn()** function returns the length of the initial segment of the string pointed to by **s1** which consists of characters NOT from the string pointed to by **s2**.

Example

```
#include <stdio.h>
#include <string.h>

void
main (void)
{
    static char set[] = "xyz";

    printf("%d\n", strcspn( "abcdevwxyz", set));
    printf("%d\n", strcspn( "xxxbcadefs", set));
    printf("%d\n", strcspn( "1234567890", set));
}
```

See Also

strspn()

Return Value

Returns the length of the segment.

STRLEN

Synopsis

```
#include <string.h>

size_t strlen (const char * s)
```

Description

The **strlen()** function returns the number of characters in the string **s**, not including the null terminator.

Example

```
#include <string.h>
#include <stdio.h>

void
main (void)
{
    char buffer[256];
    char * s1, * s2;

    strcpy(buffer, "Start of line");
    s1 = buffer;
    s2 = " ... end of line";
    strcat(s1, s2);
    printf("Length = %d\n", strlen(buffer));
    printf("string = \"%s\"\n", buffer);
}
```

Return Value

The number of characters preceding the null terminator.

STRNCAT

Synopsis

```
#include <string.h>

char * strncat (char * s1, const char * s2, size_t n)
```

Description

This function appends (concatenates) string **s2** to the end of string **s1**. At most **n** characters will be copied, and the result will be null terminated. **s1** must point to a character array big enough to hold the resultant string.

Example

```
#include <string.h>
#include <stdio.h>

void
main (void)
{
    char buffer[256];
    char * s1, * s2;

    strcpy(buffer, "Start of line");
    s1 = buffer;
    s2 = " ... end of line";
    strncat(s1, s2, 5);
    printf("Length = %d\n", strlen(buffer));
    printf("string = \"%s\"\n", buffer);
}
```

See Also

strcpy(), strcmp(), strcat(), strlen()

Return Value

The value of **s1** is returned.

STRNCAT

Synopsis

```
#include <string.h>

/* For baseline and midrange processors */
char * strncat (char * s1, const char * s2, size_t n)

/* For high-end processors */
far char * strncat (far char * s1, const char * s2, size_t n)
```

Description

This function appends (concatenates) string **s2** to the end of string **s1**. At most **n** characters will be copied, and the result will be null terminated. **s1** must point to a character array big enough to hold the resultant string.

Example

```
#include <string.h>
#include <stdio.h>

void
main (void)
{
    char buffer[256];
    char * s1, * s2;

    strcpy(buffer, "Start of line");
    s1 = buffer;
    s2 = " ... end of line";
    strncat(s1, s2, 5);
    printf("Length = %d\n", strlen(buffer));
    printf("string = \"%s\"\n", buffer);
}
```

See Also

strcpy(), strcmp(), strcat(), strlen()

Return Value

The value of **s1** is returned.

STRNCMP, STRNICMP

Synopsis

```
#include <string.h>

int strncmp (const char * s1, const char * s2, size_t n)
int strnicmp (const char * s1, const char * s2, size_t n)
```

Description

The **strncmp()** function compares its two, null terminated, string arguments, up to a maximum of **n** characters, and returns a signed integer to indicate whether **s1** is less than, equal to or greater than **s2**. The comparison is done with the standard collating sequence, which is that of the ASCII character set.

The **strnicmp()** function is the case-insensitive version of this function.

Example

```
#include <stdio.h>
#include <string.h>

void
main (void)
{
    int i;

    i = strncmp("abcxyz", "abcxyz",6);
    if(i == 0)
        printf("Both strings are equal\n");
    else if(i > 0)
        printf("String 2 less than string 1\n");
    else
        printf("String 2 is greater than string 1\n");
}
```

See Also

strlen(), strcmp(), strcpy(), strcat()

Return Value

A signed integer less than, equal to or greater than zero.

Note

Other C implementations may use a different collating sequence; the return value is negative, zero or positive, i.e. do not test explicitly for negative one (-1) or one (1).

STRNCPY

Synopsis

```
#include <string.h>

char * strncpy (char * s1, const char * s2, size_t n)
```

Description

This function copies a null terminated string **s2** to a character array pointed to by **s1**. At most **n** characters are copied. If string **s2** is longer than **n** then the destination string will not be null terminated. The destination array must be large enough to hold the entire string, including the null terminator.

Example

```
#include <string.h>
#include <stdio.h>

void
main (void)
{
    char buffer[256];
    char * s1, * s2;

    strncpy(buffer, "Start of line", 6);
    s1 = buffer;
    s2 = " ... end of line";
    strcat(s1, s2);
    printf("Length = %d\n", strlen(buffer));
    printf("string = \"%s\"\n", buffer);
}
```

See Also

strcpy(), strcat(), strlen(), strcmp()

Return Value

The destination buffer pointer **s1** is returned.

STRNCPY

Synopsis

```
#include <string.h>

/* For baseline and midrange processors */
char * strncpy (char * s1, const char * s2, size_t n)

/* For high-end processors */
far char * strncpy (far char * s1, const char * s2, size_t n)
```

Description

This function copies a null terminated string **s2** to a character array pointed to by **s1**. At most **n** characters are copied. If string **s2** is longer than **n** then the destination string will not be null terminated. The destination array must be large enough to hold the entire string, including the null terminator.

Example

```
#include <string.h>
#include <stdio.h>

void
main (void)
{
    char buffer[256];
    char * s1, * s2;

    strncpy(buffer, "Start of line", 6);
    s1 = buffer;
    s2 = " ... end of line";
    strcat(s1, s2);
    printf("Length = %d\n", strlen(buffer));
    printf("string = \"%s\"\n", buffer);
}
```

See Also

strcpy(), strcat(), strlen(), strcmp()

Return Value

The destination buffer pointer **s1** is returned.

STRPBRK

Synopsis

```
#include <string.h>

char * strpbrk (const char * s1, const char * s2)
```

Description

The **strpbrk()** function returns a pointer to the first occurrence in string **s1** of any character from string **s2**, or a null pointer if no character from **s2** exists in **s1**.

Example

```
#include <stdio.h>
#include <string.h>

void
main (void)
{
    char * str = "This is a string.";

    while(str != NULL) {
        printf( "%s\n", str );
        str = strpbrk( str+1, "aeiou" );
    }
}
```

Return Value

Pointer to the first matching character, or NULL if no character found.

STRPBRK

Synopsis

```
#include <string.h>

/* For baseline and midrange processors */
const char * strpbrk (const char * s1, const char * s2)

/* For high-end processors */
char * strpbrk (const char * s1, const char * s2)
```

Description

The **strpbrk()** function returns a pointer to the first occurrence in string **s1** of any character from string **s2**, or a null pointer if no character from **s2** exists in **s1**.

Example

```
#include <stdio.h>
#include <string.h>

void
main (void)
{
    char * str = "This is a string.";

    while(str != NULL) {
        printf( "%s\n", str );
        str = strpbrk( str+1, "aeiou" );
    }
}
```

Return Value

Pointer to the first matching character, or NULL if no character found.

STRRCHR, STRRICHR

Synopsis

```
#include <string.h>

char * strrchr (char * s, int c)
char * strrichr (char * s, int c)
```

Description

The **strrchr()** function is similar to the **strchr()** function, but searches from the end of the string rather than the beginning, i.e. it locates the *last* occurrence of the character **c** in the null terminated string **s**. If successful it returns a pointer to that occurrence, otherwise it returns NULL.

The **strrichr()** function is the case-insensitive version of this function.

Example

```
#include <stdio.h>
#include <string.h>

void
main (void)
{
    char * str = "This is a string.";

    while(str != NULL) {
        printf( "%s\n", str );
        str = strrchr( str+1, 's');
    }
}
```

See Also

strchr(), strlen(), strcmp(), strcpy(), strcat()

Return Value

A pointer to the character, or NULL if none is found.

STRRCHR, STRRICHR

Synopsis

```
#include <string.h>

/* For baseline and midrange processors */
const char * strrchr (char * s, int c)
const char * strrichr (char * s, int c)

/* For high-end processors */
char * strrchr (char * s, int c)
char * strrichr (char * s, int c)
```

Description

The **strrchr()** function is similar to the **strchr()** function, but searches from the end of the string rather than the beginning, i.e. it locates the *last* occurrence of the character **c** in the null terminated string **s**. If successful it returns a pointer to that occurrence, otherwise it returns NULL.

The **strrichr()** function is the case-insensitive version of this function.

Example

```
#include <stdio.h>
#include <string.h>

void
main (void)
{
    char * str = "This is a string.";

    while(str != NULL) {
        printf( "%s\n", str );
        str = strrchr( str+1, 's');
    }
}
```

See Also

strchr(), strlen(), strcmp(), strcpy(), strcat()

Return Value

A pointer to the character, or NULL if none is found.

STRSPN

Synopsis

```
#include <string.h>

size_t strspn (const char * s1, const char * s2)
```

Description

The **strspn()** function returns the length of the initial segment of the string pointed to by **s1** which consists entirely of characters from the string pointed to by **s2**.

Example

```
#include <stdio.h>
#include <string.h>

void
main (void)
{
    printf("%d\n", strspn("This is a string", "This"));
    printf("%d\n", strspn("This is a string", "this"));
}
```

See Also

strcspn()

Return Value

The length of the segment.

STRSTR, STRISTR

Synopsis

```
#include <string.h>

char * strstr (const char * s1, const char * s2)
char * stristr (const char * s1, const char * s2)
```

Description

The **strstr()** function locates the first occurrence of the sequence of characters in the string pointed to by **s2** in the string pointed to by **s1**.

The **stristr()** routine is the case-insensitive version of this function.

Example

```
#include <stdio.h>
#include <string.h>

void
main (void)
{
    printf("%d\n", strstr("This is a string", "str"));
}
```

Return Value

Pointer to the located string or a null pointer if the string was not found.

STRSTR, STRISTR

Synopsis

```
#include <string.h>

/* For baseline and midrange processors */
const char * strstr (const char * s1, const char * s2)
const char * stristr (const char * s1, const char * s2)

/* For high-end processors */
char * strstr (const char * s1, const char * s2)
char * stristr (const char * s1, const char * s2)
```

Description

The **strstr()** function locates the first occurrence of the sequence of characters in the string pointed to by **s2** in the string pointed to by **s1**.

The **stristr()** routine is the case-insensitive version of this function.

Example

```
#include <stdio.h>
#include <string.h>

void
main (void)
{
    printf("%d\n", strstr("This is a string", "str"));
}
```

Return Value

Pointer to the located string or a null pointer if the string was not found.

STRTOD

Synopsis

```
#include <stdlib.h>

double strtok (const char * s, const char ** res)
```

Description

Parse the string `s` converting it to a double floating point type. This function converts the first occurence of a substring of the input that is made up of characters of the expected form after skipping leading white-space characters. If `res` is not `NULL`, it will be made to point to the first character after the converted sub-string.

Example

```
#include <stdio.h>
#include <strlib.h>

void
main (void)
{
    char buf[] = " 35.7  23.27 ";
    char * end;
    double in1, in2;

    in1 = strtod(buf, &end);
    in2 = strtod(end, NULL);
    printf("in comps: %f, %f\n", in1, in2);
}
```

See Also

atof()

Return Value

Returns a double representing the floating-point value of the converted input string.

STRTOL

Synopsis

```
#include <stdlib.h>

double strtol (const char * s, const char ** res, int base)
```

Description

Parse the string `s` converting it to a long integer type. This function converts the first occurence of a substring of the input that is made up of characters of the expected form after skipping leading white-space characters. The radix of the input is determined from **base**. If this is zero, then the radix defaults to base 10. If **res** is not `NULL`, it will be made to point to the first character after the converted sub-string.

Example

```
#include <stdio.h>
#include <strlib.h>

void
main (void)
{
    char buf[] = " 0X299 0x792 ";
    char * end;
    long in1, in2;

    in1 = strtol(buf, &end, 16);
    in2 = strtol(end, NULL, 16);
    printf("in (decimal): %ld, %ld\n", in1, in2);
}
```

See Also

strtod()

Return Value

Returns a long int representing the value of the converted input string using the specified base.

STRTOK

Synopsis

```
#include <string.h>

char * strtok (char * s1, const char * s2)
```

Description

A number of calls to **strtok()** breaks the string **s1** (which consists of a sequence of zero or more text tokens separated by one or more characters from the separator string **s2**) into its separate tokens.

The first call must have the string **s1**. This call returns a pointer to the first character of the first token, or NULL if no tokens were found. The inter-token separator character is overwritten by a null character, which terminates the current token.

For subsequent calls to **strtok()**, **s1** should be set to a null pointer. These calls start searching from the end of the last token found, and again return a pointer to the first character of the next token, or NULL if no further tokens were found.

Example

```
#include <stdio.h>
#include <string.h>

void
main (void)
{
    char * ptr;
    char buf[] = "This is a string of words.";
    char * sep_tok = ".,?! ";

    ptr = strtok(buf, sep_tok);
    while(ptr != NULL) {
        printf("%s\n", ptr);
        ptr = strtok(NULL, sep_tok);
    }
}
```

Return Value

Returns a pointer to the first character of a token, or a null pointer if no token was found.

Note

The separator string **s2** may be different from call to call.

STRTOK

Synopsis

```
#include <string.h>

/* For baseline and midrange processors */
char * strtok (char * s1, const char * s2)

/* For high-end processors */
far char * strtok (far char * s1, const char * s2)
```

Description

A number of calls to **strtok()** breaks the string **s1** (which consists of a sequence of zero or more text tokens separated by one or more characters from the separator string **s2**) into its separate tokens.

The first call must have the string **s1**. This call returns a pointer to the first character of the first token, or NULL if no tokens were found. The inter-token separator character is overwritten by a null character, which terminates the current token.

For subsequent calls to **strtok()**, **s1** should be set to a null pointer. These calls start searching from the end of the last token found, and again return a pointer to the first character of the next token, or NULL if no further tokens were found.

Example

```
#include <stdio.h>
#include <string.h>

void
main (void)
{
    char * ptr;
    char * buf = "This is a string of words.";
    char * sep_tok = ".,?! ";

    ptr = strtok(buf, sep_tok);
    while(ptr != NULL) {
        printf("%s\n", ptr);
```

```
            ptr = strtok(NULL, sep_tok);
        }
    }
```

Return Value

Returns a pointer to the first character of a token, or a null pointer if no token was found.

Note

The separator string **s2** may be different from call to call.

TAN

Synopsis

```
#include <math.h>

double tan (double f)
```

Description

The **tan()** function calculates the tangent of **f**.

Example

```
#include <math.h>
#include <stdio.h>

#define C 3.141592/180.0

void
main (void)
{
    double i;

    for(i = 0 ; i <= 180.0 ; i += 10)
        printf("tan(%3.0f) = %f\n", i, tan(i*C));
}
```

See Also

sin(), cos(), asin(), acos(), atan(), atan2()

Return Value

The tangent of **f**.

TIME

Synopsis

```
#include <time.h>

time_t time (time_t * t)
```

Description

This function is not provided as it is dependant on the target system supplying the current time. This function will be user implemented. When implemented, this function should return the current time in seconds since 00:00:00 on Jan 1, 1970. If the argument **t** is not equal to NULL, the same value is stored into the object pointed to by **t**.

Example

```
#include <stdio.h>
#include <time.h>

void
main (void)
{
    time_t clock;

    time(&clock);
    printf("%s", ctime(&clock));
}
```

See Also

ctime(), gmtime(), localtime(), asctime()

Return Value

This routine when implemented will return the current time in seconds since 00:00:00 on Jan 1, 1970.

Note

The **time()** routine is not supplied, if required the user will have to implement this routine to the specifications outlined above.

TOLOWER, TOUPPER, TOASCII

Synopsis

```
#include <ctype.h>

char toupper (int c)
char tolower (int c)
char toascii (int c)
```

Description

The **toupper()** function converts its lower case alphabetic argument to upper case, the **tolower()** routine performs the reverse conversion and the **toascii()** macro returns a result that is guaranteed in the range 0-0177. The functions **toupper()** and **tolower()** return their arguments if it is not an alphabetic character.

Example

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>

void
main (void)
{
    char * array1 = "aBcDE";
    int i;

    for(i=0;i < strlen(array1); ++i) {
        printf("%c", tolower(array1[i]));
    }
    printf("\n");
}
```

See Also

islower(), isupper(), isascii(), et. al.

TRUNC

Synopsis

```
#include <math.h>

double trunc (double x)
```

Description

The **trunc** function rounds the argument to the nearest integer value, in floating-point format, that is not larger in magniture than the argument.

Example

```
#include <math.h>

void
main (void)
{
    double input, rounded;
    input = 1234.5678;
    rounded = trunc(input);
}
```

See Also

round()

UDIV

Synopsis

```
#include <stdlib.h>

int udiv (unsigned num, unsigned demon)
```

Description

The **udiv()** function calculate the quotient and remainder of the division of `number` and `denom`, storing the results into a udiv_t structure which is returned.

Example

```
#include <stdlib.h>

void
main (void)
{
    udiv_t result;
    unsigned  num = 1234, den = 7;

    result = udiv(num, den);
}
```

See Also

uldiv(), div(), ldiv()

Return Value

Returns the the quotient and remainder as a udiv_t structure.

ULDIV

Synopsis

```
#include <stdlib.h>

int uldiv (unsigned long num, unsigned long demon)
```

Description

The **uldiv()** function calculate the quotient and remainder of the division of `number` and `denom`, storing the results into a uldiv_t structure which is returned.

Example

```
#include <stdlib.h>

void
main (void)
{
    uldiv_t result;
    unsigned long num = 1234, den = 7;

    result = uldiv(num, den);
}
```

See Also

ldiv(), udiv(), div()

Return Value

Returns the the quotient and remainder as a uldiv_t structure.

UNGETCH

Synopsis

```
#include <conio.h>

void ungetch (char c)
```

Description

The **ungetch()** function will push back the character **c** onto the console stream, such that a subsequent getch() operation will return the character. At most one level of push back will be allowed.

See Also

getch(), getche()

UTOA

Synopsis

```
#include <stdlib.h>

char * utoa (char * buf, unsigned val, int base)
```

Description

The function **itoa** converts the unsigned contents of **val** into a string which is stored into **buf**. The conversion is performed according to the radix specified in **base**. **buf** is assumed to reference a buffer which has sufficent space allocated to it.

Example

```
#include <stdlib.h>
#include <stdio.h>

void
main (void)
{
    char buf[10];
    utoi(buf, 1234, 16);
    printf("The buffer holds %s\n", buf);
}
```

See Also

strtol(), itoa(), ltoa(), ultoa()

Return Value

This routine returns a copy of the buffer into which the result is written.

VA_START, VA_ARG, VA_END

Synopsis

```
#include <stdarg.h>

void va_start (va_list ap, parmN)
type va_arg  (ap, type)
void va_end  (va_list ap)
```

Description

These macros are provided to give access in a portable way to parameters to a function represented in a prototype by the ellipsis symbol (...), where type and number of arguments supplied to the function are not known at compile time.

The rightmost parameter to the function (shown as **parmN**) plays an important role in these macros, as it is the starting point for access to further parameters. In a function taking variable numbers of arguments, a variable of type **va_list** should be declared, then the macro **va_start()** invoked with that variable and the name of **parmN**. This will initialize the variable to allow subsequent calls of the macro **va_arg()** to access successive parameters.

Each call to **va_arg()** requires two arguments; the variable previously defined and a type name which is the type that the next parameter is expected to be. Note that any arguments thus accessed will have been widened by the default conventions to *int, unsigned int* or *double*. For example if a character argument has been passed, it should be accessed by **va_arg(ap, int)** since the *char* will have been widened to *int*.

An example is given below of a function taking one integer parameter, followed by a number of other parameters. In this example the function expects the subsequent parameters to be pointers to char, but note that the compiler is not aware of this, and it is the programmers responsibility to ensure that correct arguments are supplied.

Example

```
#include <stdio.h>
#include <stdarg.h>

void
pf (int a, ...)
{
```

```
    va_list ap;

    va_start(ap, a);
    while(a--)
        puts(va_arg(ap, char *));
    va_end(ap);
}

void
main (void)
{
    pf(3, "Line 1", "line 2", "line 3");
}
```

XTOI

Synopsis

```
#include <stdlib.h>

unsigned xtoi (const char * s)
```

Description

The **xtoi()** function scans the character string passed to it, skipping leading blanks reading an optional sign, and converts an ASCII representation of a hexadecimal number to an integer.

Example

```
#include <stdlib.h>
#include <stdio.h>

void
main (void)
{
    char buf[80];
    int i;

    gets(buf);
    i = xtoi(buf);
    printf("Read %s: converted to %x\n", buf, i);
}
```

See Also

atoi()

Return Value

A signed integer. If no number is found in the string, zero will be returned.

Appendix B

Error and Warning Messages

This chapter lists most error, warning and advisory messages from all HI-TECH C compilers, with an explanation of each message. Most messages have been assigned a unique number which appears in brackets before each message in this chapter, and which is also printed by the compiler when the message is issued. The messages shown here are sorted by their number. Un-numbered messages appear toward the end and are sorted alphabetically.

The name of the application(s) that could have produced the messages are listed in brackets opposite the error message. In some cases examples of code or options that could trigger the error are given. The use of * in the error message is used to represent a string that the compiler will substitute that is specific to that particular error.

Note that one problem in your C or assembler source code may trigger more than one error message.

(1) too many errors (*) ***(all applications)***

The executing compiler application has encountered too many errors and will exit immediately. Other uncompiled source files will be processed, but the compiler applications that would normally be executed in due course will not be run. The number of errors that can be accepted can be controlled using the `--ERRORS` option, See Section 2.6.29.

(2) error/warning (*) generated, but no description available ***(all applications)***

The executing compiler application has emitted a message (advisory/warning/error), but there is no description available in the message description file (MDF) to print. This may be because the MDF is out of date, or the message issue has not been tranlated into the selected language.

(3) malformed error information on line *, in file * ***(all applications)***

The compiler has attempted to load the messages for the selected language, but the message description file (MDF) was corrupted and could not be read correctly.

(100) unterminated #if[n][def] block from line * ***(Preprocessor)***

A `#if` or similar block was not terminated with a matching `#endif`, e.g.:

```
#if INPUT         /* error flagged here */
void main(void)
{
  run();
}                 /* no #endif was found in this module */
```

(101) #* may not follow #else ***(Preprocessor)***

A `#else` or `#elif` has been used in the same conditional block as a `#else`. These can only follow a `#if`, e.g.:

```
#ifdef FOO
  result = foo;
#else
  result = bar;
#elif defined(NEXT)    /* the #else above terminated the #if */
  result = next(0);
#endif
```

(102) #* must be in an #if ***(Preprocessor)***

The `#elif`, `#else` or `#endif` directive must be preceded by a matching `#if` line. If there is an apparently corresponding `#if` line, check for things like extra `#endif`'s, or improperly terminated comments, e.g.:

```
#ifdef FOO
  result = foo;
#endif
  result = bar;
#elif defined(NEXT)    /* the #endif above terminated the #if */
  result = next(0);
#endif
```

(103) #error: * ***(Preprocessor)***

This is a programmer generated error; there is a directive causing a deliberate error. This is normally used to check compile time defines etc. Remove the directive to remove the error, but first check as to why the directive is there.

(104) preprocessor #assert failure ***(Preprocessor)***

The argument to a preprocessor `#assert` directive has evaluated to zero. This is a programmer induced error.

