C Programming

© Copyright Brian Brown, 1984-2000. All rights reserved.

Notes | Tests | Home Page

Topic Areas

- Introduction
- Declaring Variables
- Preprocessor Statements
- Arithmetic Operators
- Programming Style
- Keyboard Input
- Relational Operators
- For and while loops
- If, if else, relational operators
- Switch/case
- String and character handling
- Data validation examples
- Conditional expression operator
- Arrays
- Functions
- Additional assignment operator
- Sample programs so far
- Handling user input and Validation
- Formatters for printf() and scanf(), bit operations
- Structures
- Data conversion with itoa() and atoi()
- Files
- Pointers
- Linked Lists
- Dynamic Memory Allocation
- Preprocessor Statements, Macros, Conditional Compilation, typedef
- Eumerated Data Types
- Unions

- Register based variables, null statements and strings
- Command Line Arguments
- Pointers to functions
- Formatters for characters and strings
- System Calls
- Suggested solutions to all problems
- Advanced C, hardware accessing, longjump/ctrl break, tsr's, comms
- Interactive tests

This courseware is subject to copyright and may not be reproduced or copied without the written permission of the author. You may not redistribute this courseware without permission. If you are an educator, you may reference this material for use by your students, and if you purchase the CD, may host the files locally on your own network and print them out for student use or reference.

© Copyright Brian Brown, 1984-2000. All rights reserved.

C Programming

SYSTEM CALLS

Calls may be made to the Operating System to execute standard OPsys calls, eg,

```
#include <process.h>
main()  /* SYS.C    */
{
        char *command = "dir";

        system( "cls" );
        system( command );
}
```

Do not use this method to invoke other programs. Functions like exec() and spawn() are used for this.

©Copyright B Brown. 1984-2000. All rights reserved.

Advanced C, part 2 of 3

Copyright Brian Brown, 1986-2000. All rights reserved.

INDEX Previous Next

Comprehensive listing of interrupts and hardware details.

CONTENTS OF PART TWO

- Video Screen Displays
- Graphic Routines
- Developing a fast pixel set routine
- Assembly language interfacing
- Assembly language parameter passing
- Keyboard Bios Calls
- Hardware Equipment Determination Calls
- Memory Size Determination
- RS232 Bios Calls
- Printer Bios Calls

VIDEO SCREEN DISPLAYS

Monochrome The monochrome screen is arranged as 80 characters (columns) by 25 lines (rows). The starting address in memory where the display buffer (4kbytes) starts is B000:0000. There is only one display page, and each character is stored followed by its attribute in the next byte, eg,

```
        Address         Row,Column
        B000:0000       Character 0,0
        B000:0001       Attribute 0,0
        B000:0002       Character 0,1
        B000:0003       Attribute 0,1
```

The attribute byte is made up as follows,

```
        Bit
        7, BL = Blink
        6 5 4, Background
        3, I = Intensity or Highlight
        2 1 0, Foreground
        Back  Foregnd
         000  000  None
         000  001  Underline
         000  111  Normal video
         111  000  Reverse video
```

The offset of a character at a given row, column position is specified by the formula,

```
        offset = (( row * 0x50 + column ) * 2 )
```

Color Graphics Adapter
This adapter supports the following display modes, using a 16kbyte buffer starting at B800:0000

```
Text modes 40.25 BW/Color 80.25 BW/Color
Graphic modes 320.200 BW/4 Color 640.200 BW
```

In text mode, the adapter uses more than one display page, but only one page is active at any one time. The user may set any page to be the active one. The fourty column modes support 8 display pages (0-7), whilst the eighty column modes support 4 display pages (0-3).

You may write to a page which is not currently active, then use the function call setpage() to switch to that page. Using direct memory addressing on the active page will result in snow (white lines) due to memory contention problems between the display controller and the central processor trying to use the display RAM at the same time.