```
#assert SIZE == 4  /* size should never be 4 */
```

(105) no #asm before #endasm ***(Preprocessor)***

A `#endasm` operator has been encountered, but there was no previous matching `#asm`, e.g.:

```
void cleardog(void)
{
  clrwdt
#endasm  /* in-line assembler ends here,
            only where did it begin? */
}
```

(106) nested #asm directives ***(Preprocessor)***

It is not legal to nest `#asm` directives. Check for a missing or misspelt `#endasm` directive, e.g.:

```
#asm
  move r0, #0aah
#asm      ; previous #asm must be closed before opening another
  sleep
#endasm
```

(107) illegal # directive "*" ***(Preprocessor, Parser)***

The compiler does not understand the # directive. It is probably a misspelling of a pre-processor # directive, e.g.:

```
#indef DEBUG  /* oops -- that should be #undef DEBUG */
```

(108) #if[n][def] without an argument ***(Preprocessor)***

The preprocessor directives `#if`, `#ifdef` and `#ifndef` must have an argument. The argument to `#if` should be an expression, while the argument to `#ifdef` or `#ifndef` should be a single name, e.g.:

```
#if             /* oops -- no argument to check */
  output = 10;
#else
  output = 20;
#endif
```

(109) #include syntax error ***(Preprocessor)***

The syntax of the filename argument to `#include` is invalid. The argument to `#include` must be a valid file name, either enclosed in double quotes `""` or angle brackets `< >`. Spaces should not be included, and the closing quote or bracket must be present. There should be nothing else on the line other than comments, e.g.:

```
#include stdio.h  /* oops -- should be: #include <stdio.h> */
```

(110) too many file arguments; usage: cpp [input [output]] ***(Preprocessor)***

`CPP` should be invoked with at most two file arguments. Contact HI-TECH Support if the preprocessor is being executed by a compiler driver.

(111) redefining preprocessor macro "*" ***(Preprocessor)***

The macro specified is being redefined, to something different to the original definition. If you want to deliberately redefine a macro, use #`undef` first to remove the original definition, e.g.:

```
#define ONE 1
/* elsewhere: */
/* Is this correct? It will overwrite the first definition. */
#define ONE one
```

(112) #define syntax error ***(Preprocessor)***

A macro definition has a syntax error. This could be due to a macro or formal parameter name that does not start with a letter or a missing *closing parenthesis* `)`, e.g.:

```
#define FOO(a, 2b)  bar(a, 2b)  /* 2b is not to be! */
```

(113) unterminated string in preprocessor macro body ***(Preprocessor, Assembler)***

A macro definition contains a string that lacks a closing quote.

(114) illegal #undef argument ***(Preprocessor)***

The argument to `#undef` must be a valid name. It must start with a letter, e.g.:

```
#undef 6YYY  /* this isn't a valid symbol name */
```

(115) recursive preprocessor macro definition of "*" defined by "*" ***(Preprocessor)***

The named macro has been defined in such a manner that expanding it causes a recursive expansion of itself!

(116) end of file within preprocessor macro argument from line * ***(Preprocessor)***

A macro argument has not been terminated. This probably means the closing parenthesis has been omitted from a macro invocation. The line number given is the line where the macro argument started, e.g.:

```
#define FUNC(a, b) func(a+b)
FUNC(5, 6;    /* oops -- where is the closing bracket? */
```

(117) misplaced constant in #if ***(Preprocessor)***

A constant in a `#if` expression should only occur in syntactically correct places. This error is most probably caused by omission of an operator, e.g.:

```
#if FOO BAR  /* oops -- did you mean: #if FOO == BAR ? */
```

(118) stack overflow processing #if expression ***(Preprocessor)***

The preprocessor filled up its expression evaluation stack in a `#if` expression. Simplify the expression — it probably contains too many parenthesized subexpressions.

(119) invalid expression in #if line ***(Preprocessor)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(120) operator "*" in incorrect context ***(Preprocessor)***

An operator has been encountered in a `#if` expression that is incorrectly placed, e.g. two binary operators are not separated by a value, e.g.:

```
#if FOO * % BAR == 4  /* what is "* %" ? */
  #define BIG
#endif
```

(121) expression stack overflow at operator "*" ***(Preprocessor)***

Expressions in `#if` lines are evaluated using a stack with a size of 128. It is possible for very complex expressions to overflow this. Simplify the expression.

(122) unbalanced parenthesis at operator "*" ***(Preprocessor)***

The evaluation of a `#if` expression found mismatched parentheses. Check the expression for correct parenthesisation, e.g.:

```
#if ((A) + (B)  /* oops -- a missing ), I think */
  #define ADDED
#endif
```

(123) misplaced "?" or ":"; previous operator is "*" ***(Preprocessor)***

A colon operator has been encountered in a `#if` expression that does not match up with a corresponding `?` operator, e.g.:

```
#if XXX : YYY  /* did you mean:  #if COND ? XXX : YYY */
```

(124) illegal character "*" in #if ***(Preprocessor)***

There is a character in a `#if` expression that has no business being there. Valid characters are the letters, digits and those comprising the acceptable operators, e.g.:

```
#if `YYY`  /* what are these characters doing here? */
  int m;
#endif
```

(125) illegal character (* decimal) in #if ***(Preprocessor)***

There is a non-printable character in a `#if` expression that has no business being there. Valid characters are the letters, digits and those comprising the acceptable operators, e.g.:

```
#if ^SYYY  /* what is this control characters doing here? */
  int m;
#endif
```

(126) strings can't be used in #if ***(Preprocessor)***

The preprocessor does not allow the use of strings in `#if` expressions, e.g.:

```
/* no string operations allowed by the preprocessor */
#if MESSAGE > "hello"
#define DEBUG
#endif
```

(127) bad syntax for defined() in #[el]if ***(Preprocessor)***

The `defined()` pseudo-function in a preprocessor expression requires its argument to be a single name. The name must start with a letter and should be enclosed in parentheses, e.g.:

```
/* oops -- defined expects a name, not an expression */
#if defined(a&b)
  input = read();
#endif
```

(128) illegal operator in #if ***(Preprocessor)***

A `#if` expression has an illegal operator. Check for correct syntax, e.g.:

```
#if FOO = 6  /* oops -- should that be: #if FOO == 5 ? */
```

(129) unexpected "\" in #if ***(Preprocessor)***

The *backslash* is incorrect in the `#if` statement, e.g.:

```
#if FOO == \34
  #define BIG
#endif
```

(130) unknown type "*" in #[el]if sizeof() ***(Preprocessor)***

An unknown type was used in a preprocessor `sizeof()`. The preprocessor can only evaluate `sizeof()` with basic types, or pointers to basic types, e.g.:

```
#if sizeof(unt) == 2  /* should be: #if sizeof(int) == 2 */
  i = 0xFFFF;
#endif
```

(131) illegal type combination in #[el]if sizeof() ***(Preprocessor)***

The preprocessor found an illegal type combination in the argument to `sizeof()` in a `#if` expression, e.g.

```
/* To sign, or not to sign, that is the error. */
#if sizeof(signed unsigned int) == 2
  i = 0xFFFF;
#endif
```

(132) no type specified in #[el]if sizeof() ***(Preprocessor)***

Sizeof() was used in a preprocessor `#if` expression, but no type was specified. The argument to `sizeof()` in a preprocessor expression must be a valid simple type, or pointer to a simple type, e.g.:

```
#if sizeof()  /* oops -- size of what? */
  i = 0;
#endif
```

(133) unknown type code (0x*) in #[el]if sizeof() ***(Preprocessor)***

The preprocessor has made an internal error in evaluating a `sizeof()` expression. Check for a malformed type specifier. This is an internal error. Contact HI-TECH Software technical support with details.

(134) syntax error in #[el]if sizeof() ***(Preprocessor)***

The preprocessor found a syntax error in the argument to `sizeof`, in a `#if` expression. Probable causes are mismatched parentheses and similar things, e.g.:

```
#if sizeof(int == 2)  // oops - should be: #if sizeof(int) == 2
  i = 0xFFFF;
#endif
```

(135) unknown operator (*) in #if ***(Preprocessor)***

The preprocessor has tried to evaluate an expression with an operator it does not understand. This is an internal error. Contact HI-TECH Software technical support with details.

(137) strange character "*" after ## ***(Preprocessor)***

A character has been seen after the token catenation operator ## that is neither a letter nor a digit. Since the result of this operator must be a legal token, the operands must be tokens containing only letters and digits, e.g.:

```
/* the ' character will not lead to a valid token */
#define cc(a, b) a ## 'b
```

(138) strange character (*) after ## ***(Preprocessor)***

An unprintable character has been seen after the token catenation operator ## that is neither a letter nor a digit. Since the result of this operator must be a legal token, the operands must be tokens containing only letters and digits, e.g.:

```
/* the ' character will not lead to a valid token */
#define cc(a, b) a ## 'b
```

(139) end of file in comment ***(Preprocessor)***

End of file was encountered inside a comment. Check for a missing closing comment flag, e.g.:

```
/* Here the comment begins. I'm not sure where I end, though
}
```

(140) can't open * file "*": * ***(Driver, Preprocessor, Code Generator, Assembler)***

The command file specified could not be opened for reading. Confirm the spelling and path of the file specified on the command line, e.g.:

```
picc @communds
```

should that be:

```
picc @commands
```

(141) can't open * file "*": * ***(Any)***

An output file could not be created. Confirm the spelling and path of the file specified on the command line.

(144) too many nested #if blocks ***(Preprocessor)***

`#if`, `#ifdef` etc. blocks may only be nested to a maximum of 32.

(146) #include filename too long ***(Preprocessor)***

A filename constructed while looking for an include file has exceeded the length of an internal buffer. Since this buffer is 4096 bytes long, this is unlikely to happen.

(147) too many #include directories specified ***(Preprocessor)***

A maximum of 7 directories may be specified for the preprocessor to search for include files. The number of directories specified with the driver is too great.

(148) too many arguments for preprocessor macro ***(Preprocessor)***

A macro may only have up to 31 parameters, as per the C Standard.

(149) preprocessor macro work area overflow ***(Preprocessor)***

The total length of a macro expansion has exceeded the size of an internal table. This table is normally 32768 bytes long. Thus any macro expansion must not expand into a total of more than 32K bytes.

(150) illegal "__" preprocessor macro "*" ***(Preprocessor)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(151) too many arguments in preprocessor macro expansion ***(Preprocessor)***

There were too many arguments supplied in a macro invocation. The maximum number allowed is 31.

(152) bad dp/nargs in openpar(): c = * ***(Preprocessor)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(153) out of space in preprocessor macro "*" argument expansion ***(Preprocessor)***

A macro argument has exceeded the length of an internal buffer. This buffer is normally 4096 bytes long.

(155) work buffer overflow conatenating "*" ***(Preprocessor)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(156) work buffer "*" overflow ***(Preprocessor)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(157) can't allocate * bytes of memory ***(Code Generator, Assembler, Optimiser)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(158) invalid disable in preprocessor macro "*" ***(Preprocessor)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(159) too many calls to unget() ***(Preprocessor)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(161) control line "*" within preprocessor macro expansion ***(Preprocessor)***

A preprocessor control line (one starting with a #) has been encountered while expanding a macro. This should not happen.

(162) #warning: * ***(Preprocessor, Driver)***

This warning is either the result of user-defined `#warning` preprocessor directive or the driver encountered a problem reading the the map file. If the latter then please HI-TECH Software technical support with details

(163) unexpected text in control line ignored ***(Preprocessor)***

This warning occurs when extra characters appear on the end of a control line, e.g. The extra text will be ignored, but a warning is issued. It is preferable (and in accordance with Standard C) to enclose the text as a comment, e.g.:

```
#if defined(END)
  #define NEXT
#endif END     /* END would be better in a comment here */
```

(164) #include filename "*" was converted to lower case ***(Preprocessor)***

The `#include` file name had to be converted to lowercase before it could be opened, e.g.:

```
#include <STDIO.H>  /* oops -- should be: #include <stdio.h> */
```

(165) #include filename "*" does not match actual name (check upper/lower case) ***(Preprocessor)***

In Windows versions this means the file to be included actually exists and is spelt the same way as the `#include` filename, however the case of each does not exactly match. For example, specifying `#include "code.c"` will include `Code.c` if it is found. In Linux versions this warning could occur if the file wasn't found.

(166) too few values specified with option "*" ***(Preprocessor)***

The list of values to the preprocessor (CPP) `-S` option is incomplete. This should not happen if the preprocessor is being invoked by the compiler driver. The values passes to this option represent the sizes of `char`, `short`, `int`, `long`, `float` and `double` types.

(167) too many values specified with -S option; "*" unused ***(Preprocessor)***

There were too many values supplied to the -S preprocessor option. See the Error Message `-s, too few values specified in *` on page 360.

(168) unknown option "*" ***(Any)***

This option given to the component which caused the error is not recognized.

(169) strange character (*) after ## ***(Preprocessor)***

There is an unexpected character after `#`.

(170) symbol "*" in undef was never defined ***(Preprocessor)***

The symbol supplied as argument to `#undef` was not already defined. This warning may be disabled with some compilers. This warning can be avoided with code like:

```
#ifdef SYM
  #undef SYM  /* only undefine if defined */
#endif
```

(171) wrong number of preprocessor macro arguments for "*" (* instead of *)* ***(Preprocessor)***

A macro has been invoked with the wrong number of arguments, e.g.:

```
#define ADD(a, b) (a+b)
ADD(1, 2, 3)        /* oops -- only two arguments required */
```

(172) formal parameter expected after # ***(Preprocessor)***

The stringization operator `#` (not to be confused with the leading `#` used for preprocessor control lines) must be followed by a formal macro parameter, e.g.:

```
#define str(x) #y  /* oops -- did you mean x instead of y? */
```

If you need to stringize a token, you will need to define a special macro to do it, e.g.

```
#define __mkstr__(x) #x
```

then use `__mkstr__(token)` wherever you need to convert a token into a string.

(173) undefined symbol "*" in #if, 0 used ***(Preprocessor)***

A symbol on a `#if` expression was not a defined preprocessor macro. For the purposes of this expression, its value has been taken as zero. This warning may be disabled with some compilers. Example:

```
#if FOO+BAR    /* e.g. FOO was never #defined */
  #define GOOD
#endif
```

(174) multi-byte constant "*" isn't portable ***(Preprocessor)***

Multi-byte constants are not portable, and in fact will be rejected by later passes of the compiler, e.g.:

```
#if CHAR == 'ab'
  #define MULTI
#endif
```

(175) division by zero in #if; zero result assumed ***(Preprocessor)***

Inside a `#if` expression, there is a division by zero which has been treated as yielding zero, e.g.:

```
#if foo/0  /* divide by 0: was this what you were intending? */
  int a;
#endif
```

(176) missing newline ***(Preprocessor)***

A new line is missing at the end of the line. Each line, including the last line, must have a new line at the end. This problem is normally introduced by editors.

(177) symbol "*" in -U option was never defined ***(Preprocessor)***

A macro name specified in a `-U` option to the preprocessor was not initially defined, and thus cannot be undefined.

(179) nested comments ***(Preprocessor)***

This warning is issued when nested comments are found. A nested comment may indicate that a previous closing comment marker is missing or malformed, e.g.:

```
output = 0;  /* a comment that was left unterminated
flag = TRUE; /* next comment:
                hey, where did this line go? */
```

(180) unterminated comment in included file ***(Preprocessor)***

Comments begun inside an included file must end inside the included file.

(181) non-scalar types can't be converted to other types ***(Parser)***

You can't convert a structure, union or array to another type, e.g.:

```
struct TEST test;
struct TEST * sp;
sp = test;       /* oops -- did you mean: sp = &test; ? */
```

(182) illegal conversion between types ***(Parser)***

This expression implies a conversion between incompatible types, e.g. a conversion of a structure type into an integer, e.g.:

```
struct LAYOUT layout;
int i;
layout = i;          /* int cannot be converted to struct */
```

Note that even if a structure only contains an `int`, for example, it cannot be assigned to an `int` variable, and vice versa.

(183) function or function pointer required ***(Parser)***

Only a function or function pointer can be the subject of a function call, e.g.:

```
int a, b, c, d;
a = b(c+d);   /* b is not a function --
                 did you mean a = b*(c+d) ? */
```

(184) calling an interrupt function is illegal ***(Parser)***

A function qualified `interrupt` can't be called from other functions. It can only be called by a hardware (or software) interrupt. This is because an `interrupt` function has special function entry and exit code that is appropriate only for calling from an interrupt. An `interrupt` function can call other non-`interrupt` functions.

(185) function does not take arguments ***(Parser, Code Generator)***

This function has no parameters, but it is called here with one or more arguments, e.g.:

```
int get_value(void);
void main(void)
{
  int input;
  input = get_value(6);  /* oops --
                            parameter should not be here */
}
```

(186) too many function arguments ***(Parser)***

This function does not accept as many arguments as there are here.

```
void add(int a, int b);
add(5, 7, input);      /* call has too many arguments */
```

(187) too few function arguments ***(Parser)***

This function requires more arguments than are provided in this call, e.g.:

```
void add(int a, int b);
add(5);                  /* this call needs more arguments */
```

(188) constant expression required ***(Parser)***

In this context an expression is required that can be evaluated to a constant at compile time, e.g.:

```
int a;
switch(input) {
  case a:  /* oops!
              can't use variable as part of a case label */
    input++;
}
```

(189) illegal type for array dimension ***(Parser)***

An array dimension must be either an integral type or an enumerated value.

```
int array[12.5];  /* oops -- twelve and a half elements, eh? */
```

(190) illegal type for index expression ***(Parser)***

An index expression must be either integral or an enumerated value, e.g.:

```
int i, array[10];
i = array[3.5];   /* oops --
                     exactly which element do you mean? */
```

(191) cast type must be scalar or void ***(Parser)***

A typecast (an abstract type declarator enclosed in parentheses) must denote a type which is either scalar (i.e. not an array or a structure) or the type `void`, e.g.:

```
lip = (long [])input;  /* oops -- maybe: lip = (long *)input */
```

(192) undefined identifier "*" ***(Parser)***

This symbol has been used in the program, but has not been defined or declared. Check for spelling errors if you think it has been defined.

(193) not a variable identifier "*" ***(Parser)***

This identifier is not a variable; it may be some other kind of object, e.g. a label.

(194) ")" expected ***(Parser)***

A *closing parenthesis*, `)` , was expected here. This may indicate you have left out this character in an expression, or you have some other syntax error. The error is flagged on the line at which the code first starts to make no sense. This may be a statement following the incomplete expression, e.g.:

```
if(a == b  /* the closing parenthesis is missing here */
  b = 0;   /* the error is flagged here */
```

(195) expression syntax ***(Parser)***

This expression is badly formed and cannot be parsed by the compiler, e.g.:

```
a /=% b;  /* oops -- maybe that should be: a /= b; */
```

(196) struct/union required ***(Parser)***

A structure or union identifier is required before a dot `.`, e.g.:

```
int a;
a.b = 9;  /* oops -- a is not a structure */
```

(197) struct/union member expected ***(Parser)***

A structure or union member name must follow a dot (".") or arrow ("->").

(198) undefined struct/union "*" ***(Parser)***

The specified structure or union tag is undefined, e.g.

```
struct WHAT what;  /* a definition for WHAT was never seen */
```

(199) logical type required ***(Parser)***

The expression used as an operand to `if`, `while` statements or to boolean operators like `!` and `&&` must be a scalar integral type, e.g.:

```
struct FORMAT format;
if(format)            /* this operand must be a scaler type */
  format.a = 0;
```

(200) taking the address of a register variable is illegal ***(Parser)***

A variable declared `register` may not have storage allocated for it in memory, and thus it is illegal to attempt to take the address of it by applying the `&` operator, e.g.:

```
int * proc(register int in)
{
  int * ip = &in;
  /* oops -- in may not have an address to take */
  return ip;
}
```

(201) taking the address of this object is illegal *(Parser)*

The expression which was the operand of the `&` operator is not one that denotes memory storage ("an lvalue") and therefore its address can not be defined, e.g.:

```
ip = &8;  /* oops -- you can't take the address of a literal */
```

(202) only lvalues may be assigned to or modified *(Parser)*

Only an lvalue (i.e. an identifier or expression directly denoting addressable storage) can be assigned to or otherwise modified, e.g.:

```
int array[10];
int * ip;
char c;
array = ip;   /* array isn't a variable,
                 it can't be written to */
```

A typecast does not yield an lvalue, e.g.:

```
/* the contents of c cast to int
   is only a intermediate value */
(int)c = 1;
```

However you can write this using pointers:

```
*(int *)&c = 1
```

(203) illegal operation on bit variable *(Parser)*

Not all operations on `bit` variables are supported. This operation is one of those, e.g.:

```
bit   b;
int * ip;
ip = &b;  /* oops --
             cannot take the address of a bit object */
```

(204) void function can't return a value ***(Parser)***

A void function cannot return a value. Any `return` statement should not be followed by an expression, e.g.:

```
void run(void)
{
  step();
  return 1;
  /* either run should not be void, or remove the 1 */
}
```

(205) integral type required ***(Parser)***

This operator requires operands that are of integral type only.

(206) illegal use of void expression ***(Parser)***

A `void` expression has no value and therefore you can't use it anywhere an expression with a value is required, e.g. as an operand to an arithmetic operator.

(207) simple type required for "*" ***(Parser)***

A simple type (i.e. not an array or structure) is required as an operand to this operator.

(208) operands of "*" not same type ***(Parser)***

The operands of this operator are of different pointer, e.g.:

```
int * ip;
char * cp, * cp2;
cp = flag ? ip : cp2;
/* result of ? : will be int * or char * */
```

Maybe you meant something like:

```
cp = flag ? (char *)ip : cp2;
```

(209) type conflict ***(Parser)***

The operands of this operator are of incompatible types.

(210) bad size list ***(Parser)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(211) taking sizeof bit is illegal ***(Parser)***

It is illegal to use the `sizeof` operator with the HI-TECH C `bit` type. When used against a type the `sizeof` operator gives the number of bytes required to store an object that type. Therefore its usage with the `bit` type make no sense and is an illegal operation.

(212) missing number after pragma "pack" ***(Parser)***

The pragma `pack` requires a decimal number as argument. This specifies the alignment of each member within the structure. Use this with caution as some processors enforce alignment and will not operate correctly if word fetches are made on odd boundaries, e.g.:

```
#pragma pack  /* what is the alignment value */
```

Maybe you meant something like:

```
#pragma pack 2
```

(214) missing number after pragma "interrupt_level" ***(Parser)***

The pragma `interrupt_level` requires an argument from 0 to 7.

(215) missing argument to pragma "switch" ***(Parser)***

The pragma switch requires an argument of `auto`, `direct` or `simple`, e.g.:

```
#pragma switch  /* oops -- this requires a switch mode */
```

maybe you meant something like:

```
#pragma switch simple
```

(216) missing argument to pragma "psect" ***(Parser)***

The pragma `psect` requires an argument of the form *`oldname=newname`* where *`oldname`* is an existing psect name known to the compiler, and *`newname`* is the desired new name, e.g.:

```
#pragma psect  /* oops -- this requires an psect to redirect */
```

maybe you meant something like:

```
#pragma psect text=specialtext
```

(218) missing name after pragma "inline" ***(Parser)***

The `inline` pragma expects the name of a function to follow. The function name must be recognized by the code generator for it to be expanded; other functions are not altered, e.g.:

```
#pragma inline  /* what is the function name? */
```

maybe you meant something like:

```
#pragma inline memcpy
```

(219) missing name after pragma "printf_check" ***(Parser)***

The `printf_check` pragma expects the name of a function to follow. This specifies printf-style format string checking for the function, e.g.

```
#pragma printf_check  /* what function is to be checked? */
```

Maybe you meant something like:

```
#pragma printf_check sprintf
```

Pragmas for all the standard printf-like function are already contained in `<stdio.h>`.

(220) exponent expected ***(Parser)***

A floating point constant must have at least one digit after the `e` or `E`., e.g.:

```
float f;
f = 1.234e;  /* oops -- what is the exponent? */
```

(221) hexadecimal digit expected ***(Parser)***

After `0x` should follow at least one of the hex digits `0-9` and `A-F` or `a-f`, e.g.:

```
a = 0xg6;  /* oops -- was that meant to be a = 0xf6 ? */
```

(222) binary digit expected ***(Parser)***

A binary digit was expected following the `0b` format specifier, e.g.