The formula to derive the memory offset of a character at a given row/column position is,

```
offset = (( row * 0x50 + column ) * 2 ) + ( pagenum * 0x1000 )
```

The following program illustrates some of these concepts,

```
#include <dos.h> /* TITLE 80.25 Color adapter */
#include <stdio.h>
union REGS regs;

void setpage( unsigned int pagenum ) {
        regs.h.ah = 5; regs.h.al = pagenum; int86( 0x10, &regs, &regs );
}

main() {
        int x, y, offset;
        char attr, ch;
        char far *scrn = ( char far * ) 0xB8000000;
        char *message = "This was written direct to page zero whilst page one
was being displayed.";
        /* set video mode to 80.25 color */
        regs.h.ah = 0; regs.h.al = 3; int86( 0x10, &regs, &regs );
        setpage(0); /* set display page to 0 */
        printf("This is page number 0\n"); getchar();
        setpage(1); /* set display page to 1 */
        printf("This is page number 1\n");
        /* Now write direct to screen 0 */
        x = 0; y = 1; attr = 0x82; /* column 0, row 1, green blinking */
        offset = (( y * 0x50 + x ) * 2 ) + ( 0 * 0x1000 );
        while ( *message ) {
                scrn[offset] = *message;
                ++offset;
                scrn[offset] = attr;
                ++offset;
                ++message;
        }
        getchar(); setpage(0); /* set display page to 0 */
}
```

There is a problem in writing text directly to the CGA screens. This causes **flicker** (snow). It is possible to incorporate a test which eliminates flicker. This involves only writing text to the screen during a horizontal retrace period.

The following program demonstrates the technique of direct video access, but waiting for the horizontal retrace to occur before writing a character.

```
main() {
        char far *scrn = (char far *) 0xb800000;
        register char attr = 04; /* red */
        register char byte = 'A';
        int loop, scrsize = 80 * 25 * 2;
        for( loop = 0; loop < scrsize; loop+= 2) {
                while( (inp(0x3da) & 01) == 0) ;
                while( (inp(0x3da) & 01) != 0) ;
                *scrn[loop] = byte;
                *scrn[loop+1] = attr;
        }
}
```

Medium Resolution Graphics mode (320.200)
Each pixel on the screen corresponds to two bits in the memory display buffer, which has values ranging from 0 to 3. Each scan line consists of eighty bytes, each byte specifying four pixels. All even row lines are stored in the even display bank, all odd row lines in the odd display bank (0x2000 apart). The display buffer begins at B800:0000, and the address of a particular row, column is found by,

```
offset = ((row & 1) * 0x2000) + (row / 2) * 0x50) + (column / 4)
```

Once the correct byte is located, the bits must be found. It is easiest to use a lookup table for this. In graphics modes there is no snow produced when directly updating the screen contents. The following portions of code illustrate some aspects of directly handling the video display.

```
bitset( row, column, color )
int row, column, color;
{
        int bitpos, mask[] = { 0x3f, 0xcf, 0xf3, 0xfc };
        char far *scrn = (char far *) 0xB8000000;
        unsigned int offset;
        color = color & 3;
        offset = ((row & 1) * 0x2000)+((row / 2) * 0x50) + (column / 4);
        bitpos = column & 3;
        color = color << 6;
        while( bitpos ) {
                color = color >> 2; bitpos;
        }
        scrn[offset] = scrn[offset] & mask[column & 3]; /* set bits off */
        scrn[offset] = scrn[offset] | color; /* set bits on/off */
}
```

Medium Resolution Graphics Color Modes
The ROM BIOS call int 10h allows the programmer to specify the color selections. The table below details how to use this call,

```
regs.h.ah = 0x0B;
regs.h.bh = palette;
```

```
        regs.h.bl = color;
```

If register bh contains 0, bl contains the background and border colors. If register bh contains 1, bl contains the palette being selected,

```
 Palette        Color_Value           Color
 0                0              Same as background
 0                1              Green
 0                2              Red
 0                3              Brown
 1                0              Same as background
 1                1              Cyan
 1                2              Magenta
 1                3              White
```