```
i = 0bf000;  /* wooops -- f000 is not a base two value */
```

(223) digit out of range ***(Parser, Assembler, Optimiser)***

A digit in this number is out of range of the radix for the number, e.g. using the digit 8 in an octal number, or hex digits A-F in a decimal number. An octal number is denoted by the digit string commencing with a zero, while a hex number starts with "0X" or "0x". For example:

```
int a = 058;
/* leading 0 implies octal which has digits 0 - 7 */
```

(224) illegal "#" directive ***(Parser)***

An illegal # preprocessor has been detected. Likely a directive has been misspelt in your code somewhere.

(225) missing character in character constant **(Parser)**

The character inside the single quotes is missing, e.g.:

```
char c = ';  /* the character value of what? */
```

(226) char const too long ***(Parser)***

A character constant enclosed in single quotes may not contain more than one character, e.g.:

```
c = '12';  /* oops -- only one character may be specified */
```

(227) "." expected after ".." ***(Parser)***

The only context in which two successive dots may appear is as part of the *ellipsis* symbol, which must have 3 dots. (An *ellipsis* is used in function prototypes to indicate a variable number of parameters.)

Either `..` was meant to be an *ellipsis* symbol which would require you to add an extra dot, or it was meant to be a *structure member operator* which would require you remove one dot.

(228) illegal character (*) ***(Parser)***

This character is illegal in the C code. Valid characters are the letters, digits and those comprising the acceptable operators, e.g.:

```
c = `a`;  /* oops -- did you mean c = 'a'; ? */
```

(229) unknown qualifier "*" given to -A ***(Parser)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(230) missing argument to -A ***(Parser)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(231) unknown qualifier "*" given to -I ***(Parser)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(232) missing argument to -I ***(Parser)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(233) bad -Q option "*" ***(Parser)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(234) close error ***(Parser)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(236) simple integer expression required ***(Parser)***

A simple integral expression is required after the operator `@`, used to associate an absolute address with a variable, e.g.:

```
int address;
char LOCK @ address;
```

(237) function "*" redefined *(Parser)*

More than one definition for a function has been encountered in this module. Function overloading is illegal, e.g.:

```
int twice(int a)
{
  return a*2;
}
/* only one prototype & definition of rv can exist */
long twice(long a)
{
  return a*2;
}
```

(238) illegal initialisation *(Parser)*

You can't initialise a `typedef` declaration, because it does not reserve any storage that can be initialised, e.g.:

```
/* oops -- uint is a type, not a variable */
typedef unsigned int uint = 99;
```

(239) identifier "*" redefined (from line *) *(Parser)*

This identifier has already been defined in the same scope. It cannot be defined again, e.g.:

```
  int a;  /* a filescope variable called "a" */
  int a;  /* attempting to define another of the same name */
```

Note that variables with the same name, but defined with different scopes are legal, but not recommended.

(240) too many initializers *(Parser)*

There are too many initializers for this object. Check the number of initializers against the object definition (array or structure), e.g.:

```
/* three elements, but four initializers */
int ivals[3] = { 2, 4, 6, 8};
```

(241) initialization syntax ***(Parser)***

The initialisation of this object is syntactically incorrect. Check for the correct placement and number of braces and commas, e.g.:

```
int iarray[10] = {{'a', 'b', 'c'};
/* oops -- one two many {s */
```

(242) illegal type for switch expression ***(Parser)***

A `switch` operation must have an expression that is either an integral type or an enumerated value, e.g:

```
double d;
switch(d) {  /* oops -- this must be integral */
  case '1.0':
    d = 0;
}
```

(243) inappropriate break/continue ***(Parser)***

A `break` or `continue` statement has been found that is not enclosed in an appropriate control structure. A `continue` can only be used inside a `while`, `for` or `do while` loop, while `break` can only be used inside those loops or a `switch` statement, e.g.:

```
switch(input) {
  case 0:
    if(output == 0)
      input = 0xff;
    } /* oops! this shouldn't be here and closed the switch */
    break;     /* this should be inside the switch */
```

(244) "default" case redefined ***(Parser)***

There is only allowed to be one `default` label in a switch statement. You have more than one, e.g.:

```
switch(a) {
default:     /* if this is the default case... */
  b = 9;
  break;
default:     /* then what is this? */
```

```
  b = 10;
  break;
```

(245) "default" case not in switch ***(Parser)***

A label has been encountered called `default` but it is not enclosed by a `switch` statement. A `default` label is only legal inside the body of a `switch` statement.

If there is a `switch` statement before this `default` label, there may be one too many closing braces in the `switch` code which would prematurely terminate the `switch` statement. See example for Error Message `'case' not in switch` on page 375.

(246) case label not in switch ***(Parser)***

A `case` label has been encountered, but there is no enclosing `switch` statement. A `case` label may only appear inside the body of a `switch` statement.

If there is a `switch` statement before this `case` label, there may be one too many closing braces in the `switch` code which would prematurely terminate the `switch` statement, e.g.:

```
switch(input) {
  case '0':
    count++;
    break;
  case '1':
    if(count>MAX)
      count= 0;
    }          /* oops -- this shouldn't be here */
    break;
  case '2':    /* error flagged here */
```

(247) duplicate label "*" ***(Parser)***

The same name is used for a label more than once in this function. Note that the scope of labels is the entire function, not just the block that encloses a label, e.g.:

```
start:
  if(a > 256)
    goto end;
start:           /* error flagged here */
  if(a == 0)
    goto start;  /* which start label do I jump to? */
```

(248) inappropriate "else" ***(Parser)***

An `else` keyword has been encountered that cannot be associated with an `if` statement. This may mean there is a missing brace or other syntactic error, e.g.:

```
/* here is a comment which I have forgotten to close...
if(a > b) {
  c = 0;
/* ... that will be closed here, thus removing the "if" */
else        /* my "if" has been lost */
  c = 0xff;
```

(249) probable missing "}" in previous block ***(Parser)***

The compiler has encountered what looks like a function or other declaration, but the preceding function has not been ended with a closing brace. This probably means that a closing brace has been omitted from somewhere in the previous function, although it may well not be the last one, e.g.:

```
void set(char a)
{
  PORTA = a;
                  /* the closing brace was left out here */
void clear(void)  /* error flagged here */
{
  PORTA = 0;
}
```

(251) array dimension redeclared ***(Parser)***

An array dimension has been declared as a different non-zero value from its previous declaration. It is acceptable to redeclare the size of an array that was previously declared with a zero dimension, but not otherwise, e.g.:

```
extern int array[5];
int array[10];        /* oops -- has it 5 or 10 elements? */
```

(252) argument * conflicts with prototype ***(Parser)***

The argument specified (argument 0 is the left most argument) of this function definition does not agree with a previous prototype for this function, e.g.:

```
/* this is supposedly calc's prototype */
extern int calc(int, int);
int calc(int a, long int b)  /* hmmm -- which is right? */
{                            /* error flagged here */
  return sin(b/a);
}
```

(253) argument list conflicts with prototype ***(Parser)***

The argument list in a function definition is not the same as a previous prototype for that function. Check that the number and types of the arguments are all the same.

```
extern int calc(int);   /* this is supposedly calc's prototype */
int calc(int a, int b)  /* hmmm -- which is right? */
{                       /* error flagged here */
  return a + b;
}
```

(254) undefined *: "*" ***(Parser)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(255) not a member of the struct/union "*" ***(Parser)***

This identifier is not a member of the structure or union type with which it used here, e.g.:

```
struct {
  int a, b, c;
} data;
if(data.d)    /* oops --
                 there is no member d in this structure */
  return;
```

(256) too much indirection ***(Parser)***

A pointer declaration may only have 16 levels of indirection.

(257) only "register" storage class allowed ***(Parser)***

The only storage class allowed for a function parameter is `register`, e.g.:

```
void process(static int input)
```

(258) duplicate qualifier ***(Parser)***

There are two occurrences of the same qualifier in this type specification. This can occur either directly or through the use of a typedef. Remove the redundant qualifier. For example:

```
typedef volatile int vint;
/* oops -- this results in two volatile qualifiers */
volatile vint very_vol;
```

(259) can't be qualifed both far and near ***(Parser)***

It is illegal to qualify a type as both `far` and `near`, e.g.:

```
far near int spooky;  /* oops -- choose far or near, not both */
```

(260) undefined enum tag "*" ***(Parser)***

This enum tag has not been defined, e.g.:

```
enum WHAT what;  /* a definition for WHAT was never seen */
```

(261) struct/union member "*" redefined ***(Parser)***

This name of this member of the struct or union has already been used in this `struct` or `union`, e.g.:

```
struct {
  int a;
  int b;
  int a;  /* oops -- a different name is required here */
} input;
```

(262) struct/union "*" redefined ***(Parser)***

A structure or union has been defined more than once, e.g.:

```
struct {
  int a;
} ms;
struct {
  int a;
} ms;    /* was this meant to be the same name as above? */
```

(263) members can't be functions ***(Parser)***

A member of a structure or a union may not be a function. It may be a pointer to a function, e.g.:

```
struct {
  int a;
  int get(int);  /* should be a pointer: int (*get)(int); */
} object;
```

(264) bad bitfield type ***(Parser)***

A bitfield may only have a type of `int` (`signed` or `unsigned`), e.g.:

```
struct FREG {
  char b0:1;   /* these must be part of an int, not char */
  char   :6;
  char b7:1;
} freg;
```

(265) integer constant expected ***(Parser)***

A *colon* appearing after a member name in a structure declaration indicates that the member is a bitfield. An integral constant must appear after the *colon* to define the number of bits in the bitfield, e.g.:

```
struct {
  unsigned first:  /* oops -- should be: unsigned first; */
  unsigned second;
} my_struct;
```

If this was meant to be a structure with bitfields, then the following illustrates an example:

```
struct {
  unsigned first : 4;  /* 4 bits wide */
  unsigned second: 4;  /* another 4 bits */
} my_struct;
```

(266) storage class illegal ***(Parser)***

A structure or union member may not be given a storage class. Its storage class is determined by the storage class of the structure, e.g.:

```
struct {
  /* no additional qualifiers may be present with members */
  static int first;
} ;
```

(267) bad storage class ***(Code Generator)***

The code generator has encountered a variable definition whose storage class is invalid, e.g.:

```
auto int foo; /* auto not permitted with global variables */
int power(static int a)  /* parameters may not be static */
{
  return foo * a;
}
```

(268) inconsistent storage class ***(Parser)***

A declaration has conflicting storage classes. Only one storage class should appear in a declaration, e.g.:

```
extern static int where;  /* so is it static or extern? */
```

(269) inconsistent type ***(Parser)***

Only one basic type may appear in a declaration, e.g.:

```
int float if;  /* is it int or float? */
```

(270) variable can't have storage class "register" ***(Parser)***

Only function parameters or `auto` variables may be declared using the `register` qualifier, e.g.:

```
register int gi;      /* this cannot be qualified register */
int process(register int input)  /* this is okay */
{
  return input + gi;
}
```

(271) type can't be long ***(Parser)***

Only `int` and `float` can be qualified with `long`.

```
long char lc;  /* what? */
```

(272) type can't be short ***(Parser)***

Only `int` can be modified with `short`, e.g.:

```
short float sf;  /* what? */
```

(273) type can't be both signed and unsigned ***(Parser)***

The type modifiers `signed` and `unsigned` cannot be used together in the same declaration, as they have opposite meaning, e.g.:

```
signed unsigned int confused;  /* which is it? */
```

(274) type can't be unsigned ***(Parser)***

A floating point type cannot be made `unsigned`, e.g.:

```
unsigned float uf;  /* what? */
```

(275) "..." illegal in non-prototype argument list ***(Parser)***

The *ellipsis* symbol may only appear as the last item in a prototyped argument list. It may not appear on its own, nor may it appear after argument names that do not have types, i.e. K&R-style non-prototype function definitions. For example:

```
/* K&R-style non-prototyped function definition */
int kandr(a, b, ...)
  int a, b;
{
```

(276) type specifier required for prototyped argument ***(Parser)***

A type specifier is required for a prototyped argument. It is not acceptable to just have an identifier.

(277) can't mix prototyped and non-prototyped arguments ***(Parser)***

A function declaration can only have all prototyped arguments (i.e. with types inside the parentheses) or all K&R style args (i.e. only names inside the parentheses and the argument types in a declaration list before the start of the function body), e.g.:

```
int plus(int a, b)  /* oops -- a is prototyped, b is not */
int b;
{
  return a + b;
}
```

(278) argument "*" redeclared ***(Parser)***

The specified argument is declared more than once in the same argument list, e.g.

```
/* can't have two parameters called "a" */
int calc(int a, int a)
```

(279) initialization of function arguments is illegal ***(Parser)***

A function argument can't have an initialiser in a declaration. The initialisation of the argument happens when the function is called and a value is provided for the argument by the calling function, e.g.:

```
/* oops -- a is initialized when proc is called */
extern int proc(int a = 9);
```

(280) arrays of functions are illegal ***(Parser)***

You can't define an array of functions. You can however define an array of pointers to functions, e.g.:

```
int * farray[]();  /* oops -- should be: int (* farray[])(); */
```

(281) functions can't return functions ***(Parser)***

A function cannot return a function. It can return a function pointer. A function returning a pointer to a function could be declared like this: int (* (name()))(). Note the many parentheses that are necessary to make the parts of the declaration bind correctly.

(282) functions can't return arrays ***(Parser)***

A function can return only a scalar (simple) type or a structure. It cannot return an array.

(283) dimension required ***(Parser)***

Only the most significant (i.e. the first) dimension in a multi-dimension array may not be assigned a value. All succeeding dimensions must be present as a constant expression, e.g.:

```
/* This should be, e.g.: int arr[][7] */
int get_element(int arr[2][])
{
  return array[1][6];
}
```

(284) invalid dimension ***(Parser)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(285) no identifier in declaration ***(Parser)***

The identifier is missing in this declaration. This error can also occur where the compiler has been confused by such things as missing closing braces, e.g.:

```
void interrupt(void)  /* what is the name of this function? */
{
}
```

(286) declarator too complex *(Parser)*

This declarator is too complex for the compiler to handle. Examine the declaration and find a way to simplify it. If the compiler finds it too complex, so will anybody maintaining the code.

(287) arrays of bits or pointers to bit are illegal *(Parser)*

It is not legal to have an array of bits, or a pointer to bit variable, e.g.:

```
bit barray[10];  /* wrong -- no bit arrays */
bit * bp;        /* wrong -- no pointers to bit variables */
```

(288) only functions may be void *(Parser)*

A variable may not be `void`. Only a function can be `void`, e.g.:

```
int a;
void b;  /* this makes no sense */
```

(289) only functions may be qualified "interrupt" *(Parser)*

The qualifier `interrupt` may not be applied to anything except a function, e.g.:

```
/* variables cannot be qualified interrupt */
interrupt int input;
```

(290) illegal function qualifier(s) *(Parser)*

A qualifier has been applied to a function which makes no sense in this context. Some qualifier only make sense when used with an lvalue, e.g. const or volatile. This may indicate that you have forgotten out a star `*` indicating that the function should return a pointer to a qualified object, e.g.

```
const char ccrv(void) /* const * char ccrv(void) perhaps? */
{                     /* error flagged here */
  return ccip;
}
```

(291) K&R identifier "*" not an argument ***(Parser)***

This identifier that has appeared in a K&R style argument declarator is not listed inside the parentheses after the function name, e.g.:

```
int process(input)
int unput;        /* oops -- that should be int input; */
{
}
```

(292) function parameter may not be a function ***(Parser)***

A function parameter may not be a function. It may be a pointer to a function, so perhaps a "*" has been omitted from the declaration.

(293) bad size in index_type() ***(Parser)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(294) can't allocate * bytes of memory ***(Code Generator, Hexmate)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(295) expression too complex ***(Parser)***

This expression has caused overflow of the compiler's internal stack and should be re-arranged or split into two expressions.

(296) out of memory ***(Objtohex)***

This could be an internal compiler error. Contact HI-TECH Software technical support with details.

(297) bad argument (*) to tysize() ***(Parser)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(298) end of file in #asm ***(Preprocessor)***

An end of file has been encountered inside a `#asm` block. This probably means the `#endasm` is missing or misspelt, e.g.:

```
#asm
  mov  r0, #55
  mov  [r1], r0
}                /* oops -- where is the #endasm */
```

(300) unexpected end of file *(Parser)*

An end-of-file in a C module was encountered unexpectedly, e.g.:

```
void main(void)
{
  init();
  run();    /* is that it? What about the close brace */
```

(301) end of file on string file *(Parser)*

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(302) can't reopen "*": * *(Parser)*

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(303) can't allocate * bytes of memory (line *) *(Parser)*

The parser was unable to allocate memory for the longest string encountered, as it attempts to sort and merge strings. Try reducing the number or length of strings in this module.

(306) can't allocate * bytes of memory for * *(Code Generator)*

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(307) too many qualifier names *(Parser)*

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(308) too many case labels in switch *(Code Generator)*

There are too many `case` labels in this `switch` statement. The maximum allowable number of `case` labels in any one `switch` statement is 511.

(309) too many symbols ***(Assembler)***

There are too many symbols for the assembler's symbol table. Reduce the number of symbols in your program.

(310) "]" expected ***(Parser)***

A closing square bracket was expected in an array declaration or an expression using an array index, e.g.

```
process(carray[idx);  /* oops --
                         should be: process(carray[idx]); */
```

(311) closing quote expected ***(Parser)***

A closing quote was expected for the indicated string.

(312) "*" expected ***(Parser)***

The indicated token was expected by the parser.

(313) function body expected ***(Parser)***

Where a function declaration is encountered with K&R style arguments (i.e. argument names but no types inside the parentheses) a function body is expected to follow, e.g.:

```
/* the function block must follow, not a semicolon */
int get_value(a, b);
```

(314) ";" expected ***(Parser)***

A *semicolon* is missing from a statement. A close brace or keyword was found following a statement with no terminating *semicolon*, e.g.:

```
while(a) {
  b = a--  /* oops -- where is the semicolon? */
}          /* error is flagged here */
```

Note: Omitting a semicolon from statements not preceding a close brace or keyword typically results in some other error being issued for the following code which the parser assumes to be part of the original statement.

(315) "{" expected ***(Parser)***

An *opening brace* was expected here. This error may be the result of a function definition missing the *opening brace*, e.g.:

```
/* oops! no opening brace after the prototype */
void process(char c)
  return max(c, 10) * 2; /* error flagged here */
}
```

(316) "}" expected ***(Parser)***

A *closing brace* was expected here. This error may be the result of a initialized array missing the *closing brace*, e.g.:

```
char carray[4] = { 1, 2, 3, 4;  /* oops -- no closing brace */
```

(317) "(" expected ***(Parser)***

An *opening parenthesis*, `(`, was expected here. This must be the first token after a `while`, `for`, `if`, `do` or `asm` keyword, e.g.:

```
if a == b  /* should be: if(a == b) */
  b = 0;
```

(318) string expected ***(Parser)***

The operand to an `asm` statement must be a string enclosed in parentheses, e.g.:

```
asm(nop);  /* that should be asm("nop");
```

(319) while expected ***(Parser)***

The keyword `while` is expected at the end of a `do` statement, e.g.:

```
do {
  func(i++);
}             /* do the block while what condition is true? */
if(i > 5)     /* error flagged here */
  end();
```

(320) ":" expected ***(Parser)***

A *colon* is missing after a `case` label, or after the keyword `default`. This often occurs when a *semicolon* is accidentally typed instead of a *colon*, e.g.:

```
switch(input) {
  case 0;          /* oops -- that should have been: case 0: */
    state = NEW;
```

(321) label identifier expected ***(Parser)***

An identifier denoting a label must appear after `goto`, e.g.:

```
if(a)
  goto 20;
/* this is not BASIC -- a valid C label must follow a goto */
```

(322) enum tag or "{" expected ***(Parser)***

After the keyword `enum` must come either an identifier that is or will be defined as an `enum` tag, or an opening brace, e.g.:

```
enum 1, 2;  /* should be, e.g.: enum {one=1, two }; */
```

(323) struct/union tag or "{" expected ***(Parser)***

An identifier denoting a structure or union or an opening brace must follow a `struct` or `union` keyword, e.g.:

```
struct int a;  /* this is not how you define a structure */
```

You might mean something like:

```
struct {
  int a;
} my_struct;
```

(324) too many arguments for printf-style format string ***(Parser)***

There are too many arguments for this format string. This is harmless, but may represent an incorrect format string, e.g.:

```
/* oops -- missed a placeholder? */
printf("%d - %d", low, high, median);
```

(325) error in printf-style format string ***(Parser)***

There is an error in the format string here. The string has been interpreted as a `printf()` style format string, and it is not syntactically correct. If not corrected, this will cause unexpected behaviour at run time, e.g.:

```
printf("%l", lll);  /* oops -- maybe: printf("%ld", lll); */
```

(326) long int argument required in printf-style format string ***(Parser)***

A long argument is required for this format specifier. Check the number and order of format specifiers and corresponding arguments, e.g.:

```
printf("%lx", 2);  // maybe you meant: printf("%lx", 2L);
```

(327) long long int argument required in printf-style format string ***(Parser)***

A long long argument is required for this format specifier. Check the number and order of format specifiers and corresponding arguments, e.g.:

```
printf("%llx", 2);  // maybe you meant: printf("%llx", 2LL);
```

Note that not all HI-TECH C compilers provide support for a long long integer type.

(328) int argument required in printf-style format string ***(Parser)***

An integral argument is required for this printf-style format specifier. Check the number and order of format specifiers and corresponding arguments, e.g.:

```
printf("%d", 1.23); /* wrong number or wrong placeholder */
```

(329) double argument required in printf-style format string ***(Parser)***

The printf format specifier corresponding to this argument is `%f` or similar, and requires a floating point expression. Check for missing or extra format specifiers or arguments to printf.

```
printf("%f", 44);  /* should be: printf("%f", 44.0); */
```

(330) pointer to * argument required in printf-style format string ***(Parser)***

A pointer argument is required for this format specifier. Check the number and order of format specifiers and corresponding arguments.

(331) too few arguments for printf-style format string ***(Parser)***

There are too few arguments for this format string. This would result in a garbage value being printed or converted at run time, e.g.:

```
printf("%d - %d", low);
  /* oops! where is the other value to print? */
```

(332) "interrupt_level" should be 0 to 7 ***(Parser)***

The pragma `interrupt_level` must have an argument from 0 to 7, e.g.:

```
#pragma interrupt_level  /* oops -- what is the level */
void interrupt isr(void)
{
  /* isr code goes here */
}
```

(333) unrecognized qualifier name after "strings" ***(Parser)***

The `pragma strings` was passed a qualifier that was not identified, e.g.:

```
/* oops -- should that be #pragma strings const ? */
#pragma strings cinst
```

(334) unrecognized qualifier name after "printf_check" ***(Parser)***

The `#pragma printf_check` was passed a qualifier that could not be identified, e.g.:

```
/* oops -- should that be const not cinst? */
#pragma printf_check(printf) cinst
```

(335) unknown pragma "*" ***(Parser)***

An unknown `pragma` directive was encountered, e.g.:

```
#pragma rugsused w  /* I think you meant regsused */
```

(336) string concatenation across lines ***(Parser)***

Strings on two lines will be concatenated. Check that this is the desired result, e.g.:

```
char * cp = "hi"
  "there";    /* this is okay,
                 but is it what you had intended? */
```

(337) line does not have a newline on the end ***(Parser)***

The last line in the file is missing the *newline* (operating system dependent character) from the end. Some editors will create such files, which can cause problems for include files. The ANSI C standard requires all source files to consist of complete lines only.

(338) can't create * file "*" ***(Any)***

The application tried to create or open the named file, but it could not be created. Check that all file pathnames are correct.

(339) initializer in extern declaration ***(Parser)***

A declaration containing the keyword `extern` has an initialiser. This overrides the `extern` storage class, since to initialise an object it is necessary to define (i.e. allocate storage for) it, e.g.:

```
extern int other = 99;  /* if it's extern and not allocated
                           storage, how can it be initialized? */
```

(340) string not terminated by null character. ***(Parser)***

A char array is being initialized with a string literal larger than the array. Hence there is insufficient space in the array to safely append a null terminating character, e.g.:

```
char foo[5] = "12345"; /* the string stored in foo won't have
                          a null terminating, i.e.
                          foo = ['1', '2', '3', '4', '5'] */
```

(343) implicit return at end of non-void function *(Parser)*

A function which has been declared to return a value has an execution path that will allow it to reach the end of the function body, thus returning without a value. Either insert a `return` statement with a value, or if the function is not to return a value, declare it `void`, e.g.:

```
int mydiv(double a, int b)
{
  if(b != 0)
    return a/b;    /* what about when b is 0? */
}                  /* warning flagged here */
```

(344) non-void function returns no value *(Parser)*

A function that is declared as returning a value has a `return` statement that does not specify a return value, e.g.:

```
int get_value(void)
{
  if(flag)
    return val++;
  return;
  /* what is the return value in this instance? */
}
```

(345) unreachable code *(Parser)*

This section of code will never be executed, because there is no execution path by which it could be reached, e.g.:

```
while(1)           /* how does this loop finish? */
  process();
flag = FINISHED;   /* how do we get here? */
```

(346) declaration of "*" hides outer declaration *(Parser)*

An object has been declared that has the same name as an outer declaration (i.e. one outside and preceding the current function or block). This is legal, but can lead to accidental use of one variable when the outer one was intended, e.g.:

```
int input;          /* input has filescope */
void process(int a)
{
  int input;        /* local blockscope input */
  a = input;        /* this will use the local variable.
                       Is this right? */
```

(347) external declaration inside function ***(Parser)***

A function contains an `extern` declaration. This is legal but is invariably not desirable as it restricts the scope of the function declaration to the function body. This means that if the compiler encounters another declaration, use or definition of the extern object later in the same file, it will no longer have the earlier declaration and thus will be unable to check that the declarations are consistent. This can lead to strange behaviour of your program or signature errors at link time. It will also hide any previous declarations of the same thing, again subverting the compiler's type checking. As a general rule, always declare `extern` variables and functions outside any other functions. For example:

```
int process(int a)
{
  /* this would be better outside the function */
  extern int away;
  return away + a;
}
```

(348) auto variable "*" should not be qualified ***(Parser)***

An `auto` variable should not have qualifiers such as `near` or `far` associated with it. Its storage class is implicitly defined by the stack organization. An `auto` variable may be qualified with `static`, but it is then no longer `auto`.

(349) non-prototyped function declaration for "*" ***(Parser)***

A function has been declared using old-style (K&R) arguments. It is preferable to use prototype declarations for all functions, e.g.:

```
int process(input)
int input;          /* warning flagged here */
{
}
```

This would be better written:

```
int process(int input)
{
}
```

(350) unused * "*" (from line *) ***(Parser)***

The indicated object was never used in the function or module being compiled. Either this object is redundant, or the code that was meant to use it was excluded from compilation or misspelt the name of the object. Note that the symbols `rcsid` and `sccsid` are never reported as being unused.

(352) float parameter coerced to double ***(Parser)***

Where a non-prototyped function has a parameter declared as `float`, the compiler converts this into a `double float`. This is because the default C type conversion conventions provide that when a floating point number is passed to a non-prototyped function, it will be converted to `double`. It is important that the function declaration be consistent with this convention, e.g.:

```
double inc_flt(f)  /* f will be converted to double */
float f;           /* warning flagged here */
{
  return f * 2;
}
```

(353) sizeof external array "*" is zero ***(Parser)***

The size of an external array evaluates to zero. This is probably due to the array not having an explicit dimension in the extern declaration.

(354) possible pointer truncation ***(Parser)***

A pointer qualified far has been assigned to a default pointer or a pointer qualified near, or a default pointer has been assigned to a pointer qualified near. This may result in truncation of the pointer and loss of information, depending on the memory model in use.

(355) implicit signed to unsigned conversion ***(Parser)***

A `signed` number is being assigned or otherwise converted to a larger `unsigned` type. Under the ANSI "value preserving" rules, this will result in the `signed` value being first sign-extended to a `signed` number the size of the target type, then converted to `unsigned` (which involves no change in bit pattern). Thus an unexpected sign extension can occur. To ensure this does not happen, first convert the signed value to an unsigned equivalent, e.g.:

```
signed char sc;
unsigned int ui;
ui = sc;    /* if sc contains 0xff,
               ui will contain 0xffff for example */
```

will perform a sign extension of the `char` variable to the longer type. If you do not want this to take place, use a cast, e.g.:

```
ui = (unsigned char)sc;
```

(356) implicit conversion of float to integer ***(Parser)***

A floating point value has been assigned or otherwise converted to an integral type. This could result in truncation of the floating point value. A typecast will make this warning go away.

```
double dd;
int i;
i = dd;    /* is this really what you meant? */
```

If you do intend to use an expression like this, then indicate that this is so by a cast:

```
i = (int)dd;
```

(357) illegal conversion of integer to pointer ***(Parser)***

An integer has been assigned to or otherwise converted to a pointer type. This will usually mean you have used the wrong variable, but if this is genuinely what you want to do, use a typecast to inform the compiler that you want the conversion and the warning will be suppressed. This may also mean you have forgotten the `&` address operator, e.g.:

```
int * ip;
int i;
ip = i;    /* oops -- did you mean ip = &i ? */
```

If you do intend to use an expression like this, then indicate that this is so by a cast:

```
ip = (int *)i;
```

(358) illegal conversion of pointer to integer *(Parser)*

A pointer has been assigned to or otherwise converted to a integral type. This will usually mean you have used the wrong variable, but if this is genuinely what you want to do, use a typecast to inform the compiler that you want the conversion and the warning will be suppressed. This may also mean you have forgotten the * dereference operator, e.g.:

```
int * ip;
int i;
i = ip;     /* oops -- did you mean i = *ip ? */
```

If you do intend to use an expression like this, then indicate that this is so by a cast:

```
i = (int)ip;
```

(359) illegal conversion between pointer types *(Parser)*

A pointer of one type (i.e. pointing to a particular kind of object) has been converted into a pointer of a different type. This will usually mean you have used the wrong variable, but if this is genuinely what you want to do, use a typecast to inform the compiler that you want the conversion and the warning will be suppressed, e.g.:

```
long input;
char * cp;
cp = &input;  /* is this correct? */
```

This is common way of accessing bytes within a multi-byte variable. To indicate that this is the intended operation of the program, use a cast:

```
cp = (char *)&input;  /* that's better */
```

This warning may also occur when converting between pointers to objects which have the same type, but which have different qualifiers, e.g.:

```
char * cp;
/* yes, but what sort of characters? */
cp = "I am a string of characters";
```

If the default type for string literals is `const char *`, then this warning is quite valid. This should be written:

```
const char * cp;
cp = "I am a string of characters";  /* that's better */
```

Omitting a qualifier from a pointer type is often disastrous, but almost certainly not what you intend.

(360) array index out of bounds *(Parser)*

An array is being indexed with a constant value that is less than zero, or greater than or equal to the number of elements in the array. This warning will not be issued when accessing an array element via a pointer variable, e.g.:

```
int i, * ip, input[10];
i = input[-2];           /* oops -- this element doesn't exist */
ip = &input[5];
i = ip[-2];              /* this is okay */
```

(361) function declared implicit int *(Parser)*

Where the compiler encounters a function call of a function whose name is presently undefined, the compiler will automatically declare the function to be of type `int`, with unspecified (K&R style) parameters. If a definition of the function is subsequently encountered, it is possible that its type and arguments will be different from the earlier implicit declaration, causing a compiler error. The solution is to ensure that all functions are defined or at least declared before use, preferably with prototyped parameters. If it is necessary to make a forward declaration of a function, it should be preceded with the keywords `extern` or `static` as appropriate. For example:

```
/* I may prevent an error arising from calls below */
void set(long a, int b);
void main(void)
{
  /* by here a prototype for set should have seen */
  set(10L, 6);
}
```

(362) redundant "&" applied to array *(Parser)*

The address operator `&` has been applied to an array. Since using the name of an array gives its address anyway, this is unnecessary and has been ignored, e.g.:

```
int array[5];
int * ip;
/* array is a constant, not a variable; the & is redundant. */
ip = &array;
```

(363) redundant "&" or "*" applied to function address *(Parser)*

The address operator "&" has been applied to a function. Since using the name of a function gives its address anyway, this is unnecessary and has been ignored, e.g.:

```
extern void foo(void);
void main(void)
{
    void(*bar)(void);
    /* both assignments are equivalent */
    bar = &foo;
    bar = foo;  /* the & is redundant */
}
```

(364) attempt to modify object qualified * *(Parser)*

Objects declared `const` or `code` may not be assigned to or modified in any other way by your program. The effect of attempting to modify such an object is compiler-specific.

```
const int out = 1234;  /* "out" is read only */
out = 0;               /* oops --
                          writing to a read-only object */
```

(365) pointer to non-static object returned *(Parser)*

This function returns a pointer to a non-`static` (e.g. `auto`) variable. This is likely to be an error, since the storage associated with automatic variables becomes invalid when the function returns, e.g.:

```
char * get_addr(void)
{
  char c;
  /* returning this is dangerous;
     the pointer could be dereferenced */
  return &c;
}
```

(366) operands of "*" not same pointer type *(Parser)*

The operands of this operator are of different pointer types. This probably means you have used the wrong pointer, but if the code is actually what you intended, use a typecast to suppress the error message.

(367) identifier is already extern; can't be static ***(Parser)***

This function was already declared `extern`, possibly through an implicit declaration. It has now been redeclared `static`, but this redeclaration is invalid.

```
void main(void)
{
  /* at this point the compiler assumes set is extern... */
  set(10L, 6);
}
/* now it finds out otherwise */
static void set(long a, int b)
{
  PORTA = a + b;
}
```

(368) array dimension on "*[]" ignored ***(Preprocessor)***

An array dimension on a function parameter has been ignored because the argument is actually converted to a pointer when passed. Thus arrays of any size may be passed. Either remove the dimension from the parameter, or define the parameter using pointer syntax, e.g.:

```
/* param should be: "int array[]" or "int *" */
int get_first(int array[10])
{                               /* warning flagged here */
  return array[0];
}
```

(369) signed bitfields not supported ***(Parser)***

Only `unsigned` bitfields are supported. If a bitfield is declared to be type `int`, the compiler still treats it as `unsigned`, e.g.:

```
struct {
  signed int sign: 1;    /* this must be unsigned */
  signed int value: 15;
} ;
```

(370) illegal basic type; int assumed ***(Parser)***

The basic type of a cast to a qualified basic type couldn't not be recognised and the basic type was assumed to be `int`, e.g.:

```
/* here ling is assumed to be int */
unsigned char bar = (unsigned ling) 'a';
```

(371) missing basic type; int assumed ***(Parser)***

This declaration does not include a basic type, so `int` has been assumed. This declaration is not illegal, but it is preferable to include a basic type to make it clear what is intended, e.g.:

```
char c;
i;      /* don't let the compiler make assumptions, use : int i */
func(); /* ditto, use: extern int func(int); */
```

(372) "," expected ***(Parser)***

A *comma* was expected here. This could mean you have left out the *comma* between two identifiers in a declaration list. It may also mean that the immediately preceding type name is misspelled, and has thus been interpreted as an identifier, e.g.:

```
unsigned char a;
/* thinks: chat & b are unsigned, but where is the comma? */
unsigned chat b;
```

(373) implicit signed to unsigned conversion ***(Parser)***

An `unsigned` type was expected where a `signed` type was given and was implicitly cast to `unsigned`, e.g.:

```
unsigned int foo = -1;
/* the above initialization is implicitly treated as:
   unsigned int foo = (unsigned) -1; */
```

(374) missing basic type; int assumed ***(Parser)***

The basic type of a cast to a qualified basic type was missing and assumed to be `int`., e.g.:

```
int i = (signed) 2; /* (signed) assumed to be (signed int) */
```

(375) unknown FNREC type "*" ***(Linker)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(376) bad non-zero node in call graph ***(Linker)***

The linker has encountered a top level node in the call graph that is referenced from lower down in the call graph. This probably means the program has indirect recursion, which is not allowed when using a compiled stack.

(378) can't create * file "*" ***(Hexmate)***

This type of file could not be created. Is the file or a file by this name already in use?

(379) bad record type "*" ***(Linker)***

This is an internal compiler error. Ensure the object file is a valid HI-TECH object file. Contact HI-TECH Software technical support with details.

(380) unknown record type (*) ***(Linker)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(381) record "*" too long (*) ***(Linker)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(382) incomplete record: type = *, length = * ***(Dump, Xstrip)***

This message is produced by the DUMP or XSTRIP utilities and indicates that the object file is not a valid HI-TECH object file, or that it has been truncated. Contact HI-TECH Support with details.

(383) text record has length (*) too small ***(Linker)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(384) assertion failed: file *, line *, expression * ***(Linker, Parser)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(387) illegal or too many -G options ***(Linker)***

There has been more than one linker `-g` option, or the `-g` option did not have any arguments following. The arguments specify how the segment addresses are calculated.

(388) duplicate -M option ***(Linker)***

The map file name has been specified to the linker for a second time. This should not occur if you are using a compiler driver. If invoking the linker manually, ensure that only one instance of this option is present on the command line. See Section 5.7.9 for information on the correct syntax for this option.

(389) illegal or too many -O options ***(Linker)***

This linker `-o` flag is illegal, or another `-o` option has been encountered. A `-o` option to the linker must be immediately followed by a filename with no intervening space.

(390) missing argument to -P ***(Linker)***

There have been too many `-p` options passed to the linker, or a `-p` option was not followed by any arguments. The arguments of separate `-p` options may be combined and separated by *commas*.

(391) missing argument to -Q ***(Linker)***

The `-Q` linker option requires the machine type for an argument.

(392) missing argument to -U ***(Linker)***

The `-U` (undefine) option needs an argument.

(393) missing argument to -W ***(Linker)***

The `-W` option (listing width) needs a numeric argument.

(394) duplicate -D or -H option ***(Linker)***

The symbol file name has been specified to the linker for a second time. This should not occur if you are using a compiler driver. If invoking the linker manually, ensure that only one instance of either of these options is present on the command line.

(395) missing argument to -J ***(Linker)***

The maximum number of errors before aborting must be specified following the `-j` linker option.

(397) usage: hlink [-options] files.obj files.lib ***(Linker)***

Improper usage of the command-line linker. If you are invoking the linker directly then please refer to Section 5.7 for more details. Otherwise this may be an internal compiler error and you should contact HI-TECH Software technical support with details.

(398) output file can't be also an input file ***(Linker)***

The linker has detected an attempt to write its output file over one of its input files. This cannot be done, because it needs to simultaneously read and write input and output files.

(400) bad object code format ***(Linker)***

This is an internal compiler error. The object code format of an object file is invalid. Ensure it is a valid HI-TECH object file. Contact HI-TECH Software technical support with details.

(402) bad argument to -F ***(Objtohex)***

The `-F` option for `objtohex` has been supplied an invalid argument. If you are invoking this command-line tool directly then please refer to Section 5.12 for more details. Otherwise this may be an internal compiler error and you should contact HI-TECH Software technical support with details.

(403) bad -E option: "*" ***(Objtohex)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(404) bad maximum length value to -<digits> ***(Objtohex)***

The first value to the OBJTOHEX `-n,m` hex length/rounding option is invalid.

(405) bad record size rounding value to -<digits> ***(Objtohex)***

The second value to the OBJTOHEX `-n,m` hex length/rounding option is invalid.

(406) bad argument to -A ***(Objtohex)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(407) bad argument to -U ***(Objtohex)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(408) bad argument to -B *(Objtohex)*

This option requires an integer argument in either base 8, 10 or 16. If you are invoking `objtohex` directly then see Section 5.12 for more details. Otherwise this may be an internal compiler error and you should contact HI-TECH Software technical support with details.

(409) bad argument to -P *(Objtohex)*

This option requires an integer argument in either base 8, 10 or 16. If you are invoking `objtohex` directly then see Section 5.12 for more details. Otherwise this may be an internal compiler error and you should contact HI-TECH Software technical support with details.

(410) bad combination of options *(Objtohex)*

The combination of options supplied to `OBJTOHEX` is invalid.

(412) text does not start at 0 *(Objtohex)*

Code in some things must start at zero. Here it doesn't.

(413) write error on "*" *(Assembler, Linker, Cromwell)*

A write error occurred on the named file. This probably means you have run out of disk space.

(414) read error on "*" *(Linker)*

The linker encountered an error trying to read this file.

(415) text offset too low in COFF file *(Objtohex)*

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(416) bad character (*) in extended TEKHEX line *(Objtohex)*

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(417) seek error in "*" *(Linker)*

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(418) image too big ***(Objtohex)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(419) object file is not absolute ***(Objtohex)***

The object file passed to `OBJTOHEX` has relocation items in it. This may indicate it is the wrong object file, or that the linker or OBJTOHEX have been given invalid options. The object output files from the assembler are relocatable, not absolute. The object file output of the linker is absolute.

(420) too many relocation items ***(Objtohex)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(421) too many segments ***(Objtohex)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(422) no end record ***(Linker)***

This object file has no end record. This probably means it is not an object file. Contact HI-TECH Support if the object file was generated by the compiler.

(423) illegal record type ***(Linker)***

There is an error in an object file. This is either an invalid object file, or an internal error in the linker. Contact HI-TECH Support with details if the object file was created by the compiler.

(424) record too long ***(Objtohex)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(425) incomplete record ***(Objtohex, Libr)***

The object file passed to OBJTOHEX or the librarian is corrupted. Contact HI-TECH Support with details.

(427) syntax error in checksum list ***(Objtohex)***

There is a syntax error in a checksum list read by OBJTOHEX. The checksum list is read from standard input in response to an option.

(428) too many segment fixups ***(Objtohex)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(429) bad segment fixups ***(Objtohex)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(430) bad checksum specification ***(Objtohex)***

A checksum list supplied to `OBJTOHEX` is syntactically incorrect.

(431) bad argument to -E ***(Objtoexe)***

This option requires an integer argument in either base 8, 10 or 16. If you are invoking `objtoexe` directly then check this argument. Otherwise this may be an internal compiler error and you should contact HI-TECH Software technical support with details.

(432) usage: objtohex [-ssymfile] [object-file [exe-file]] ***(Objtohex)***

Improper usage of the command-line tool `objtohex`. If you are invoking `objtohex` directly then please refer to Section 5.12 for more details. Otherwise this may be an internal compiler error and you should contact HI-TECH Software technical support with details.

(434) too many symbols (*) ***(Linker)***

There are too many symbols in the symbol table, which has a limit of * symbols. Change some global symbols to local symbols to reduce the number of symbols.

(435) bad segment selector "*" ***(Linker)***

The segment specification option (`-G`) to the linker is invalid, e.g.:

```
-GA/f0+10
```

Did you forget the radix?

```
-GA/f0h+10
```

(436) psect "*" re-orged ***(Linker)***

This psect has had its start address specified more than once.

(437) missing "=" in class spec ***(Linker)***

A class spec needs an = sign, e.g. -Ctext=ROM See Section 5.7.9 for more information.

(438) bad size in -S option ***(Linker)***

The address given in a `-S` specification is invalid: it should be a valid number, in decimal, octal or hexadecimal radix. The radix is specified by a trailing `O`, for octal, or `H` for hex. A leading `0x` may also be used for hexadecimal. Case in not important for any number or radix. Decimal is the default, e.g.:

```
-SCODE=f000
```

Did you forget the radix?

```
-SCODE=f000h
```

(439) bad -D spec: "*" ***(Linker)***

The format of a `-D` specification, giving a *delta* value to a class, is invalid, e.g.:

```
-DCODE
```

What is the *delta* value for this class? Maybe you meant something like:

```
-DCODE=2
```

(440) bad delta value in -D spec ***(Linker)***

The *delta* value supplied to a `-D` specification is invalid. This value should an integer of base 8, 10 or 16.

(441) bad -A spec: "*" ***(Linker)***

The format of a `-A` specification, giving address ranges to the linker, is invalid, e.g.:

```
-ACODE
```

What is the range for this class? Maybe you meant:

```
-ACODE=0h-1fffh
```

(442) missing address in -A spec *(Linker)*

The format of a `-A` specification, giving address ranges to the linker, is invalid, e.g.:

```
-ACODE=
```

What is the range for this class? Maybe you meant:

```
-ACODE=0h-1fffh
```

(443) bad low address "*" in -A spec *(Linker)*

The low address given in a -A specification is invalid: it should be a valid number, in decimal, octal or hexadecimal radix. The radix is specified by a trailing O (for octal) or H for hex. A leading `0x` may also be used for hexadecimal. Case in not important for any number or radix. Decimal is default, e.g.:

```
-ACODE=1fff-3fffh
```

Did you forget the radix?

```
-ACODE=1fffh-3fffh
```

(444) expected "-" in -A spec *(Linker)*

There should be a minus sign, `-`, between the high and low addresses in a `-A` linker option, e.g.

```
-AROM=1000h
```

maybe you meant:

```
-AROM=1000h-1fffh
```

(445) bad high address "*" in -A spec *(Linker)*

The high address given in a `-A` specification is invalid: it should be a valid number, in decimal, octal or hexadecimal radix. The radix is specified by a trailing `O`, for octal, or `H` for hex. A leading `0x` may also be used for hexadecimal. Case in not important for any number or radix. Decimal is the default, e.g.:

```
-ACODE=0h-ffff
```

Did you forget the radix?

```
-ACODE=0h-ffffh
```

See Section 5.7.20 for more information.

(446) bad overrun address "*" in -A spec ***(Linker)***

The overrun address given in a -A specification is invalid: it should be a valid number, in decimal, octal or hexadecimal radix. The radix is specified by a trailing O (for octal) or H for hex. A leading `0x` may also be used for hexadecimal. Case in not important for any number or radix. Decimal is default, e.g.:

```
-AENTRY=0-0FFh-1FF
```

Did you forget the radix?

```
-AENTRY=0-0FFh-1FFh
```

(447) bad load address "*" in -A spec ***(Linker)***

The load address given in a `-A` specification is invalid: it should be a valid number, in decimal, octal or hexadecimal radix. The radix is specified by a trailing `O` (for octal) or `H` for hex. A leading `0x` may also be used for hexadecimal. Case in not important for any number or radix. Decimal is default, e.g.:

```
-ACODE=0h-3fffh/a000
```

Did you forget the radix?

```
-ACODE=0h-3fffh/a000h
```

(448) bad repeat count "*" in -A spec ***(Linker)***

The repeat count given in a `-A` specification is invalid, e.g.:

```
-AENTRY=0-0FFhxf
```

Did you forget the radix?

```
-AENTRY=0-0FFhxfh
```

(449) syntax error in -A spec: * ***(Linker)***

The `-A` spec is invalid. A valid -A spec should be something like:
-AROM=1000h-1FFFh

(450) psect "*" was never defined ***(Linker, Optimiser)***

This psect has been listed in a `-P` option, but is not defined in any module within the program.

(451) bad psect origin format in -P option ***(Linker)***

The origin format in a `-p` option is not a validly formed decimal, octal or hex number, nor is it the name of an existing psect. A hex number must have a trailing H, e.g.:

```
-pbss=f000
```

Did you forget the radix?

```
-pbss=f000h
```

(452) bad "+" (minimum address) format in -P option ***(Linker)***

The minimum address specification in the linker's `-p` option is badly formatted, e.g.:

```
-pbss=data+f000
```

Did you forget the radix?

```
-pbss=data+f000h
```

(453) missing number after "%" in -P option ***(Linker)***

The `%` operator in a `-p` option (for rounding boundaries) must have a number after it.

(454) link and load address can't both be set to "." in -P option ***(Linker)***

The link and load address of a psect have both been specified with a *dot* character. Only one of these addresses may be specified in this manner, e.g.:

```
-Pmypsect=1000h/.
-Pmypsect=./1000h
```

Both of these options are valid and equivalent, however the following usage is ambiguous:

```
-Pmypsect=./.
```

What is the link or load address of this psect?

(455) psect "*" not relocated on 0x* byte boundary ***(Linker)***

This psect is not relocated on the required boundary. Check the relocatability of the psect and correct the `-p` option. if necessary.

(456) psect "*" not loaded on 0x* boundary ***(Linker)***

This psect has a relocatability requirement that is not met by the load address given in a `-p` option. For example if a psect must be on a 4K byte boundary, you could not start it at 100H.

(459) remove failed, error: *, * ***(xstrip)***

The creation of the output file failed when removing an interemediate file.

(460) rename failed, error: *, * ***(xstrip)***

The creation of the output file failed when renaming an interemediate file.

(461) can't create * file "*" ***(Assembler, Code Generator)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(464) missing key in avmap file ***(Linker)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(465) undefined symbol "*" in FNBREAK record ***(Linker)***

The linker has found an undefined symbol in the `FNBREAK` record for a non-reentrant function. Contact HI-TECH Support if this is not handwritten assembler code.

(466) undefined symbol "*" in FNINDIR record ***(Linker)***

The linker has found an undefined symbol in the `FNINDIR` record for a non-reentrant function. Contact HI-TECH Support if this is not handwritten assembler code.

(467) undefined symbol "*" in FNADDR record ***(Linker)***

The linker has found an undefined symbol in the FNADDR record for a non-reentrant function. Contact HI-TECH Support if this is not handwritten assembler code.

(468) undefined symbol "*" in FNCALL record ***(Linker)***

The linker has found an undefined symbol in the `FNCALL` record for a non-reentrant function. Contact HI-TECH Support if this is not handwritten assembler code.

(469) undefined symbol "*" in FNROOT record ***(Linker)***

The linker has found an undefined symbol in the `FNROOT` record for a non-reentrant function. Contact HI-TECH Support if this is not handwritten assembler code.

(470) undefined symbol "*" in FNSIZE record ***(Linker)***

The linker has found an undefined symbol in the `FNSIZE` record for a non-reentrant function. Contact HI-TECH Support if this is not handwritten assembler code.

(471) recursive function calls: ***(Linker)***

These functions (or function) call each other recursively. One or more of these functions has statically allocated local variables (compiled stack). Either use the `reentrant` keyword (if supported with this compiler) or recode to avoid recursion, e.g.:

```
int test(int a)
{
   if(a == 5) {
      /* recursion may not be supported by some compilers */
      return test(a++);
   }
   return 0;
}
```

(472) non-reentrant function "*" appears in multiple call graphs: rooted at "*" and "*" ***(Linker)***

This function can be called from both main-line code and interrupt code. Use the `reentrant` keyword, if this compiler supports it, or recode to avoid using local variables or parameters, or duplicate the function, e.g.:

```
void interrupt my_isr(void)
{
  scan(6);     /* scan is called from an interrupt function */
}
```

```
void process(int a)
{
  scan(a);      /* scan is also called from main-line code */
}
```

(473) function "*" is not called from specified interrupt_level ***(Linker)***

The indicated function is never called from an interrupt function of the same interrupt level, e.g.:

```
#pragma interrupt_level 1
void foo(void)
{
    ...
}
#pragma interrupt_level 1
void interrupt bar(void)
{
    // this function never calls foo()
}
```

(474) no psect specified for function variable/argument allocation ***(Linker)***

The `FNCONF` assembler directive which specifies to the linker information regarding the auto/parameter block was never seen. This is supplied in the standard runtime files if necessary. This error may imply that the correct run-time startup module was not linked. Ensure you have used the `FNCONF` directive if the runtime startup module is hand-written.

(475) conflicting FNCONF records ***(Linker)***

The linker has seen two conflicting `FNCONF` directives. This directive should only be specified once and is included in the standard runtime startup code which is normally linked into every program.

(476) fixup overflow referencing * * (location 0x* (0x*+*), size *, value 0x*) ***(Linker)***

The linker was asked to relocate (fixup) an item that would not fit back into the space after relocation. See the following error message (477) for more information..

(477) fixup overflow in expression (location 0x* (0x*+*), size *, value 0x*) ***(Linker)***

Fixup is the process conducted by the linker of replacing symbolic references to variables etc, in an assembler instruction with an absolute value. This takes place after positioning the psects (program

sections or blocks) into the available memory on the target device. Fixup overflow is when the value determined for a symbol is too large to fit within the allocated space within the assembler instruction. For example, if an assembler instruction has an 8-bit field to hold an address and the linker determines that the symbol that has been used to represent this address has the value 0x110, then clearly this value cannot be inserted into the instruction.

The causes for this can be many, but hand-written assembler code is always the first suspect. Badly written C code can also generate assembler that ultimately generates fixup overflow errors. Consider the following error message.