High Resolution Graphics mode (640.200)
Each pixel on the screen corresponds to a single bit in the video display buffer. Two colors are supported, on or off (1 or 0). The home position is 0,0 and bottom right is 199,639 The display is split up into two areas, called odd and even. Even lines are stored in the even area, odd lines are stored in the odd area. One horizontal line contains eighty bytes, each byte represents eight pixels. To find the byte address for a particular co-ordinate, the formula is

```
        offset = (( row & 1) * 0x2000 ) + ( row/w * 0x50 ) + ( column/8)
```

Having determined the offset byte (B800:offset), the following formula derives the bit position for the required pixel,

```
        bit number = 7 - ( column % 8 )
```

To determine the status of the particular pixel (ie, set or off) requires the use of a bit mask. The following portions of code demonstrate this,

```
        bittest( row, column) /* Title bittest(), return 1 if pixel set, else return
0 */
        int row, column;
        {
                static int mask[] = {0x80,0x40,0x20,0x10,0x08,0x04,0x02,0x01};
                int byte, bitpos;
                char far *scrn = (char far *) 0xB8000000;
                unsigned int offset;
                offset = ((row & 1) * 0x2000)+((row / 2) * 0x50) + (column / 8);
                byte = scrn[offset]; bitpos = 7 - ( column % 8 );
                return( ((byte & mask[bitpos]) > 0 ? 1 : 0) );
        }

        bitset( row, column, color ) /* TITLE bitset(), set bit at row, column to
color */
        int row, column, color;
        {
                static int mask[] = {0xfe,0xfd,0xfb,0xf7,0xef,0xdf,0xbf,0x7f};
                int bitpos;
                char far *scrn = (char far *) 0xB8000000;
                unsigned int offset;
                color = color & 1;
                offset = ((row & 1) * 0x2000)+((row / 2) * 0x50) + (column / 8);
```

```
            bitpos = 7 - ( column % 8 );
            color = color << bitpos;
            scrn[offset] = scrn[offset] & mask[bitpos]; /* set bit off first */
            scrn[offset] = scrn[offset] | color; /* set bit on or off */
        }
```

GRAPHIC ROUTINES

Creating Lines
Straight horizontal or vertical lines are relatively simple. However, diagonal lines are relatively complex to draw, as decisions need to be made as to where the dots must go, not where they should go according to a formula. General formula used in line calculations are,

```
        line start points x1, y1
        line end points x2, y2
```

Then the slope of the line (M) = (y2 - y1) / (x2 - x1)

and the Y intercept (B) = y1 - M * x1

An algorithm for plotting lines which is well known in computer circles is Bresenham's algorithm. I refer you to the book "Assembly language primer for the IBM-PC & XT: R Lafore, page 334". A diagonal line routine is present in the library.

Circles
Circles are easier to generate than lines. The basic equations involved in specifying a circle are, Given an angle in radians called A, and the centre of the circle as XC, YC and a radius of R, then

```
        X = XC + R * COS(A)
        Y = YC + R * SIN(A)
```

where X and Y are the two circle co-ordinates to plot next. Angles are always specified in radians. The circle function is present in the library.

Resolution Factors Since the video screen is not a 1:1 relationship, the x or y factor factor needs scaling by a specifed amount. The amount to scale by is depicted below.