```
main.obj: 8: Fixup overflow in expression (loc 0x1FD (0x1FC+1),
    size 1, value 0x7FC)
```

This indicates that the file causing the problem was `main.obj`. This would be typically be the output of compiling `main.c` or `main.as`. This tells you the file in which you should be looking. The next number (8 in this example) is the record number in the object file that was causing the problem. If you use the `DUMP` utility to examine the object file, you can identify the record, however you do not normally need to do this.

The location (`loc`) of the instruction (0x1FD), the `size` (in bytes) of the field in the instruction for the value (1) , and the `value` which is the actual value the symbol represents, is typically the only information needed to track down the cause of this error. Note that a size which is not a multiple of 8 bits will be rounded up to the nearest byte size, i.e. a 7 bit space in an instruction will be shown as 1 byte.

Generate an assembler list file for the appropriate module. Look for the address specified in the error message.

```
7    07FC    0E21  movlw 33
8    07FD    6FFC  movwf _foo
9    07FE    0012  return
```

and to confirm, look for the symbol referenced in the assembler instruction at this address in the symbol table at the bottom of the same file.

```
Symbol Table                    Fri Aug 12 13:17:37 2004
_foo 01FC   _main 07FF
```

In this example, the instruction causing the problem takes an 8-bit offset into a bank of memory, but clearly the address 0x1FC exceeds this size. Maybe the instruction should have been written as:

```
movwf  (_foo&0ffh)
```

which masks out the top bits of the address containing the bank information.

If the assembler instruction that caused this error was generated by the compiler, in the assembler list file look back up the file from the instruction at fault to determine which C statement has generated this instruction. You will then need to examine the C code for possible errors. incorrectly qualified pointers are an common trigger.

(478) * range check failed (location 0x* (0x*+*), value 0x* > limit 0x*) ***(Linker)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(479) circular indirect definition of symbol "*" ***(Linker)***

The specified symbol has been equated to an external symbol which, in turn, has been equated to the first symbol.

(480) function signatures do not match: * (*): 0x*/0x* ***(Linker)***

The specified function has different signatures in different modules. This means it has been declared differently, e.g. it may have been prototyped in one module and not another. Check what declarations for the function are visible in the two modules specified and make sure they are compatible, e.g.:

```
extern int get_value(int in);
/* and in another module: */
/* this is different to the declaration */
int get_value(int in, char type)
{
```

(481) common symbol "*" psect conflict ***(Linker)***

A common symbol has been defined to be in more than one psect.

(482) symbol "*" is defined more than once in "*" ***(Assembler)***

This symbol has been defined in more than one place. The assembler will issue this error if a symbol is defined more than once in the same module, e.g.:

```
_next:
  move r0, #55
  move [r1], r0
_next:           ; oops -- choose a different name
```

The linker will issue this warning if the symbol (C or assembler) was defined multiple times in different modules. The names of the modules are given in the error message. Note that C identifiers often have an *underscore* prepended to their name after compilation.

(483) symbol "*" can't be global ***(Linker)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(484) psect "*" can't be in classes "*" and "*" ***(Linker)***

A psect cannot be in more than one class. This is either due to assembler modules with conflicting `class=` options to the PSECT directive, or use of the `-C` option to the linker, e.g.:

```
psect final,class=CODE
finish:
/* elsewhere: */
psect final,class=ENTRY
```

(485) unknown "with" psect referenced by psect "*" ***(Linker)***

The specified psect has been placed with a psect using the psect `with` flag. The psect it has been placed with does not exist, e.g.:

```
psect starttext,class=CODE,with=rext
    ; was that meant to be with text?
```

(486) psect "*" selector value redefined ***(Linker)***

The selector value for this psect has been defined more than once.

(487) psect "*" type redefined: */* ***(Linker)***

This psect has had its type defined differently by different modules. This probably means you are trying to link incompatible object modules, e.g. linking 386 flat model code with 8086 real mode code.

(488) psect "*" memory space redefined: */* ***(Linker)***

A global psect has been defined in two different memory spaces. Either rename one of the psects or, if they are the same psect, place them in the same memory space using the `space` psect flag, e.g.:

```
psect spdata,class=RAM,space=0
  ds 6
; elsewhere:
psect spdata,class=RAM,space=1
```

(489) psect "*" memory delta redefined: */* ***(Linker)***

A global psect has been defined with two different delta values, e.g.:

```
psect final,class=CODE,delta=2
finish:
; elsewhere:
psect final,class=CODE,delta=1
```

(490) class "*" memory space redefined: */* ***(Linker)***

A class has been defined in two different memory spaces. Either rename one of the classes or, if they are the same class, place them in the same memory space.

(491) can't find 0x* words for psect "*" in segment "*" ***(Linker)***

One of the main tasks the linker performs is positioning the blocks (or psects) of code and data that is generated from the program into the memory available for the target device. This error indicates that the linker was unable to find an area of free memory large enough to accommodate one of the psects. The error message indicates the name of the psect that the linker was attempting to position and the segment name which is typically the name of a class which is defined with a linker `-A` option.

Section 3.8.1 lists each compiler-generated psect and what it contains. Typically psect names which are, or include, `text` relate to program code. Names such as `bss` or `data` refer to variable blocks. This error can be due to two reasons.

First, the size of the program or the program's data has exceeded the total amount of space on the selected device. In other words, some part of your device's memory has completely filled. If this is the case, then the size of the specified psect must be reduced.

The second cause of this message is when the total amount of memory needed by the psect being positioned is sufficient, but that this memory is fragmented in such a way that the largest contiguous block is too small to accommodate the psect. The linker is unable to split psects in this situation. That is, the linker cannot place part of a psect at one location and part somewhere else. Thus, the linker must be able to find a contiguous block of memory large enough for every psect. If this is the cause of the error, then the psect must be split into smaller psects if possible.

To find out what memory is still available, generate and look in the map file, see Section 2.6.8 for information on how to generate a map file. Search for the string `UNUSED ADDRESS RANGES`.

Under this heading, look for the name of the segment specified in the error message. If the name is not present, then all the memory available for this psect has been allocated. If it is present, there will be one address range specified under this segment for each free block of memory. Determine the size of each block and compare this with the number of words specified in the error message.

Psects containing code can be reduced by using all the compiler's optimizations, or restructuring the program. If a code psect must be split into two or more small psects, this requires splitting a function into two or more smaller functions (which may call each other). These functions may need to be placed in new modules.

Psects containing data may be reduced when invoking the compiler optimizations, but the effect is less dramatic. The program may need to be rewritten so that it needs less variables. Section 5.10.2.2 has information on interpreting the map file's call graph if the compiler you are using uses a compiled stack. (If the string `Call graph:` is not present in the map file, then the compiled code uses a hardware stack.) If a data psect needs to be split into smaller psects, the definitions for variables will need to be moved to new modules or more evenly spread in the existing modules. Memory allocation for `auto` variables is entirely handled by the compiler. Other than reducing the number of these variables used, the programmer has little control over their operation. This applies whether the compiled code uses a hardware or compiled stack.

For example, after receiving the message:

```
Can't find 0x34 words (0x34 withtotal) for psect text
 in segment CODE (error)
```

look in the map file for the ranges of unused memory.

```
UNUSED ADDRESS RANGES
        CODE             00000244-0000025F
                         00001000-0000102f
        RAM              00300014-00301FFB
```

In the `CODE` segment, there is 0x1c (0x25f-0x244+1) bytes of space available in one block and 0x30 available in another block. Neither of these are large enough to accommodate the psect `text` which is 0x34 bytes long. Notice, however, that the total amount of memory available is larger than 0x34 bytes.

(492) attempt to position absolute psect "*" is illegal *(Linker)*

This psect is absolute and should not have an address specified in a `-P` option. Either remove the `abs` psect flag, or remove the `-P` linker option.

(493) origin of psect "*" is defined more than once ***(Linker)***

The origin of this psect is defined more than once. There is most likely more than one `-p` linker option specifying this psect.

(494) bad -P format "*/*" ***(Linker)***

The `-P` option given to the linker is malformed. This option specifies placement of a psect, e.g.:

```
-Ptext=10g0h
```

Maybe you meant:

```
-Ptext=10f0h
```

(495) use of both "with=" and "INCLASS/INCLASS" allocation is illegal ***(Linker)***

It is not legal to specify both the link and location of a psect as within a class, when that psect was also defined using a `with` psect flag.

(497) psect "*" exceeds max size: *h > *h ***(Linker)***

The psect has more bytes in it than the maximum allowed as specified using the `size` psect flag.

(498) psect "*" exceeds address limit: *h > *h ***(Linker)***

The maximum address of the psect exceeds the limit placed on it using the `limit` psect flag. Either the psect needs to be linked at a different location or there is too much code/data in the psect.

(499) undefined symbol: ***(Assembler, Linker)***

The symbol following is undefined at link time. This could be due to spelling error, or failure to link an appropriate module.

(500) undefined symbols: ***(Linker)***

A list of symbols follows that were undefined at link time. These errors could be due to spelling error, or failure to link an appropriate module.

(501) program entry point is defined more than once ***(Linker)***

There is more than one entry point defined in the object files given the linker. End entry point is specified after the `END` directive. The runtime startup code defines the entry point, e.g.:

```
powerup:
  goto start
  END  powerup  ; end of file and define entry point
; other files that use END should not define another entry point
```

(502) incomplete * record body: length = * ***(Linker)***

An object file contained a record with an illegal size. This probably means the file is truncated or not an object file. Contact HI-TECH Support with details.

(503) ident records do not match ***(Linker)***

The object files passed to the linker do not have matching ident records. This means they are for different processor types.

(504) object code version is greater than *.* ***(Linker)***

The object code version of an object module is higher than the highest version the linker is known to work with. Check that you are using the correct linker. Contact HI-TECH Support if the object file if you have not patched the linker.

(505) no end record found inobject file ***(Linker)***

An object file did not contain an end record. This probably means the file is corrupted or not an object file. Contact HI-TECH Support if the object file was generated by the compiler.

(506) object file record too long: *+* ***(Linker)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(507) unexpected end of file in object file ***(Linker)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(508) relocation offset (*) out of range 0..*-*-1 ***(Linker)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(509) illegal relocation size: * ***(Linker)***

There is an error in the object code format read by the linker. This either means you are using a linker that is out of date, or that there is an internal error in the assembler or linker. Contact HI-TECH Support with details if the object file was created by the compiler.

(510) complex relocation not supported for -R or -L options ***(Linker)***

The linker was given a `-R` or `-L` option with file that contain complex relocation.

(511) bad complex range check ***(Linker)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(512) unknown complex operator 0x* ***(Linker)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(513) bad complex relocation ***(Linker)***

The linker has been asked to perform complex relocation that is not syntactically correct. Probably means an object file is corrupted.

(514) illegal relocation type: * ***(Linker)***

An object file contained a relocation record with an illegal relocation type. This probably means the file is corrupted or not an object file. Contact HI-TECH Support with details if the object file was created by the compiler.

(515) unknown symbol type * ***(Linker)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(516) text record has bad length: *-*-(*+1) < 0 ***(Linker)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(520) function "*" is never called ***(Linker)***

This function is never called. This may not represent a problem, but space could be saved by removing it. If you believe this function should be called, check your source code. Some assembler library routines are never called, although they are actually execute. In this case, the routines are linked in a special sequence so that program execution falls through from one routine to the next.

(521) call depth exceeded by function "*" ***(Linker)***

The call graph shows that functions are nested to a depth greater than specified.

(522) library "*" is badly ordered ***(Linker)***

This library is badly ordered. It will still link correctly, but it will link faster if better ordered.

(523) argument to -W option (*) illegal and ignored ***(Linker)***

The argument to the linker option `-w` is out of range. This option controls two features. For warning levels, the range is -9 to 9. For the map file width, the range is greater than or equal to 10.

(524) unable to open list file "*": * ***(Linker)***

The named list file could not be opened. The linker would be trying to fixup the list file so that it will contain absolute addresses. Ensure that an assembler list file was generated during the compilation stage. Alternatively, remove the assembler list file generation option from the link step.

(525) too many address (memory) spaces; space (*) ignored ***(Linker)***

The limit to the number of address spaces (specified with the `PSECT` assembler directive) is currently 16.

(526) psect "*" not specified in -P option (first appears in "*") ***(Linker)***

This psect was not specified in a `-P` or `-A` option to the linker. It has been linked at the end of the program, which is probably not where you wanted it.

(528) no start record; entry point defaults to zero ***(Linker)***

None of the object files passed to the linker contained a start record. The start address of the program has been set to zero. This may be harmless, but it is recommended that you define a start address in your startup module by using the `END` directive.

(529) usage: objtohex [-Ssymfile] [object-file [hex-file]] ***(Objtohex)***

Improper usage of the command-line tool `objtohex`. If you are invoking `objtohex` directly then please refer to Section 5.12 for more details. Otherwise this may be an internal compiler error and you should contact HI-TECH Software technical support with details.

(593) can't find 0x* words (0x* withtotal) for psect "*" in segment "*" ***(Linker)***

See error (491) on Page 418.

(594) undefined symbol: ***(Linker)***

The symbol following is undefined at link time. This could be due to spelling error, or failure to link an appropriate module.

(595) undefined symbols: ***(Linker)***

A list of symbols follows that were undefined at link time. These errors could be due to spelling error, or failure to link an appropriate module.

(596) segment "*" (*-*) overlaps segment "*" (*-*) ***(Linker)***

The named segments have overlapping code or data. Check the addresses being assigned by the `-P` linker option.

(599) No psect classes given for COFF write ***(Cromwell)***

Cromwell requires that the program memory psect classes be specified to produce a COFF file. Ensure that you are using the `-N` option as per Section 5.14.2.

(600) No chip arch given for COFF write ***(Cromwell)***

Cromwell requires that the chip architecture be specified to produce a COFF file. Ensure that you are using the -P option as per Section 5.14.1.

(601) Unknown chip arch "*" for COFF write ***(Cromwell)***

The chip architecture specified for producing a COFF file isn't recognised by Cromwell. Ensure that you are using the -P option as per Section 5.14.1 and that the architecture specified matches one of those in Table 5.8.

(602) null file format name ***(Cromwell)***

The `-I` or `-O` option to Cromwell must specify a file format.

(603) ambiguous file format name "*" ***(Cromwell)***

The input or output format specified to Cromwell is ambiguous. These formats are specified with the `-i`*`key`* and `-o`*`key`* options respectively.

(604) unknown file format name "*" ***(Cromwell)***

The output format specified to `CROMWELL` is unknown, e.g.:

```
cromwell -m -P16F877 main.hex main.sym -ocot
```

and output file type of `cot`, did you mean `cof`?

(605) did not recognize format of input file ***(Cromwell)***

The input file to Cromwell is required to be COD, Intel HEX, Motorola HEX, COFF, OMF51, P&E or HI-TECH.

(606) inconsistent symbol tables ***(Cromwell)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(607) inconsistent line number tables ***(Cromwell)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(608) bad path specification ***(Cromwell)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(609) missing processor spec after -P ***(Cromwell)***

The `-p` option to cromwell must specify a processor name.

(610) missing psect classes after -N ***(Cromwell)***

Cromwell requires that the `-N` option be given a list of the names of psect classes.

(611) too many input files ***(Cromwell)***

To many input files have been specified to be converted by `CROMWELL`.

(612) too many output files ***(Cromwell)***

To many output file formats have been specified to `CROMWELL`.

(613) no output file format specified ***(Cromwell)***

The output format must be specified to CROMWELL.

(614) no input files specified ***(Cromwell)***

CROMWELL must have an input file to convert.

(616) option -Cbaseaddr is illegal with options -R or -L ***(Linker)***

The linker option `-Cbaseaddr` cannot be used in conjunction with either the `-R` or `-L` linker options.

(618) error reading COD file data ***(Cromwell)***

An error occurred reading the input COD file. Confirm the spelling and path of the file specified on the command line.

(619) I/O error reading symbol table ***(Cromwell)***

The COD file has an invalid format in the specified record.

(620) filename index out of range in line number record ***(Cromwell)***

The COD file has an invalid value in the specified record.

(621) error writing ELF/DWARF section "*" on "*" ***(Cromwell)***

An error occurred writing the indicated section to the given file. Confirm the spelling and path of the file specified on the command line.

(622) too many type entries ***(Cromwell)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(623) bad class in type hashing ***(Cromwell)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(624) bad class in type compare ***(Cromwell)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(625) too many files in COFF file ***(Cromwell)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(626) string lookup failed in COFF: get_string() ***(Cromwell)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(627) missing "*" in SDB file "*" line * column * ***(Cromwell)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(629) bad storage class "*" in SDB file "*" line * column * ***(Cromwell)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(630) invalid syntax for prefix list in SDB file "*" ***(Cromwell)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(631) syntax error at token "*" in SDB file "*" line * column * ***(Cromwell)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(632) can't handle address size (*) ***(Cromwell)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(633) unknown symbol class (*) ***(Cromwell)***

Cromwell has encountered a symbol class in the symbol table of a COFF, Microchip COFF, or ICOFF file which it can't identify.

(634) error dumping "*" *(Cromwell)*

Either the input file to CROMWELL is of an unsupported type or that file cannot be dumped to the screen.

(635) invalid HEX file "*" on line * *(Cromwell)*

The specified HEX file contains an invalid line. Contact HI-TECH Support if the HEX file was generated by the compiler.

(636) checksum error in Intel HEX file "*" on line * *(Cromwell, Hexmate)*

A checksum error was found at the specified line in the specified Intel hex file. The HEX file may be corrupt.

(637) unknown prefix "*" in SDB file "*" *(Cromwell)*

This is an internal compiler warning. Contact HI-TECH Software technical support with details.

(638) version mismatch: 0x* expected *(Cromwell)*

The input Microchip COFF file wasn't produced using Cromwell.

(639) zero bit width in Microchip optional header *(Cromwell)*

The optional header in the input Microchip COFF file indicates that the program or data memory spaces are zero bits wide.

(668) prefix list did not match any SDB types *(Cromwell)*

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(669) prefix list matched more than one SDB type *(Cromwell)*

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(670) bad argument to -T *(Clist)*

The argument to the -T option to specify tab size was not present or correctly formed. The option expects a decimal interger argument.

(671) argument to -T should be in range 1 to 64 ***(Clist)***

The argument to the `-T` option to specify tab size was not in the expected range. The option expects a decimal interger argument ranging from 1 to 64 inclusive.

(673) missing filename after * option ***(Objtohex)***

The indicated option requires a valid file name. Ensure that the filename argument supplied to this option exists and is spelt correctly.

(674) too many references to "*" ***(Cref)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(677) set_fact_bit on pic17! ***(Code Generator)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(678) case 55 on pic17! ***(Code Generator)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(679) unknown extraspecial: * ***(Code Generator)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(680) bad format for -P option ***(Code Generator)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(681) bad common spec in -P option ***(Code Generator)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(682) this architecture is not supported by the PICC Lite compiler ***(Code Generator)***

A target device other than baseline, midrange or highend was specified. This compiler only supports devices from these architecture families.

(683) bank 1 variables are not supported by the PICC Lite compiler *(Code Generator)*

A variable with an absolute address located in bank 1 was detected. This compiler does not support code generation of variables in this bank.

(684) bank 2 and 3 variables are not supported by the PICC Lite compiler *(Code Generator)*

A variable with an absolute address located in bank 2 or 3 was detected. This compiler does not support code generation of variables in these banks.

(685) bad putwsize() *(Code Generator)*

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(686) bad switch size (*) *(Code Generator)*

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(687) bad pushreg "*" *(Code Generator)*

This is an internal compiler error. Contact HI-TECH Software technical support with details. See Section 5.7.2 for more information.

(688) bad popreg "*" *(Code Generator)*

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(689) unknown predicate "*" *(Code Generator)*

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(690) interrupt function requires address *(Code Generator)*

The Highend PIC devices support multiple interrupts. An `@ address` is required with the interrupt definition to indicate with which vector this routine is associated, e.g.:

```
void interrupt isr(void) @ 0x10
{
  /* isr code goes here */
}
```

This construct is not required for midrange PIC devices.

(691) interrupt functions not implemented for 12 bit PIC ***(Code Generator)***

The 12-bit range of PIC processors do not support interrupts.

(692) interrupt function "*" may only have one interrupt level ***(Code Generator)***

Only one interrupt level may be associated with an `interrupt` function. Check to ensure that only one `interrupt_level` pragma has been used with the function specified. This pragma may be used more than once on main-line functions that are called from `interrupt` functions. For example:

```
#pragma interrupt_level 0
#pragma interrupt_level 1  /* which is it to be: 0 or 1? */
void interrupt isr(void)
{
```

(693) interrupt level may only be 0 (default) or 1 ***(Code Generator)***

The only possible interrupt levels are 0 or 1. Check to ensure that all `interrupt_level` pragmas use these levels.

```
#pragma interrupt_level 2  /* oops -- only 0 or 1 */
void interrupt isr(void)
{
  /* isr code goes here */
}
```

(694) no interrupt strategy available ***(Code Generator)***

The processor does not support saving and subsequent restoring of registers during an interrupt service routine.

(695) duplicate case label (*) ***(Code Generator)***

There are two case labels with the same value in this `switch` statement, e.g.:

```
switch(in) {
case '0':  /* if this is case '0'... */
  b++;
  break;
case '0':  /* then what is this case? */
```

```
  b--;
  break;
}
```

(696) out-of-range case label (*) ***(Code Generator)***

This case label is not a value that the controlling expression can yield, and thus this label will never be selected.

(697) non-constant case label ***(Code Generator)***

A `case` label in this `switch` statement has a value which is not a constant.

(698) bit variables must be global or static ***(Code Generator)***

A `bit` variable cannot be of type `auto`. If you require a `bit` variable with scope local to a block of code or function, qualify it `static`, e.g.:

```
bit proc(int a)
{
  bit bb;         /* oops --  this should be: static bit bb; */
  bb = (a > 66);
  return bb;
}
```

(699) no case labels in switch ***(Code Generator)***

There are no `case` labels in this `switch` statement, e.g.:

```
switch(input) {
}               /* there is nothing to match the value of input */
```

(700) truncation of enumerated value ***(Code Generator)***

An enumerated value larger than the maximum value supported by this compiler was detected and has been truncated, e.g.:

```
enum { ZERO, ONE, BIG=0x99999999 } test_case;
```

(701) unreasonable matching depth ***(Code Generator)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(702) regused(): bad arg to G *(Code Generator)*

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(703) bad GN *(Code Generator)*

This is an internal compiler error. Contact HI-TECH Software technical support with details. See Section 5.7.2 for more information.

(704) bad RET_MASK *(Code Generator)*

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(705) bad which (*) after I *(Code Generator)*

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(706) bad which in expand() *(Code Generator)*

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(707) bad SX *(Code Generator)*

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(708) bad mod "+" for how = "*" *(Code Generator)*

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(709) metaregister "*" can't be used directly *(Code Generator)*

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(710) bad U usage *(Code Generator)*

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(711) bad how in expand() *(Code Generator)*

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(712) can't generate code for this expression *(Code Generator)*

This error indicates that a C expression is too difficult for the code generator to actually compile. For successful code generation, the code generator must know how to compile an expression and there must be enough resources (e.g. registers or temporary memory locations) available. Simplifying the expression, e.g. using a temporary variable to hold an intermediate result, may get around this message. Contact HI-TECH Support with details of this message.

This error may also be issued if the code being compiled is in some way unusual. For example code which writes to a const-qualified object is illegal and will result in warning messages, but the code generator may unsuccessfully try to produce code to perform the write.

(713) bad initialization list *(Code Generator)*

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(714) bad intermediate code *(Code Generator)*

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(715) bad pragma "*" *(Code Generator)*

The code generator has been passed a `pragma` directive that it does not understand. This implies that the pragma you have used is a HI-TECH specific pragma, but the specific compiler you are using has not implemented this pragma.

(716) bad argument to -M option "*" *(Code Generator)*

The code generator has been passed a `-M` option that it does not understand. This should not happen if it is being invoked by a standard compiler driver.

(718) incompatible intermediate code version; should be *.* *(Code Generator)*

The intermediate code file produced by P1 is not the correct version for use with this code generator. This is either that incompatible versions of one or more compilers have been installed in the same directory, or a temporary file error has occurred leading to corruption of a temporary file. Check the setting of the TEMP environment variable. If it refers to a long path name, change it to something shorter. Contact HI-TECH Support with details if required.

(720) multiple free: * *(Code Generator)*

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(721) element count must be constant expression ***(Code Generator)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(722) bad variable syntax in intermediate code ***(Code Generator)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(723) function definitions nested too deep ***(Code Generator)***

This error is unlikely to happen with C code, since C cannot have nested functions! Contact HI-TECH Support with details.

(724) bad op (*) in revlog() ***(Code Generator)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(726) bad op "*" in uconval() ***(Code Generator)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(727) bad op "*" in bconfloat() ***(Code Generator)***

This is an internal code generator error. Contact HI-TECH technical support with details.

(728) bad op "*" in confloat() ***(Code Generator)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(729) bad op "*" in conval() ***(Code Generator)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(730) bad op "*" ***(Code Generator)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(731) expression error with reserved word ***(Code Generator)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(732) initialization of bit types is illegal *(Code Generator)*

Variables of type `bit` cannot be initialised, e.g.:

```
bit b1 = 1; /* oops!
               b1 must be assigned after its definition */
```

(733) bad string "*" in pragma "psect" *(Code Generator)*

The code generator has been passed a `pragma psect` directive that has a badly formed string, e.g.:

```
#pragma psect text  /* redirect text psect into what? */
```

Maybe you meant something like:

```
#pragma psect text=special_text
```

(734) too many "psect" pragmas *(Code Generator)*

Too many `#pragma psect` directives have been used.

(735) bad string "*" in pragma "stack_size" *(Code Generator)*

The argument to the stack_size pragma is malformed. This pragma must be followed by a number representing the maximum allowed stack size.

(737) unknown argument "*" to pragma "switch" *(Code Generator)*

The `#pragma switch` directive has been used with an invalid switch code generation method. Possible arguments are: `auto`, `simple` and `direct`.

(739) error closing output file *(Code Generator, Optimiser)*

The compiler detected an error when closing a file. Contact HI-TECH Support with details.

(740) zero dimension array is illegal *(Code Generator)*

The code generator has been passed a declaration that results in an array having a zero dimension.

(741) bitfield too large (* bits) ***(Code Generator)***

The maximum number of bits in a bit field is the same as the number of bits in an `int`, e.g. assuming an `int` is 16 bits wide:

```
struct {
  unsigned flag  : 1;
  unsigned value : 12;
  unsigned cont  : 6;   /* oops -- that's a total of 19 bits */
} object;
```

(742) function "*" argument evaluation overlapped ***(Linker)***

A function call involves arguments which overlap between two functions. This could occur with a call like:

```
void fn1(void)
{
  fn3( 7, fn2(3), fn2(9));    /* Offending call */
}
char fn2(char fred)
{
  return fred + fn3(5,1,0);
}
char fn3(char one, char two, char three)
{
  return one+two+three;
}
```

where `fn1` is calling `fn3`, and two arguments are evaluated by calling `fn2`, which in turn calls `fn3`. The program structure should be modified to prevent this type of call sequence.

(743) divide by zero ***(Code Generator)***

An expression involving a division by zero has been detected in your code.

(744) static object "*" has zero size ***(Code Generator)***

A static object has been declared, but has a size of zero.

(745) nodecount = * *(Code Generator)*

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(746) object "*" qualified const, but not initialized *(Code Generator)*

An object has been qualified as `const`, but there is no initial value supplied at the definition. As this object cannot be written by the C program, this may imply the intial value was accidently omitted.

(747) unrecognized option "*" to -Z *(Code Generator)*

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(748) variable "*" may be used before set *(Code Generator)*

This variable may be used before it has been assigned a value. Since it is an `auto` variable, this will result in it having a random value, e.g.:

```
void main(void)
{
  int a;
  if(a)          /* oops -- a has never been assigned a value */
    process();
}
```

(749) unknown register name "*" used with pragma *(Linker)*

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(750) constant operand to || or && *(Code Generator)*

One operand to the logical operators `||` or `&&` is a constant. Check the expression for missing or badly placed parentheses. This message may also occur if the global optimizer is enabled and one of the operands is an `auto` or `static` local variable whose value has been tracked by the code generator, e.g.:

```
{
int a;
a = 6;
if(a || b)  /* a is 6, therefore this is always true */
    b++;
```

(751) arithmetic overflow in constant expression ***(Code Generator)***

A constant expression has been evaluated by the code generator that has resulted in a value that is too big for the type of the expression. The most common code to trigger this warning is assignments to `signed` data types. For example:

```
signed char c;
c = 0xFF;
```

As a `signed` 8-bit quantity, `c` can only be assigned values -128 to 127. The constant is equal to 255 and is outside this range. If you mean to set all bits in this variable, then use either of:

```
c = ~0x0;
c = -1;
```

which will set all the bits in the variable regardless of the size of the variable and without warning.

This warning can also be triggered by intermediate values overflowing. For example:

```
unsigned int i;  /* assume ints are 16 bits wide */
i = 240 * 137;   /* this should be okay, right? */
```

A quick check with your calculator reveals that 240 * 137 is 32880 which can easily be stored in an `unsigned int`, but a warning is produced. Why? Because 240 and 137 and both `signed int` values. Therefore the result of the multiplication must also be a `signed int` value, but a `signed int` cannot hold the value 32880. (Both operands are constant values so the code generator can evaluate this expression at compile time, but it must do so following all the ANSI rules.) The following code forces the multiplication to be performed with an `unsigned` result:

```
i = 240u * 137;  /* force at least one operand
                    to be unsigned */
```

(752) conversion to shorter data type ***(Code Generator)***

Truncation may occur in this expression as the lvalue is of shorter type than the rvalue, e.g.:

```
char a;
int b, c;
a = b + c;  /* int to char conversion
               may result in truncation */
```

(753) undefined shift (* bits) ***(Code Generator)***

An attempt has been made to shift a value by a number of bits equal to or greater than the number of bits in the data type. This will produce an undefined result on many processors. This is non-portable code and is flagged as having undefined results by the C Standard, e.g.:

```
int input;
input <<= 33;  /* oops -- that shifts the entire value out */
```

(754) bitfield comparison out of range ***(Code Generator)***

This is the result of comparing a bitfield with a value when the value is out of range of the bitfield. For example, comparing a 2-bit bitfield to the value 5 will never be true as a 2-bit bitfield has a range from 0 to 3, e.g.:

```
struct {
  unsigned mask : 2;  /* mask can hold values 0 to 3 */
} value;
int compare(void)
{
  return (value.mask == 6);  /* test can
}
```

(755) divide by zero ***(Code Generator)***

A constant expression that was being evaluated involved a division by zero, e.g.:

```
a /= 0;  /* divide by 0: was this what you were intending */
```

(757) constant conditional branch ***(Code Generator)***

A conditional branch (generated by an `if`, `for`, `while` statement etc.) always follows the same path. This will be some sort of comparison involving a variable and a constant expression. For the code generator to issue this message, the variable must have local scope (either `auto` or `static` local) and the global optimizer must be enabled, possibly at higher level than 1, and the warning level threshold may need to be lower than the default level of 0.

The global optimizer keeps track of the contents of local variables for as long as is possible during a function. For C code that compares these variables to constants, the result of the comparison can be deduced at compile time and the output code hard coded to avoid the comparison, e.g.:

```
{
  int a, b;
  a = 5;
  /* this can never be false;
     always perform the true statement */
  if(a == 4)
    b = 6;
```

will produce code that sets `a` to 5, then immediately sets `b` to 6. No code will be produced for the comparison `if(a == 4)`. If a was a global variable, it may be that other functions (particularly interrupt functions) may modify it and so tracking the variable cannot be performed.

This warning may indicate more than an optimization made by the compiler. It may indicate an expression with missing or badly placed parentheses, causing the evaluation to yield a value different to what you expected.

This warning may also be issued because you have written something like `while(1)`. To produce an infinite loop, use `for(;;)`.

A similar situation arises with for loops, e.g.:

```
{
  int a, b;
  /* this loop must iterate at least once */
  for(a=0; a!=10; a++)
    b = func(a);
```

In this case the code generator can again pick up that `a` is assigned the value 0, then immediately checked to see if it is equal to 10. Because `a` is modified during the `for` loop, the comparison code cannot be removed, but the code generator will adjust the code so that the comparison is not performed on the first pass of the loop; only on the subsequent passes. This may not reduce code size, but it will speed program execution.

(758) constant conditional branch: possible use of "=" instead of "==" *(Code Generator)*

There is an expression inside an `if` or other conditional construct, where a constant is being assigned to a variable. This may mean you have inadvertently used an assignment = instead of a compare ==, e.g.:

```
int a, b;
/* this can never be false;
   always perform the true statement */
if(a = 4)
  b = 6;
```

will assign the value 4 to a, then , as the value of the assignment is always true, the comparison can be omitted and the assignment to `b` always made. Did you mean:

```
/* this can never be false;
   always perform the true statement */
if(a == 4)
    b = 6;
```

which checks to see if a is equal to 4.

(759) expression generates no code *(Code Generator)*

This expression generates no output code. Check for things like leaving off the parentheses in a function call, e.g.:

```
int fred;
fred;     /* this is valid, but has no effect at all */
```

Some devices require that special function register need to be read to clear hardware flags. To accommodate this, in some instances the code generator *does* produce code for a statement which only consists of a variable ID. This may happen for variables which are qualified as `volatile`. Typically the output code will read the variable, but not do anything with the value read.

(760) portion of expression has no effect *(Code Generator)*

Part of this expression has no side effects, and no effect on the value of the expression, e.g.:

```
int a, b, c;
a = b,c;  /* "b" has no effect,
             was that meant to be a comma? */
```

(761) sizeof yields 0 *(Code Generator)*

The code generator has taken the size of an object and found it to be zero. This almost certainly indicates an error in your declaration of a pointer, e.g. you may have declared a pointer to a zero length array. In general, pointers to arrays are of little use. If you require a pointer to an array of objects of unknown length, you only need a pointer to a single object that can then be indexed or incremented.

(762) constant truncated when assigned to bitfield ***(Code Generator)***

A constant value is too large for a bitfield structure member to which it is being assigned, e.g.

```
struct INPUT {
  unsigned a : 3;
  unsigned b : 5;
} input_grp;
input_grp.a = 0x12;
/* 12h cannot fit into a 3-bit wide object */
```

(763) constant left operand to "? :" operator ***(Code Generator)***

The left operand to a conditional operator `?` is constant, thus the result of the tertiary operator `?:` will always be the same, e.g.:

```
a = 8 ? b : c;  /* this is the same as saying a = b; */
```

(764) mismatched comparison ***(Code Generator)***

A comparison is being made between a variable or expression and a constant value which is not in the range of possible values for that expression, e.g.:

```
unsigned char c;
if(c > 300)      /* oops -- how can this be true? */
  close();
```

(765) degenerate unsigned comparison ***(Code Generator)***

There is a comparison of an `unsigned` value with zero, which will always be true or false, e.g.:

```
unsigned char c;
if(c >= 0)
```

will always be true, because an `unsigned` value can never be less than zero.

(766) degenerate signed comparison ***(Code Generator)***

There is a comparison of a `signed` value with the most negative value possible for this type, such that the comparison will always be true or false, e.g.:

```
char c;
if(c >= -128)
```

will always be true, because an 8 bit signed char has a maximum negative value of -128.

(767) constant truncated to bitfield width *(Code Generator)*

A constant value is too large for a bitfield structure member on which it is operating, e.g.

```
struct INPUT {
  unsigned a : 3;
  unsigned b : 5;
} input_grp;
input_grp.a |= 0x13;
/* 13h to large for 3-bit wide object */
```

(768) constant relational expression *(Code Generator)*

There is a relational expression that will always be true or false. This may be because e.g. you are comparing an `unsigned` number with a negative value, or comparing a variable with a value greater than the largest number it can represent, e.g.:

```
unsigned int a;
if(a == -10)    /* if a is unsigned, how can it be -10? */
  b = 9;
```

(769) no space for macro definition *(Assembler)*

The assembler has run out of memory.

(772) include files nested too deep *(Assembler)*

Macro expansions and include file handling have filled up the assembler's internal stack. The maximum number of open macros and include files is 30.

(773) macro expansions nested too deep *(Assembler)*

Macro expansions in the assembler are nested too deep. The limit is 30 macros and include files nested at one time.

(774) too many macro parameters *(Assembler)*

There are too many macro parameters on this macro definition.

(776) can't allocate space for object "*" (offs: *) *(Assembler)*

The assembler has run out of memory.

(777) can't allocate space for opnd structure within object "*", (offs: *) ***(Assembler)***

The assembler has run out of memory.

(780) too many psects defined ***(Assembler)***

There are too many psects defined! Boy, what a program!

(781) can't enter abs psect ***(Assembler)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(782) REMSYM error ***(Assembler)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(783) "with" psects are cyclic ***(Assembler)***

If Psect A is to be placed "with" Psect B, and Psect B is to be placed "with" Psect A, there is no hierarchy. The `with` flag is an attribute of a psect and indicates that this psect must be placed in the same memory page as the specified psect.

Remove a `with` flag from one of the psect declarations. Such an assembler declaration may look like:

```
psect my_text,local,class=CODE,with=basecode
```

which will define a psect called `my_text` and place this in the same page as the psect `basecode`.

(784) overfreed ***(Assembler)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(785) too many temporary labels ***(Assembler)***

There are too many temporary labels in this assembler file. The assembler allows a maximum of 2000 temporary labels.

(787) can't handle "v_rtype" of * in copyexpr ***(Assembler)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(788) invalid character "*" in number *(Assembler)*

A number contained a character that was not part of the range 0-9 or 0-F.

(790) end of file inside conditional *(Assembler)*

END-of-FILE was encountered while scanning for an "endif" to match a previous "if".

(793) unterminated macro argument *(Assembler)*

An argument to a macro is not terminated. Note that angle brackets ("< >") are used to quote macro arguments.

(794) invalid number syntax *(Assembler, Optimiser)*

The syntax of a number is invalid. This can be, e.g. use of 8 or 9 in an octal number, or other malformed numbers.

(796) use of LOCAL outside macros is illegal *(Assembler)*

The `LOCAL` directive is only legal inside macros. It defines local labels that will be unique for each invocation of the macro.

(797) syntax error in LOCAL argument *(Assembler)*

A symbol defined using the `LOCAL` assembler directive in an assembler macro is syntactically incorrect. Ensure that all symbols and all other assembler identifiers conform with the assembly language of the target device.

(798) macro argument may not appear after LOCAL *(Assembler)*

The list of labels after the directive `LOCAL` may not include any of the formal parameters to the macro, e.g.:

```
mmm macro a1
  move  r0, #a1
  LOCAL a1  ; oops --
            ; the macro parameter cannot be used with local
ENDM
```

(799) REPT argument must be >= 0 *(Assembler)*

The argument to a REPT directive must be greater than zero, e.g.:

```
rept -2             ; -2 copies of this code? */
  move r0, [r1]++
endm
```

(800) undefined symbol "*" *(Assembler)*

The named symbol is not defined in this module, and has not been specified `GLOBAL`.

(801) range check too complex *(Assembler)*

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(802) invalid address after END directive *(Assembler)*

The start address of the program which is specified after the assembler `END` directive must be a label in the current file.

(803) undefined temporary label *(Assembler)*

A temporary label has been referenced that is not defined. Note that a temporary label must have a number >= 0.

(804) write error on object file *(Assembler)*

The assembler failed to write to an object file. This may be an internal compiler error. Contact HI-TECH Software technical support with details.

(806) attempted to get an undefined object (*) *(Assembler)*

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(807) attempted to set an undefined object (*) *(Assembler)*

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(808) bad size in add_reloc() *(Assembler)*

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(809) unknown addressing mode (*) *(Assembler, Optimiser)*

An unknown addressing mode was used in the assembly file.

(811) "cnt" too large (*) in display() *(Assembler)*

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(814) processor type not defined *(Assembler)*

The processor must be defined either from the command line (eg. -16c84), via the PROCESSOR assembler directive, or via the LIST assembler directive.

(815) syntax error in chipinfo file at line * *(Assembler)*

The chipinfo file contains non-standard syntax at the specified line.

(816) duplicate ARCH specification in chipinfo file "*" at line * *(Assembler, Driver)*

The chipinfo file has a processor section with multiple ARCH values. Only one ARCH value is allowed. If you have not manually edited the chip info file, contact HI-TECH Support with details.

(817) unknown architecture in chipinfo file at line * *(Assembler, Driver)*

An chip architecture (family) that is unknown was encountered when reading the chip INI file.

(818) duplicate BANKS for "*" in chipinfo file at line * *(Assembler)*

The chipinfo file has a processor section with multiple BANKS values. Only one BANKS value is allowed. If you have not manually edited the chip info file, contact HI-TECH Support with details.

(819) duplicate ZEROREG for "*" in chipinfo file at line * *(Assembler)*

The chipinfo file has a processor section with multiple ZEROREG values. Only one ZEROREG value is allowed. If you have not manually edited the chip info file, contact HI-TECH Support with details.

(820) duplicate SPAREBIT for "*" in chipinfo file at line * *(Assembler)*

The chipinfo file has a processor section with multiple SPAREBIT values. Only one SPAREBIT value is allowed. If you have not manually edited the chip info file, contact HI-TECH Support with details.

(821) duplicate INTSAVE for "*" in chipinfo file at line * ***(Assembler)***

The chipinfo file has a processor section with multiple INTSAVE values. Only one INTSAVE value is allowed. If you have not manually edited the chip info file, contact HI-TECH Support with details.

(822) duplicate ROMSIZE for "*" in chipinfo file at line * ***(Assembler)***

The chipinfo file has a processor section with multiple ROMSIZE values. Only one ROMSIZE value is allowed. If you have not manually edited the chip info file, contact HI-TECH Support with details.

(823) duplicate START for "*" in chipinfo file at line * ***(Assembler)***

The chipinfo file has a processor section with multiple START values. Only one START value is allowed. If you have not manually edited the chip info file, contact HI-TECH Support with details.

(824) duplicate LIB for "*" in chipinfo file at line * ***(Assembler)***

The chipinfo file has a processor section with multiple `LIB` values. Only one `LIB` value is allowed. If you have not manually edited the chip info file, contact HI-TECH Support with details.

(825) too many RAMBANK lines in chipinfo file for "*" ***(Assembler)***

The chipinfo file contains a processor section with too many RAMBANK fields. Reduce the number of values.

(826) inverted ram bank in chipinfo file at line * ***(Assembler, Driver)***

The second hex number specified in the RAM field in the chipinfo file must be greater in value than the first.

(827) too many COMMON lines in chipinfo file for "*" ***(Assembler)***

There are too many lines specifying common (access bank) memory in the chip configuration file.

(828) inverted common bank in chipinfo file at line * ***(Assembler, Driver)***

The second hex number specified in the COMMON field in the chipinfo file must be greater in value than the first. Contact HI-TECH Support if you have not modified the chipinfo INI file.

(829) unrecognized line in chipinfo file at line * *(Assembler)*

The chipinfo file contains a processor section with an unrecognised line. Contact HI-TECH Support if the INI has not been edited.

(830) missing ARCH specification for "*" in chipinfo file *(Assembler)*

The chipinfo file has a processor section without an ARCH values. The architecture of the processor must be specified. Contact HI-TECH Support if the chipinfo file has not been modified.

(832) empty chip info file "*" *(Assembler)*

The chipinfo file contains no data. If you have not manually edited the chip info file, contact HI-TECH Support with details.

(833) no valid entries in chipinfo file *(Assembler)*

The chipinfo file contains no valid processor descriptions.

(834) page width must be >= 60 *(Assembler)*

The listing page width must be at least 60 characters. Any less will not allow a properly formatted listing to be produced, e.g.:

```
LIST C=10  ; the page width will need to be wider than this
```

(835) form length must be >= 15 *(Assembler)*

The form length specified using the `-Flength` option must be at least 15 lines. Setting this length to zero is allowed and turns off paging altogether. The default value is zero (pageless).

(836) no file arguments *(Assembler)*

The assembler has been invoked without any file arguments. It cannot assemble anything.

(839) relocation too complex *(Assembler)*

The complex relocation in this expression is too big to be inserted into the object file.

(840) phase error *(Assembler)*

The assembler has calculated a different value for a symbol on two different passes. This is probably due to bizarre use of macros or conditional assembly.

(841) bad source/destination for movfp/movpf instruction *(Assembler)*

The absolute address specified with the `movfp`/`movpf` instruction is too large.

(842) bad bit number *(Assembler, Optimiser)*

A bit number must be an absolute expression in the range 0-7.

(843) a macro name can't also be an EQU/SET symbol *(Assembler)*

An EQU or SET symbol has been found with the same name as a macro. This is not allowed. For example:

```
getval MACRO
  mov  r0, r1
ENDM
getval EQU 55h  ; oops -- choose a different name to the macro
```

(844) lexical error *(Assembler, Optimiser)*

An unrecognized character or token has been seen in the input.

(845) symbol "*" defined more than once *(Assembler)*

This symbol has been defined in more than one place. The assembler will issue this error if a symbol is defined more than once in the same module, e.g.:

```
_next:
  move r0, #55
  move [r1], r0
_next:           ; oops -- choose a different name
```

The linker will issue this warning if the symbol (C or assembler) was defined multiple times in different modules. The names of the modules are given in the error message. Note that C identifiers often have an *underscore* prepended to their name after compilation.

(846) relocation error *(Assembler, Optimiser)*

It is not possible to add together two relocatable quantities. A constant may be added to a relocatable value, and two relocatable addresses in the same psect may be subtracted. An absolute value must be used in various places where the assembler must know a value at assembly time.

(847) operand error *(Assembler, Optimiser)*

The operand to this opcode is invalid. Check your assembler reference manual for the proper form of operands for this instruction.

(848) symbol has been declared EXTERN *(Assembler)*

An assembly label uses the same name as a symbol that has already been declared as `EXTERN`.

(849) illegal instruction for this processor *(Assembler)*

The instruction is not supported by this processor.

(850) PAGESEL not usable with this processor *(Assembler)*

The PAGESEL pseudo-instruction is not usable with the device selected.

(851) illegal destination *(Assembler)*

The destination (either `,f` or `,w`) is not correct for this instruction.

(852) radix must be from 2 - 16 *(Assembler)*

The radix specified using the `RADIX` assembler directive must be in the range from 2 (binary) to 16 (hexadecimal).

(853) invalid size for FNSIZE directive *(Assembler)*

The assembler FNSIZE assembler directive arguments must be positive constants.

(855) ORG argument must be a positive constant *(Assembler)*

An argument to the `ORG` assembler directive must be a positive constant or a symbol which has been equated to a positive constant, e.g.:

```
ORG -10  /* this must a positive offset to the current psect */
```

(856) ALIGN argument must be a positive constant ***(Assembler)***

The `align` assembler directive requires a non-zero positive integer argument.

(857) psect may not be local and global ***(Linker)***

A local psect may not have the same name as a global psect, e.g.:

```
psect text,class=CODE       ; text is implicitly global
  move r0, r1
; elsewhere:
psect text,local,class=CODE
  move r2, r4
```

The `global` flag is the default for a psect if its scope is not explicitly stated.

(859) argument to C option must specify a positive constant ***(Assembler)***

The parameter to the `LIST` assembler control's `C=` option (which sets the column width of the listing output) must be a positive decimal constant number, e.g.:

```
LIST C=a0h  ; constant must be decimal and positive,
              try: LIST C=80
```

(860) page width must be >= 49 ***(Assembler)***

The page width suboption to the `LIST` assembler directive must specify a with of at least 49.

(861) argument to N option must specify a positive constant ***(Assembler)***

The parameter to the `LIST` assembler control's `N` option (which sets the page length for the listing output) must be a positive constant number, e.g.:

```
LIST N=-3  ; page length must be positive
```

(862) symbol is not external ***(Assembler)***

A symbol has been declared as EXTRN but is also defined in the current module.

(863) symbol can't be both extern and public *(Assembler)*

If the symbol is declared as extern, it is to be imported. If it is declared as public, it is to be exported from the current module. It is not possible for a symbol to be both.

(864) argument to "size" psect flag must specify a positive constant *(Assembler)*

The parameter to the `PSECT` assembler directive's `size` option must be a positive constant number, e.g.:

```
PSECT text,class=CODE,size=-200  ; a negative size?
```

(865) psect flag "size" redefined *(Assembler)*

The `size` flag to the `PSECT` assembler directive is different from a previous `PSECT` directive, e.g.:

```
psect spdata,class=RAM,size=400
; elsewhere:
psect spdata,class=RAM,size=500
```

(866) argument to "reloc" psect flag must specify a positive constant *(Assembler)*

The parameter to the `PSECT` assembler directive's `reloc` option must be a positive constant number, e.g.:

```
psect test,class=CODE,reloc=-4  ; the reloc must be positive
```

(867) psect flag "reloc" redefined *(Assembler)*

The `reloc` flag to the `PSECT` assembler directive is different from a previous `PSECT` directive, e.g.:

```
psect spdata,class=RAM,reloc=4
; elsewhere:
psect spdata,class=RAM,reloc=8
```

(868) argument to "delta" psect flag must specify a positive constant *(Assembler)*

The parameter to the `PSECT` assembler directive's `DELTA` option must be a positive constant number, e.g.:

```
PSECT text,class=CODE,delta=-2  ; negative delta value doesn't make sense
```

(869) psect flag "delta" redefined ***(Assembler)***

The 'DELTA' option of a psect has been redefined more than once in the same module.

(870) argument to "pad" psect flag must specify a positive constant ***(Assembler)***

The parameter to the PSECT assembler directive's 'PAD' option must be a non-zero positive integer.

(871) argument to "space" psect flag must specify a positive constant ***(Assembler)***

The parameter to the PSECT assembler directive's `space` option must be a positive constant number, e.g.:

```
PSECT text,class=CODE,space=-1  ; space values start at zero
```

(872) psect flag "space" redefined ***(Assembler)***

The `space` flag to the `PSECT` assembler directive is different from a previous `PSECT` directive, e.g.:

```
psect spdata,class=RAM,space=0
; elsewhere:
psect spdata,class=RAM,space=1
```

(873) a psect may only be in one class ***(Assembler)***

You cannot assign a psect to more than one class. The psect was defined differently at this point than when it was defined elsewhere. A psect's class is specified via a flag as in the following:

```
psect text,class=CODE
```

Look for other psect definitions that specify a different class name.

(874) a psect may only have one "with" option ***(Assembler)***

A psect can only be placed `with` one other psect. A psect's with option is specified via a flag as in the following:

```
psect bss,with=data
```

Look for other psect definitions that specify a different `with` psect name.

(875) bad character constant in expression *(Assembler,Optimizer)*

The character constant was expected to consist of only one character, but was found to be greater than one character or none at all. An assembler specific example:

```
mov  r0, #'12'    ; '12' specifies two characters
```

(876) syntax error *(Assembler, Optimiser)*

A syntax error has been detected. This could be caused a number of things.

(877) yacc stack overflow *(Assembler)*

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(878) -S option used: "*" ignored *(Driver)*

The indicated assembly file has been supplied to the driver in conjunction with the `-S` option. The driver really has nothing to do since the file is already an assembly file.

(880) invalid number of parameters. Use "* –HELP" for help *(Driver)*

Improper command-line usage of the of the compiler's driver.

(881) setup succeeded *(Driver)*

The compiler has been successfully setup using the `--setup` driver option.

(883) setup failed *(Driver)*

The compiler was not successfully setup using the `--setup` driver option. Ensure that the directory argument to this option is spelt correctly, is syntactically correct for your host operating system and it exists.

(884) please ensure you have write permissions to the configuration file *(Driver)*

The compiler was not successfully setup using the `--setup` driver option because the driver was unable to access the XML configuration file. Ensure that you have write permission to this file. The driver will search the following configuration files in order:

- the file specified by the environment variable `HTC_XML`

- the file `/etc/htsoft.xml` if the directory '`/etc`' is writable and there is no `.htsoft.xml` file in your home directory
- the file `.htsoft.xml` file in your home directory

If none of the files can be located then the above error will occur.

(889) this * compiler has expired ***(Driver)***

The demo period for this compiler has concluded.

(890) contact HI-TECH Software to purchase and re-activate this compiler ***(Driver)***

The evaluation period of this demo installation of the compiler has expired. You will need to purchase the compiler to re-activate it. If however you sincerely believe the evaluation period has ended prematurely please contact HI-TECH technical support.

(891) can't open psect usage map file "*": * ***(Driver)***

The driver was unable to open the indicated file. The psect usage map file is generated by the driver when the driver option `--summary=file` is used. Ensure that the file is not open in another application.

(892) can't open memory usage map file "*": * ***(Driver)***

The driver was unable to open the indicated file. The memory usage map file is generated by the driver when the driver option `--summary=file` is used. Ensure that the file is not open in another application.

(893) can't open HEX usage map file "*": * ***(Driver)***

The driver was unable to open the indicated file. The HEX usage map file is generated by the driver when the driver option `--summary=file` is used. Ensure that the file is not open in another application.

(894) unknown source file type "*" ***(Driver)***

The extension of the indicated input file could not be determined. Only files with the extensions `as`, `c`, `obj`, `usb`, `p1`, `lib` or `hex` are identified by the driver.

(895) can't request and specify options in the one command ***(Driver)***

The usage of the driver options `--getoption` and `--setoption` is mutually exclusive.

(896) no memory ranges specified for data space ***(Driver)***

No on-chip or external memory ranges have been specified for the data space memory for the device specified.

(897) no memory ranges specified for program space ***(Driver)***

No on-chip or external memory ranges have been specified for the program space memory for the device specified.

(899) can't open option file "*" for application "*": * ***(Driver)***

An option file specified by a `--getoption` or `--setoption` driver option could not be opened. If you are using the `--setoption` option ensure that the name of the file is spelt correctly and that it exists. If you are using the `--getoption` option ensure that this file can be created at the given location or that it is not in use by any other application.

(900) exec failed: * ***(Driver)***

The subcomponent listed failed to execute. Does the file exist? Try re-installing the compiler.

(902) no chip name specified; use "* –CHIPINFO" to see available chip names ***(Driver)***

The driver was invoked without selecting what chip to build for. Running the driver with the –CHIPINFO option will display a list of all chips that could be selected to build for.

(904) illegal format specified in "*" option ***(Driver)***

The usage of this option was incorrect. Confirm correct usage with –HELP or refer to the part of the manual that discusses this option.

(905) illegal application specified in "*" option ***(Driver)***

The application given to this option is not understood or does not belong to the compiler.

(907) unknown memory space tag "*" in "*" option specification ***(Driver)***

A parameter to this memory option was a string but did not match any valid *tags*. Refer to the section of this manual that describes this option to see what tags (if any) are valid for this device.

(908) exit status = * ***(Driver)***

One of the subcomponents being executed encountered a problem and returned an error code. Other messages should have been reported by the subcomponent to explain the problem that was encountered.

(913) "*" option may cause compiler errors in some standard header files ***(Driver)***

Using this option will invalidate some of the qualifiers used in the standard header files resulting in errors. This issue and its solution are detailed in the section of this manual that specifically discusses this option.

(915) no room for arguments ***(Preprocessor, Parser, Code Generator, Linker, Objtohex)***

The code generator could not allocate any more memory.

(917) argument too long ***(Preprocessor, Parser)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(918) *: no match ***(Preprocessor, Parser)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(919) * in chipinfo file "*" at line * ***(Driver)***

The specified parameter in the chip configuration file is illegal.

(920) empty chipinfo file ***(Driver, Assembler)***

The chip configuration file was able to be opened but it was empty. Try re-installing the compiler.

(922) chip "*" not present in chipinfo file "*" ***(Driver)***

The chip selected does not appear in the compiler's chip configuration file. You may need to contact HI-TECH Software to see if support for this device is available or upgrade the version of your compiler.

(923) unknown suboption "*" ***(Driver)***

This option can take suboptions, but this suboption is not understood. This may just be a simple spelling error. If not, –HELP to look up what suboptions are permitted here.

(924) missing argument to "*" option ***(Driver)***

This option expects more data but none was given. Check the usage of this option.

(925) extraneous argument to "*" option ***(Driver)***

This option does not accept additional data, yet additional data was given. Check the usage of this option.

(926) duplicate "*" option ***(Driver)***

This option can only appear once, but appeared more than once.

(928) bad "*" option value ***(Driver, Assembler)***

The indicated option was expecting a valid hexadecimal integer argument.

(929) bad "*" option ranges ***(Driver)***

This option was expecting a parameter in a range format (*start_of_range-end_of_range*), but the parameter did not conform to this syntax.

(930) bad "*" option specification ***(Driver)***

The parameters to this option were not specified correctly. Run the driver with –HELP or refer to the driver's chapter in this manual to verify the correct usage of this option.

(931) command file not specified ***(Driver)***

Command file to this application, expected to be found after `'@'` or `'<'` on the command line was not found.

(939) no file arguments ***(Driver)***

The driver has been invoked with no input files listed on its command line. If you are getting this message while building through a third party IDE, perhaps the IDE could not verify the source files to compile or object files to link and withheld them from the command line.

(940) *-bit checksum * placed at * ***(Objtohex)***

Presenting the result of the requested checksum calculation.

(941) bad "*" assignment; USAGE: ** ***(Hexmate)***

An option to Hexmate was incorrectly used or incomplete. Follow the usage supplied by the message and ensure that that the option has been formed correctly and completely.

(942) unexpected character on line * of file "*" ***(Hexmate)***

File contains a character that was not valid for this type of file, the file may be corrupt. For example, an Intel hex file is expected to contain only ASCII representations of hexadecimal digits, colons (:) and line formatting. The presence of any other characters will result in this error.

(944) data conflict at address *h between * and * ***(Hexmate)***

Sources to Hexmate request differing data to be stored to the same address. To force one data source to override the other, use the '+' specifier. If the two named sources of conflict are the same source, then the source may contain an error.

(945) checksum range (*h to *h) contained an indeterminate value ***(Hexmate)***

The range for this checksum calculation contained a value that could not be resolved. This can happen if the checksum result was to be stored within the address range of the checksum calculation.

(948) checksum result width must be between 1 and 4 bytes ***(Hexmate)***

The requested checksum byte size is illegal. Checksum results must be within 1 to 4 bytes wide. Check the parameters to the -CKSUM option.

(949) start of checksum range must be less than end of range ***(Hexmate)***

The -CKSUM option has been given a range where the start is greater than the end. The parameters may be incomplete or entered in the wrong order.

(951) start of fill range must be less than end of range ***(Hexmate)***

The -FILL option has been given a range where the start is greater than the end. The parameters may be incomplete or entered in the wrong order.

(953) unknown -HELP sub-option: * ***(Hexmate)***

Invalid sub-option passed to -HELP. Check the spelling of the sub-option or use -HELP with no sub-option to list all options.

(956) -SERIAL value must be between 1 and * bytes long ***(Hexmate)***

The serial number being stored was out of range. Ensure that the serial number can be stored in the number of bytes permissible by this option.

(958) too many input files specified; * file maximum ***(Hexmate)***

Too many file arguments have been used. Try merging these files in several stages rather than in one command.

(960) unexpected record type (*) on line * of "*" ***(Hexmate)***

Intel hex file contained an invalid record type. Consult the Intel hex format specification for valid record types.

(962) forced data conflict at address *h between * and * ***(Hexmate)***

Sources to Hexmate force differing data to be stored to the same address. More than one source using the '+' specifier store data at the same address. The actual data stored there may not be what you expect.

(963) checksum range includes voids or unspecified memory locations ***(Hexmate)***

Checksum range had gaps in data content. The runtime calculated checksum is likely to differ from the compile-time checksum due to gaps/unused byes within the address range that the checksum is calculated over. Filling unused locations with a known value will correct this.

(964) unpaired nibble in -FILL value will be truncated ***(Hexmate)***

The hexadecimal code given to the FILL option contained an incomplete byte. The incomplete byte (nibble) will be disregarded.

(965) -STRPACK option not yet implemented, option will be ignored ***(Hexmate)***

This option currently is not available and will be ignored.

(966) no END record for HEX file "*" ***(Hexmate)***

Intel hex file did not contain a record of type END. The hex file may be incomplete.

(967) unused function definition "*" (from line *) ***(Parser)***

The indicated `static` function was never called in the module being compiled. Being static, the function cannot be called from other modules so this warning implies the function is never used. Either the function is redundant, or the code that was meant to call it was excluded from compilation or misspelt the name of the function.

(968) unterminated string ***(Assembler, Optimiser)***

A string constant appears not to have a closing quote missing.

(969) end of string in format specifier ***(Parser)***

The format specifier for the printf() style function is malformed.

(970) character not valid at this point in format specifier ***(Parser)***

The printf() style format specifier has an illegal character.

(971) type modifiers not valid with this format ***(Parser)***

Type modifiers may not be used with this format.

(972) only modifiers "h" and "l" valid with this format ***(Parser)***

Only modifiers `h` (`short`) and `l` (`long`) are legal with this `printf` format specifier.

(973) only modifier "l" valid with this format ***(Parser)***

The only modifier that is legal with this format is `l` (for long).

(974) type modifier already specified ***(Parser)***

This type modifier has already be specified in this type.

(975) invalid format specifier or type modifier ***(Parser)***

The format specifier or modifier in the printf-style string is illegal for this particular format.

(976) field width not valid at this point ***(Parser)***

A field width may not appear at this point in a printf() type format specifier.

(978) this identifier is already an enum tag ***(Parser)***

This identifier following a `struct` or `union` keyword is already the tag for an enumerated type, and thus should only follow the keyword `enum`, e.g.:

```
enum IN {ONE=1, TWO};
struct IN {             /* oops -- IN is already defined */
  int a, b;
};
```

(979) this identifier is already a struct tag ***(Parser)***

This identifier following a `union` or `enum` keyword is already the tag for a structure, and thus should only follow the keyword `struct`, e.g.:

```
struct IN {
  int a, b;
};
enum IN {ONE=1, TWO};  /* oops -- IN is already defined */
```

(980) this identifier is already a union tag ***(Parser)***

This identifier following a `struct` or `enum` keyword is already the tag for a `union`, and thus should only follow the keyword `union`, e.g.:

```
union IN {
  int a, b;
};
enum IN {ONE=1, TWO};  /* oops -- IN is already defined */
```

(981) pointer required ***(Parser)***

A pointer is required here, e.g.:

```
struct DATA data;
data->a = 9;       /* data is a structure,
                      not a pointer to a structure */
```

(982) unknown op "*" in nxtuse() ***(Optimiser,Assembler)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(983) storage class redeclared ***(Parser)***

A variable previously declared as being *static*, has now be redeclared as *extern*.

(984) type redeclared ***(Parser)***

The type of this function or object has been redeclared. This can occur because of two incompatible declarations, or because an implicit declaration is followed by an incompatible declaration, e.g.:

```
int a;
char a;  /* oops -- what is the correct type? */
```

(985) qualifiers redeclared ***(Parser)***

This function or variable has different qualifiers in different declarations.

(986) enum member redeclared ***(Parser)***

A member of an enumeration is defined twice or more with differing values. Does the member appear twice in the same list or does the name of the member appear in more than one enum list?

(987) arguments redeclared ***(Parser)***

The data types of the parameters passed to this function do not match its prototype.

(988) number of arguments redeclared ***(Parser)***

The number of arguments in this function declaration does not agree with a previous declaration of the same function.

(989) module has code below file base of *h ***(Linker)***

This module has code below the address given, but the `-C` option has been used to specify that a binary output file is to be created that is mapped to this address. This would mean code from this module would have to be placed before the beginning of the file! Check for missing psect directives in assembler files.

(990) modulus by zero in #if; zero result assumed *(Preprocessor)*

A modulus operation in a `#if` expression has a zero divisor. The result has been assumed to be zero, e.g.:

```
#define ZERO 0
#if FOO%ZERO    /* this will have an assumed result of 0 */
  #define INTERESTING
#endif
```

(991) integer expression required *(Parser)*

In an `enum` declaration, values may be assigned to the members, but the expression must evaluate to a constant of type `int`, e.g.:

```
enum {one = 1, two, about_three = 3.12};
  /* no non-int values allowed */
```

(992) can't find op *(Assembler, Optimiser)*

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(993) some command-line options are disabled *(Driver)*

The compiler is operating in demo mode. Some command-line options are disabled.

(994) some command-line options are disabled and compilation is delayed *(Driver)*

The compiler is operating in demo mode. Some command-line options are disabled, the compilation speed will be slower.

(995) some command-line options are disabled, code size is limited to 16kB, compilation is delayed *(Driver)*

The compiler is operating in demo mode. Some command-line options are disabled, the compilation speed will be slower, and the maximum allowed code size is limited to 16kB.

(1015) missing "*" specification in chipinfo file "*" at line * *(Driver)*

This attribute was expected to appear at least once but was not defined for this chip.

(1016) missing argument* to "*" specification in chipinfo file "*" at line * ***(Driver)***

This value of this attribute is blank in the chip configuration file.

(1017) extraneous argument* to "*" specification in chipinfo file "*" at line * ***(Driver)***

There are too many attributes for the the listed specification in the chip configuration file.

(1018) illegal number of "*" specification* (* found; * expected) in chipinfo file "*" at line * ***(Driver)***

This attribute was expected to appear a certain number of times but it did not for this chip.

(1019) duplicate "*" specification in chipinfo file "*" at line * ***(Driver)***

This attribute can only be defined once but has been defined more than once for this chip.

(1020) unknown attribute "*" in chipinfo file "*" at line * ***(Driver)***

The chip configuration file contains an attribute that is not understood by this version of the compiler. Has the chip configuration file or the driver been replaced with an equivalent component from another version of this compiler?

(1021) syntax error reading "*" value in chipinfo file "*" at line * ***(Driver)***

The chip configuration file incorrectly defines the specified value for this device. If you are modifying this file yourself, take care and refer to the comments at the beginning of this file for a description on what type of values are expected here.

(1022) syntax error reading "*" range in chipinfo file "*" at line * ***(Driver)***

The chip configuration file incorrectly defines the specified range for this device. If you are modifying this file yourself, take care and refer to the comments at the beginning of this file for a description on what type of values are expected here.

(1024) syntax error in chipinfo file "*" at line * ***(Driver)***

The chip configuration file contains a syntax error at the line specified.

(1025) unknown architecture in chipinfo file "*" at line * ***(Driver)***

The attribute at the line indicated defines an architecture that is unknown to this compiler.

(1026) missing architecture in chipinfo file "*" at line * ***(Assembler)***

The chipinfo file has a processor section without an ARCH values. The architecture of the processor must be specified. Contact HI-TECH Support if the chipinfo file has not been modified.

(1027) activation was successful ***(Driver)***

The compiler was successfully activated.

(1028) activation was not successful - error code (*) ***(Driver)***

The compiler did not activated successfully.

(1029) compiler not installed correctly - error code (*) ***(Driver)***

This compiler has failed to find any activation information and cannot proceed to execute. The compiler may have been installed incorrectly or incompletely. The error code quoted can help diagnose the reason for this failure. You may be asked for this failure code if contacting HI-TECH Software for assistance with this problem.

(1030) HEXMATE - Intel hex editing utility (Build 1.%i) ***(Hexmate)***

Indicating the version number of the Hexmate being executed.

(1031) USAGE: * [input1.hex] [input2.hex]... [inputN.hex] [options] ***(Hexmate)***

The suggested usage of Hexmate.

(1032) use –HELP=<option> for usage of these command line options ***(Hexmate)***

More detailed information is available for a specific option by passing that option to the HELP option.

(1033) available command-line options: ***(Hexmate)***

This is a simple heading that appears before the list of available options for this application.

(1034) type "*" for available options ***(Hexmate)***

It looks like you need help. This advisory suggests how to get more information about the options available to this application or the usage of these options.

(1035) bad argument count (*) ***(Parser)***

The number of arguments to a function is unreasonable. This is an internal compiler error. Contact HI-TECH Software technical support with details.

(1036) bad "*" optional header length (0x* expected) ***(Cromwell)***

The length of the optional header in this COFF file was of an incorrect length.

(1037) short read on * ***(Cromwell)***

When reading the type of data indicated in this message, it terminated before reaching its specified length.

(1038) string table length too short ***(Cromwell)***

The specified length of the COFF string table is less than the minimum.

(1039) inconsistent symbol count ***(Cromwell)***

The number of symbols in the symbol table has exceeded the number indicated in the COFF header.

(1040) bad checksum: record 0x*, checksum 0x* ***(Cromwell)***

A record of the type specified failed to match its own checksum value.

(1041) short record ***(Cromwell)***

While reading a file, one of the file's records ended short of its specified length.

(1042) unknown * record type 0x* ***(Cromwell)***

The type indicator of this record did not match any valid types for this file format.

(1043) unknown optional header ***(Cromwell)***

When reading this Microchip COFF file, the optional header within the file header was of an incorrect length.

(1044) end of file encountered ***(Cromwell, Linker)***

The end of the file was found while more data was expected. Has this input file been truncated?

(1045) short read on block of * bytes ***(Cromwell)***

A while reading a block of byte data from a UBROF record, the block ended before the expected length.

(1046) short string read ***(Cromwell)***

A while reading a string from a UBROF record, the string ended before the specified length.

(1047) bad type byte for UBROF file ***(Cromwell)***

This UBROF file did not begin with the correct record.

(1048) bad time/date stamp ***(Cromwell)***

This UBROF file has a bad time/date stamp.

(1049) wrong CRC on 0x* bytes; should be * ***(Cromwell)***

An end record has a mismatching CRC value in this UBROF file.

(1050) bad date in 0x52 record ***(Cromwell)***

A debug record has a bad date component in this UBROF file.

(1051) bad date in 0x01 record ***(Cromwell)***

A start of program record or segment record has a bad date component in this UBROF file.

(1052) unknown record type ***(Cromwell)***

A record type could not be determined when reading this UBROF file.

(1053) additional RAM ranges larger than bank size ***(Driver)***

A block of additional RAM being requested exceeds the size of a bank. Try breaking the block into multiple ranges that do not cross bank boundaries.

(1054) additional RAM range out of bounds ***(Driver)***

The RAM memory range as defined through custom RAM configuration is out of range.

(1055) RAM range out of bounds (*) ***(Driver)***

The RAM memory range as defined in the chip configuration file or through custom configuration is out of range.

(1056) unknown chip architecture ***(Driver)***

The compiler is attempting to compile for a device of an architecture that is either unsupported or disabled.

(1057) fast double option only available on 17 series processors ***(Driver)***

The fast double library cannot be selected for this device. These routines are only available for PIC17 devices.

(1058) assertion ***(Code Generator)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(1059) rewrite loop ***(Code Generator)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(1081) static initialization of persistent variable "*" ***(Parser, Code Generator)***

A persistent variable has been assigned an initial value. This is somewhat contradictory as the initial value will be assigned to the variable during execution of the compiler's startup code, however the *persistent* qualifier requests that this variable shall be unchanged by the compiler's startup code.

(1082) size of initialized array element is zero ***(Code Generator)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(1088) function pointer "*" is used but never assigned a value ***(Code Generator)***

A function call involving a function pointer was made, but the pointer was never assigned a target address, e.g.:

```
void (*fp)(int);
fp(23);     /* oops -- what function does fp point to? */
```

(1089) recursive function call to "*" ***(Code Generator)***

A recursive call to the specified function has been found. The call may be direct or indirect (using function pointers) and may be either a function calling itself, or calling another function whose call graph includes the function under consideration.

(1090) variable "*" is not used ***(Code Generator)***

This variable is declared but has not been used by the program. Consider removing it from the program.

(1091) main function "*" not defined ***(Code Generator)***

The *main* function has not been defined. Every C program must have a function called *main*.

(1094) bad derived type ***(Code Generator)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(1095) bad call to typeSub() ***(Code Generator)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(1096) type should be unqualified ***(Code Generator)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(1097) unknown type string "*" ***(Code Generator)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(1098) conflicting declarations for variable "*" (*:*) ***(Parser, Code Generator)***

Differing type information has been detected in the declarations for a variable, or between a declaratin and the definition of a variable, e.g.:

```
extern long int test;
int test;    /* oops -- which is right? int or long int ? */
```

(1104) unqualified error ***(Code Generator)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(1118) bad string "*" in getexpr(J) ***(Code Generator)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(1119) bad string "*" in getexpr(LRN) ***(Code Generator)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(1121) expression error ***(Code Generator)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(1137) match() error: * ***(Code Generator)***

This is an internal compiler error. Contact HI-TECH Software technical support with details.

(1157) W register must be W9 ***(Assembler)***

The working register required here has to be W9, but an other working register was selected.

(1159) W register must be W11 ***(Assembler)***

The working register required here has to be W11, but an other working register was selected.

(1178) the "*" option has been removed and has no effect ***(Driver)***

This option no longer exists in this version of the compiler and has been ignored. Use the compiler's *–help* option or refer to the manual to find a replacement option.

(1179) interrupt level for function "*" may not exceed * ***(Code Generator)***

The interrupt level for the function specified is too high. Each interrupt function is assigned a unique interrupt level. This level is considered when analysing the call graph and re-entrantly called functions. If using the `interrupt_level` pragma, check the value specified.

(1180) directory "*" does not exist ***(Driver)***

The directory specified in the setup option does not exist. Create the directory and try again.

(1182) near variables must be global or static ***(Code Generator)***

A variable qualified as *near* must also be qualified with *static* or made global. An auto variable cannot be qualified as *near*.

(1183) invalid version number ***(Activation)***

During activation, no matching version number was found on the HI-TECH activation server database for the serial number specified.

(1184) activation limit reached ***(Activation)***

The number of activations of the serial number specified has exceeded the maximum number allowed for the license.

(1185) invalid serial number ***(Activation)***

During activation, no matching serial number was found on the HI-TECH activation server database.

(1186) licence has expired ***(Driver)***

The time-limited license for this compiler has expired.

(1187) invalid activation request ***(Driver)***

The compiler has not been correctly activated.

(1188) network error * ***(Activation)***

The compiler activation software was unable to connect to the HI-TECH activation server via the network.

(1190) FAE license only - not for use in commercial applications ***(Driver)***

Indicates that this compiler has been activated with an FAE licence. This licence does not permit the product to be used for the development of commercial applications.

(1191) licensed for educational use only ***(Driver)***

Indicates that this compiler has been activated with an education licence. The educational licence is only available to educational facilities and does not permit the product to be used for the development of commercial applications.

(1192) licensed for evaluation purposes only ***(Driver)***

Indicates that this compiler has been activated with an evaluation licence.

(1193) this licence will expire on * ***(Driver)***

The compiler has been installed as a time-limited trial. This trial will end on the date specified.

(1195) invalid syntax for "*" option ***(Driver)***

A command line option that accepts additional parameters was given inappropriate data or insufficient data. For example an option may expect two parameters with both being integers. Passing a string as one of these parameters or supplying only one parameter could result in this error.

(1198) too many "*" specifications; * maximum ***(Hexmate)***

This option has been specified too many times. If possible, try performing these operations over several command lines.

(1199) compiler has not been activated ***(Driver)***

The trial period for this compiler has expired. The compiler is now inoperable until activated with a valid serial number. Contact HI-TECH Software to purchase this software and obtain a serial number.

(1200) Found %0*lXh at address *h ***(Hexmate)***

The code sequence specified in a -FIND option has been found at this address.

(1201) all FIND/REPLACE code specifications must be of equal width ***(Hexmate)***

All find, replace and mask attributes in this option must be of the same byte width. Check the parameters supplied to this option. For example finding 1234h (2 bytes) masked with FFh (1 byte) will result in an error, but masking with 00FFh (2 bytes) will be Ok.

(1202) unknown format requested in -FORMAT: * ***(Hexmate)***

An unknown or unsupported INHX format has been requested. Refer to documentation for supported INHX formats.

(1203) unpaired nibble in * value will be truncated ***(Hexmate)***

Data to this option was not entered as whole bytes. Perhaps the data was incomplete or a leading zero was omitted. For example the value Fh contains only four bits of significant data and is not a whole byte. The value 0Fh contains eight bits of significant data and is a whole byte.

(1204) * value must be between 1 and * bytes long ***(Hexmate)***

An illegal length of data was given to this option. The value provided to this option exceeds the maximum or minimum bounds required by this option.

(1205) using the configuration file *; you may override this with the environment variable HTC_XML ***(Driver)***

This is the compiler configuration file selected during compiler setup. This can be changed via the HTC_XML environment variable. This file is used to determine where the compiler has been installed.

(1207) some of the command line options you are using are now obsolete ***(Driver)***

Some of the command line options passed to the driver have now been discontinued in this version of the compiler, however during a grace period these old options will still be processed by the driver.

(1208) use –help option or refer to the user manual for option details ***(Driver)***

An obsolete option was detected. Use –help or refer to the manual to find a replacement option that will not result in this advisory message.

(1209) An old MPLAB tool suite plug-in was detected. ***(Driver)***

The options passed to the driver resemble those that the Microchip MPLAB IDE would pass to a previous version of this compiler. Some of these options are now obsolete, however they were still interpreted. It is recommended that you install an updated HI-TECH options plug-in for the MPLAB IDE.

(1210) Visit the HI-TECH Software website (www.htsoft.com) for a possible update ***(Driver)***

Visit our website to see if an update is available to address the issue(s) listed in the previous compiler message. Please refer to the on-line self-help facilities such as the *Frequently asked Questions* or search the *On-line forums*. In the event of no details being found here, contact HI-TECH Software for further information.

(1212) Found * (%0*lXh) at address *h ***(Hexmate)***

The code sequence specified in a -FIND option has been found at this address.

(1213) duplicate ARCH for * in chipinfo file at line * ***(Assembler, Driver)***

The chipinfo file has a processor section with multiple ARCH values. Only one ARCH value is allowed. If you have not manually edited the chip info file, contact HI-TECH Support with details.

(1218) can't create cross reference file * ***(Assembler)***

The assembler attempted to create a cross reference file, but it could not be created. Check that the file's pathname is correct.

(1228) unable to locate installation directory ***(Driver)***

The compiler cannot determine the directory where it has been installed.

(1230) dereferencing uninitialized pointer "*" ***(Code Generator)***

A pointer that has not yet been assigned a value has been dereferenced. This can result in erroneous behaviour at runtime.

(1235) unknown keyword * ***(Driver)***

The token contained in the USB descriptor file was not recognised.

(1236) invalid argument to *: * ***(Driver)***

An option that can take additional parameters was given an invalid parameter value. Check the usage of the option or the syntax or range of the expected parameter.

(1237) endpoint 0 is pre-defined ***(Driver)***

An attempt has been made to define endpoint 0 in a USB file. This channel c

(1238) FNALIGN failure on * ***(Linker)***

Two functions have their auto/parameter blocks aligned using the `FNALIGN` directive, but one function calls the other, which implies that must not be aligned. This will occur if a function pointer is assigned the address of each function, but one function calls the other. For example:

```
int one(int a) { return a; }
int two(int a) { return two(a)+2; }  /* ! */
int (*ip)(int);
ip = one;
ip(23);
ip = two;      /* ip references one and two; two calls one */
ip(67);
```

(1239) pointer * has no valid targets ***(Code Generator)***

A function call involving a function pointer was made, but the pointer was never assigned a target address, e.g.:

```
void (*fp)(int);
fp(23);     /* oops -- what function does fp point to? */
```

(1240) unknown checksum algorithm type (%i) ***(Driver)***

The error file specified after the `-Efile` or `-E+file` options could not be opened. Check to ensure that the file or directory is valid and that has read only access.

(1241) bad start address in * ***(Driver)***

The start of range address for the `--CHECKSUM` option could not be read. This value must be a hexadecimal number.

(1242) bad end address in * *(Driver)*

The end of range address for the `--CHECKSUM` option could not be read. This value must be a hexadecimal number.

(1243) bad destination address in * *(Driver)*

The destination address for the `--CHECKSUM` option could not be read. This value must be a hexadecimal number.

(1245) value greater than zero required for * *(Hexmate)*

The *align* operand to the HEXMATE `-FIND` option must be positive.

(1246) no RAM defined for variable placement *(Code Generator)*

No memory has been specified to cover the banked RAM memory.

(1247) no access RAM defined for variable placement *(Code Generator)*

No memory has been specified to cover the access bank memory.

(1248) symbol (*) encountered with undefined type size *(Code Generator)*

The code generator was asked to position a variable, but the size of the variable is not known. This is an internal compiler error. Contact HI-TECH Software technical support with details.

(1250) could not find space (* byte*) for variable * *(Code Generator)*

The code generator could not find space in the banked RAM for the variable specified.

(1253) could not find space (* byte*) for auto/param block *(Code Generator)*

The code generator could not find space in RAM for the psect that holds `auto` and parameter variables.

(1254) could not find space (* byte*) for data block *(Code Generator)*

The code generator could not find space in RAM for the data psect that holds initialised variables.

(1255) conflicting paths for output directory ***(Driver)***

The compiler has been given contradictory paths for the output directory via any of the `-O` or `--OUTDIR` options, e.g.

```
--outdir=../../   -o../main.hex
```

(1256) undefined symbol "*" treated as hex constant ***(Assembler)***

A token which could either be interpreted as a symbol or a hexadecimal value does not match any previously defined symbol and so will be interpreted as the latter. Use a leading *zero* to avoid the ambiguity, or use an alternate radix sepcifier such as `0x`. For example:

```
mov  a, F7h  ; is this the symbol F7h, or the hex number 0xF7?
```

(1257) local variable "*" is used but never given a value ***(Code Generator)***

An `auto` variable has been defined and used in an expression, but it has not been assigned a value in the C code before its first use. Auto variables are not cleared on startup and their initial value is undefined. For example:

```
void main(void) {
  double src, out;
  out = sin(src);   /* oops -- what value was in src? */
```

(1258) possible stack overflow when calling function "*" ***(Code Generator)***

The call tree analysis by the code generator indicates that the hardware stack may overflow. This should be treated as a guide only. Interrupts, the assembler optimizer and the program structure may affect the stack usage. The stack usuage is based on the C program and does not include any call tree derived from assembly code.

(1259) can't optimize for both speed and space ***(Driver)***

The driver has been given contradictory options of compile for speed and compile for space, e.g.

```
--opt=speed,space
```

(1260) macro "*" redefined ***(Assembler)***

More than one definition for a macro with the same name has been encountered, e.g.

```
MACRO fin
  ret
ENDM
MACRO fin   ; oops -- was this meant to be a different macro?
  reti
ENDM
```

(1261) string constant required ***(Assembler)***

A string argument is required with the `DS` or `DSU` directive, e.g.

```
DS ONE     ; oops -- did you mean DS "ONE"?
```

(1264) unsafe pointer conversion ***(Code Generator)***

A pointer to one kind of structure has been converted to another kind of structure and the structures do not have a similar definition, e.g.

```
struct ONE {
  unsigned a;
  long b;          /* ! */
} one;
struct TWO {
  unsigned a;
  unsigned b;      /* ! */
} two;
struct ONE * oneptr;
oneptr = & two;    /* oops --
                   was ONE meant to be same struct as TWO? */
```

(1267) fixup overflow referencing * into * bytes at 0x* ***(Linker)***

See the following error message (1268) for more information..

(1268) fixup overflow storing 0x* in * bytes at * ***(Linker)***

Fixup is the process conducted by the linker of replacing symbolic references to variables etc, in an assembler instruction with an absolute value. This takes place after positioning the psects (program sections or blocks) into the available memory on the target device. Fixup overflow is when the value determined for a symbol is too large to fit within the allocated space within the assembler instruction. For example, if an assembler instruction has an 8-bit field to hold an address and the linker determines that the symbol that has been used to represent this address has the value 0x110, then clearly this value cannot be inserted into the instruction.

(0) delete what ? ***(Libr)***

The librarian requires one or more modules to be listed for deletion when using the `d` key, e.g.:

```
libr d c:\ht-pic\lib\pic704-c.lib
```

does not indicate which modules to delete. try something like:

```
libr d c:\ht-pic\lib\pic704-c.lib wdiv.obj
```

(0) incomplete ident record ***(Libr)***

The IDENT record in the object file was incomplete. Contact HI-TECH Support with details.

(0) incomplete symbol record ***(Libr)***

The SYM record in the object file was incomplete. Contact HI-TECH Support with details.

(0) library file names should have .lib extension: * ***(Libr)***

Use the `.lib` extension when specifying a library filename.

(0) module * defines no symbols ***(Libr)***

No symbols were found in the module's object file. This may be what was intended, or it may mean that part of the code was inadvertently removed or commented.

(0) replace what ? ***(Libr)***

The librarian requires one or more modules to be listed for replacement when using the `r` key, e.g.:

```
libr r lcd.lib
```

This command needs the name of a module (`.obj` file) after the library name.

Appendix C

Chip Information

The following table lists all devices currently supported by HI-TECH C PRO for the PIC10/12/16 MCU Family.

Table C.1: Devices supported by HI-TECH C PRO for the PIC10/12/16 MCU Family

DEVICE	ARCH	ROMSIZE	RAMBANK	EEPROMSIZE
10F200	PIC12	100	10-1F	
10F202	PIC12	200	08-1F	
10F204	PIC12	100	10-1F	
10F206	PIC12	200	08-1F	
10F220	PIC12	100	10-1F	
10F222	PIC12	200	09-1F	
12C508	PIC12	200	07-1F	
12F508	PIC12	200	07-1F	
12C509	PIC12	400	07-1F,30-3F	
12F509	PIC12	400	07-1F,30-3F	
12F510	PIC12	400	0A-1F,30-3F	
12F519	PIC12	400	07-1F,30-3F	
12C508A	PIC12	200	07-1F	
12C509A	PIC12	400	07-1F,30-3F	
12C509AG	PIC12	400	07-1F,30-3F	
RF509AG	PIC12	400	07-1F,30-3F	
12C509AF	PIC12	400	07-1F,30-3F	
RF509AF	PIC12	400	07-1F,30-3F	
12CR509A	PIC12	400	07-1F,30-3F	
12CE518	PIC12	200	07-1F	
12CE519	PIC12	400	07-1F,30-3F	
16C505	PIC12	400	08-1F,30-3F,50-5F,70-7F	
16F505	PIC12	400	08-1F,30-3F,50-5F,70-7F	

continued...

Table C.1: Devices supported by HI-TECH C PRO for the PIC10/12/16 MCU Family

DEVICE	ARCH	ROMSIZE	RAMBANK	EEPROMSIZE
16F506	PIC12	400	0D-1F,30-3F,50-5F,70-7F	
16F526	PIC12	400	0D-1F,30-3F,50-5F,70-7F	
16C52	PIC12	180	07-1F	
16C54	PIC12	200	07-1F	
16CR54A	PIC12	200	07-1F	
16CR54B	PIC12	200	07-1F	
16CR54C	PIC12	200	07-1F	
16HV540	PIC12	200	08-1F	
16C54A	PIC12	200	07-1F	
16C54B	PIC12	200	07-1F	
16C54C	PIC12	200	07-1F	
16F54	PIC12	200	07-1F	
16C55	PIC12	200	08-1F	
16C55A	PIC12	200	08-1F	
16C56	PIC12	400	07-1F	
16C56A	PIC12	400	07-1F	
16CR56A	PIC12	400	07-1F	
16C57	PIC12	800	08-1F,30-3F,50-5F,70-7F	
16C57C	PIC12	800	08-1F,30-3F,50-5F,70-7F	
16CR57B	PIC12	800	08-1F,30-3F,50-5F,70-7F	
16CR57C	PIC12	800	08-1F,30-3F,50-5F,70-7F	
16F57	PIC12	800	08-1F,30-3F,50-5F,70-7F	
16C58	PIC12	800	07-1F,30-3F,50-5F,70-7F	
16C58A	PIC12	800	07-1F,30-3F,50-5F,70-7F	
16C58B	PIC12	800	07-1F,30-3F,50-5F,70-7F	
16CR58A	PIC12	800	07-1F,30-3F,50-5F,70-7F	
16CR58B	PIC12	800	07-1F,30-3F,50-5F,70-7F	
16F59	PIC12	800	90-9F,B0-BF,D0-DF,E0-EF	
MCV08A	PIC12	400	0A-1F,30-3F	
MCV14A	PIC12	400	0D-1F,30-3F,50-5F,70-7F	
MCV18A	PIC12	200	07-1F	
MCV28A	PIC12	800	08-1F,30-3F,50-5F,70-7F	
12F609	PIC14	400	40-7F	
12HV609	PIC14	400	40-7F	
12F615	PIC14	400	40-7F	
12HV615	PIC14	400	40-7F	
12F629	PIC14	3FF	20-5F	80
12F635	PIC14	400	40-7F	80
12C671	PIC14	3FF	20-7F,A0-BF	
12C672	PIC14	7FF	20-7F,A0-BF	
12CE673	PIC14	3FF	20-7F,A0-BF	
12CE674	PIC14	7FF	20-7F,A0-BF	
12F675	PIC14	3FF	20-5F	80
12F675F	PIC14	3FF	20-5F	80
12F675H	PIC14	3FF	20-5F	80

continued...

Table C.1: Devices supported by HI-TECH C PRO for the PIC10/12/16 MCU Family

DEVICE	ARCH	ROMSIZE	RAMBANK	EEPROMSIZE
12F675K	PIC14	3FF	20-5F	80
12F683	PIC14	800	20-7F,A0-BF	100
14000	PIC14	FC0	20-7F,A0-FF	
16C432	PIC14	800	20-7F,A0-BF	
16C433	PIC14	800	20-7F,A0-BF	
16C554	PIC14	200	20-6F	
16C554A	PIC14	200	20-6F	
16C556	PIC14	400	20-6F	
16C556A	PIC14	400	20-6F	
16C557	PIC14	800	20-7F,A0-BF	
16C558	PIC14	800	20-7F,A0-BF	
16C558A	PIC14	800	20-7F,A0-BF	
16C61	PIC14	400	0C-2F	
16C62	PIC14	800	20-7F,A0-BF	
16C62A	PIC14	800	20-7F,A0-BF	
16C62B	PIC14	800	20-7F,A0-BF	
16CR62	PIC14	800	20-7F,A0-BF	
16C63	PIC14	1000	20-7F,A0-FF	
16C63A	PIC14	1000	20-7F,A0-FF	
16CR63	PIC14	1000	20-7F,A0-FF	
16C64	PIC14	800	20-7F,A0-BF	
16C64A	PIC14	800	20-7F,A0-BF	
16CR64	PIC14	800	20-7F,A0-BF	
16C65	PIC14	1000	20-7F,A0-FF	
16CR65	PIC14	1000	20-7F,A0-FF	
16C65A	PIC14	1000	20-7F,A0-FF	
16C65B	PIC14	1000	20-7F,A0-FF	
16C66	PIC14	2000	20-7F,A0-EF,110-16F,190-1EF	
16C67	PIC14	2000	20-7F,A0-EF,110-16F,190-1EF	
16C620	PIC14	200	20-6F	
16C620A	PIC14	200	20-7F	
16CR620A	PIC14	200	20-7F	
16C621	PIC14	400	20-6F	
16C621A	PIC14	400	20-7F	
16C622	PIC14	800	20-7F,A0-BF	
16C622A	PIC14	800	20-7F,A0-BF	
16CE623	PIC14	200	20-7F	
16CE624	PIC14	400	20-7F	
16CE625	PIC14	800	20-7F,A0-BF	
16F610	PIC14	400	40-7F	
16HV610	PIC14	400	40-7F	
16F616	PIC14	800	A0-BF	
16HV616	PIC14	800	A0-BF	
16F630	PIC14	3FF	20-5F	80
16F631	PIC14	400	40-7F	80

continued...

Table C.1: Devices supported by HI-TECH C PRO for the PIC10/12/16 MCU Family

DEVICE	ARCH	ROMSIZE	RAMBANK	EEPROMSIZE
16F636	PIC14	800	20-7F,A0-BF	100
16F639	PIC14	800	20-7F,A0-BF	100
16C641	PIC14	800	20-7F,A0-BF	
16C642	PIC14	1000	20-7F,A0-EF	
16C661	PIC14	800	20-7F,A0-BF	
16C662	PIC14	1000	20-7F,A0-EF	
16F676	PIC14	3FF	20-5F	80
16F677	PIC14	800	20-7F,A0-BF	100
16F684	PIC14	800	20-7F,A0-BF	100
16F685	PIC14	1000	20-7F,A0-EF,120-16F	100
16F687	PIC14	800	20-7F,A0-BF	100
16F688	PIC14	1000	20-7F,A0-EF,120-16F	100
16F689	PIC14	1000	20-7F,A0-EF,120-16F	100
16F690	PIC14	1000	20-7F,A0-EF,120-16F	100
16C710	PIC14	200	0C-2F	
16C71	PIC14	400	0C-2F	
16C711	PIC14	400	0C-4F	
16C712	PIC14	400	20-7F,A0-BF	
16C715	PIC14	800	20-7F,A0-BF	
16C716	PIC14	800	20-7F,A0-BF	
16C717	PIC14	800	20-7F,A0-EF,120-16F	
16C72	PIC14	800	20-7F,A0-BF	
16C72A	PIC14	800	20-7F,A0-BF	
16CR72	PIC14	800	20-7F,A0-BF	
16F72	PIC14	800	20-7F,A0-BF	
16C73	PIC14	1000	20-7F,A0-FF	
16F73	PIC14	1000	20-7F,A0-FF	
16F722	PIC14	800	20-7F,A0-BF	
16LF722	PIC14	800	20-7F,A0-BF	
16F723	PIC14	1000	20-7F,A0-EF,120-12F	
16LF723	PIC14	1000	20-7F,A0-EF,120-12F	
16F724	PIC14	1000	20-7F,A0-EF,120-12F	
16LF724	PIC14	1000	20-7F,A0-EF,120-12F	
16F726	PIC14	2000	20-7F,A0-EF,110-16F,190-1EF	
16LF726	PIC14	2000	20-7F,A0-EF,110-16F,190-1EF	
16F727	PIC14	2000	20-7F,A0-EF,110-16F,190-1EF	
16LF727	PIC14	2000	20-7F,A0-EF,110-16F,190-1EF	
16F737	PIC14	1000	20-7F,A0-EF,110-16F,190-1EF	
16C73A	PIC14	1000	20-7F,A0-FF	
16C73B	PIC14	1000	20-7F,A0-FF	
16C74	PIC14	1000	20-7F,A0-FF	
16F74	PIC14	1000	20-7F,A0-FF	
16F747	PIC14	1000	20-7F,A0-EF,110-16F,190-1EF	
16C74A	PIC14	1000	20-7F,A0-FF	
16C74B	PIC14	1000	20-7F,A0-FF	

continued...

Table C.1: Devices supported by HI-TECH C PRO for the PIC10/12/16 MCU Family

DEVICE	ARCH	ROMSIZE	RAMBANK	EEPROMSIZE
16LC74B	PIC14	1000	20-7F,A0-FF	
16C76	PIC14	2000	20-7F,A0-EF,110-16F,190-1EF	
16F76	PIC14	2000	20-7F,A0-EF,110-16F,190-1EF	
16F767	PIC14	2000	20-7F,A0-EF,110-16F,190-1EF	
16C77	PIC14	2000	20-7F,A0-EF,110-16F,190-1EF	
16F77	PIC14	2000	20-7F,A0-EF,110-16F,190-1EF	
16F777	PIC14	2000	20-7F,A0-EF,110-16F,190-1EF	
16C770	PIC14	800	20-7F,A0-EF,120-16F	
16C771	PIC14	1000	20-7F,A0-EF,120-16F	
16C773	PIC14	1000	20-7F,A0-EF,120-16F	
16C774	PIC14	1000	20-7F,A0-EF,120-16F	
16C745	PIC14	2000	20-7F,A0-EF,120-16F	
16C765	PIC14	2000	20-7F,A0-EF,120-16F	
16C781	PIC14	400	20-7F,A0-BF	
16C782	PIC14	800	20-7F,A0-BF	
16F785	PIC14	800	20-7F,A0-BF	100
16HV785	PIC14	800	20-7F,A0-BF	100
16F818	PIC14	400	20-7F,A0-BF	80
16F819	PIC14	800	20-7F,A0-EF,120-16F	100
16F83	PIC14	200	0C-2F	40
16CR83	PIC14	200	0C-2F	40
16C84	PIC14	400	0C-2F	40
16F84	PIC14	400	0C-4F	40
16F84A	PIC14	400	0C-4F	40
16CR84	PIC14	400	0C-4F	40
16F627	PIC14	400	20-7F,A0-EF,120-14F	80
16F627A	PIC14	400	20-7F,A0-EF,120-14F	80
16F628	PIC14	800	20-7F,A0-EF,120-14F	80
16F628A	PIC14	800	20-7F,A0-EF,120-14F	80
16F648A	PIC14	1000	20-7F,A0-EF,120-16F	100
16F716	PIC14	800	20-7F,A0-BF	
16F87	PIC14	1000	20-7F,A0-EF,110-16F,190-1EF	100
16F870	PIC14	800	20-7F,A0-BF	40
16F871	PIC14	800	20-7F,A0-BF	40
16F872	PIC14	800	20-7F,A0-BF	40
16F873	PIC14	1000	20-7F,A0-FF	80
16F873A	PIC14	1000	20-7F,A0-FF	80
16F874	PIC14	1000	20-7F,A0-FF	80
16F874A	PIC14	1000	20-7F,A0-FF	80
16F876	PIC14	2000	20-7F,A0-EF,110-16F,190-1EF	100
16F876A	PIC14	2000	20-7F,A0-EF,110-16F,190-1EF	100
16F877	PIC14	2000	20-7F,A0-EF,110-16F,190-1EF	100
16F877A	PIC14	2000	20-7F,A0-EF,110-16F,190-1EF	100
16F88	PIC14	1000	20-7F,A0-EF,110-16F,190-1EF	100
16F882	PIC14	800	20-7F,A0-BF	80

continued...

Table C.1: Devices supported by HI-TECH C PRO for the PIC10/12/16 MCU Family

DEVICE	ARCH	ROMSIZE	RAMBANK	EEPROMSIZE
16F883	PIC14	1000	20-7F,A0-EF,120-16F	100
16F884	PIC14	1000	20-7F,A0-EF,120-16F	100
16F886	PIC14	2000	20-7F,A0-EF,110-16F,190-1EF	100
16F887	PIC14	2000	20-7F,A0-EF,110-16F,190-1EF	100
16F913	PIC14	1000	20-7F,A0-EF,120-16F	100
16F914	PIC14	1000	20-7F,A0-EF,120-16F	100
16F916	PIC14	2000	20-7F,A0-EF,120-16F,190-1EF	100
16F917	PIC14	2000	20-7F,A0-EF,120-16F,190-1EF	100
16C923	PIC14	1000	20-7F,A0-EF	
16C924	PIC14	1000	20-7F,A0-EF	
16C925	PIC14	1000	20-7F,A0-EF	
16C926	PIC14	2000	20-7F,A0-EF,120-16F,1A0-1EF	
16F946	PIC14	2000	20-7F,A0-EF,120-16F,1A0-1EF	100