Medium Resolution correction is 200/320 = .92

High Resolution correction is 200/640 = .46

WINDOWS
Windows are small viewing areas located on the display screen. Each application is run in a seperate window, and thus allows the user to view many events simultaneously. The following program illustrates the setting up of a window using the C language. Typically, the area where the new window is created would need to be saved for later recall, this can be done using the techniques illustrated under the section Memory accessing. A structure which contains an array of windows or pointers to windows can be used to implement multiple windows.

```
        /* WINDOWS.C A window demonstration program */
        /* Enter a backslash to quit program */
        #include <dos.h>
        #include <conio.h>
        #define ENTER_KEY 0xd
        #define BACK_SLASH '\\'
```

```
clear_window( tlr, tlc, brr, brc, attr )
unsigned int tlr, tlc, brr, brc, attr;
{
        union REGS regs;
        regs.h.ah = 6; regs.h.al = 0; regs.h.bh = attr;
        regs.h.ch = tlr; regs.h.cl = tlc; regs.h.dh = brr;
        regs.h.dl = brc; int86( 0x10, &regs, &regs );
}

setcursor( row, column )
int row, column;
{
        union REGS regs;
        regs.h.ah = 15; int86( 0x10, &regs, &regs ); /* get active page */
        regs.h.ah = 2; regs.h.dh = row;
        regs.h.dl = column; int86( 0x10, &regs, &regs );
}

scrollup( tlr, tlc, brr, brc, attr )
unsigned int tlr, tlc, brr, brc, attr;
{
        union REGS regs;
        regs.h.ah = 6; regs.h.al = 1; regs.h.bh = attr;
        regs.h.ch = tlr; regs.h.cl = tlc; regs.h.dh = brr;
        regs.h.dl = brc; int86( 0x10, &regs, &regs );
}

main()
{
        int attr, tlc, tlr, brc, brr, column, row;
        union REGS regs;
        char ch;
        printf("Co-ordinates top-left (row,column) ");
        scanf ("%d,%d", &tlr, &tlc);
        printf("Co-ordinates bottom-right (row,column) ");
        scanf ("%d,%d", &brr, &brc);
        printf("Enter attribute for window ");
        scanf ("%d", &attr);
        clear_window( tlr, tlc, brr, brc, attr );
        column = tlc; row = tlr;
        loop:
                setcursor( row, column);
                ch = getche();
                if( ch == ENTER_KEY) {
                        ++row; column = tlc;
                }
                else if( ch == BACK_SLASH) {
                        exit(0);
                }
                else
                        ++column;
                if ( column > brc ) {
                        column = tlc; ++row;
                }
                if( row > brr ) {
```

```
                                scrollup( tlr, tlc, brr, brc, attr );
                                row = brr;
                }
                goto loop;
        }
```

A FASTER PSET ROUTINE

The problem with the wdot() and pset() routines so far is that they are slow! For each pixel set, an interrupt must be generated, parameters passed and recieved. The overhead is therefore quite high, and a faster way must be found. A program is thus needed which translates the pixel row/column numbers into an absolute address which can be accessed via a FAR type pointer as outlined in the previous section Accessing Memory. The next code segment illustrates such a technique.

```
        #include <dos.h> /* TITLE: CPSET.C */
        char far *screen = (char far *) 0xb8000000; /* Video RAM screen */
        char far *parm = (char far *) 0x00400000; /* ROM BIOS DATA SEG */

        cpset( x, y, color ) /* A fast PSET() routine! */
        int x, y, color;
        {
                /* defaults are for 640x200 graphics mode */
                int shift=3, max=7, mask=0x7f, rotate=1, temp=1, bank, mode=0xC0;
                if( parm[0x49] == 4 ) /* 320.200 Color mode */
                {
                        shift=2; max = 3; mask = 0x3f; rotate = 2; temp = 3, mode =
0x80;
                }
                bank = y & 1; /* get bit zero */
                y = ( y & 0xfe ) * 40; /* adjust address to point to line*/
                if( bank ) /* select odd or even bank */
                        y += 0x2000;
                y += ( x >> shift ); /* add columns to address */
                x = x & max; /* select column bits for position in byte */
                color = color & temp; /* valid color ranges only */
                if( parm[0x49] == 4 )
                        color = color << 6; /* shift color bits into place */
                else
                        color = color << 7; /* this for 640.200 BW mode */
                while( x ) {
                        mask = ( mask >> rotate ) + mode; /* rotate bits to required
*/
                        color = color >> rotate; /* position in both mask & color */
                        x++;
                }
                screen[y] = screen[y] & mask; /* erase previous color */
                screen[y] = screen[y] | color; /* insert new color */
        }
```

Dedicated routines, ie, seperate functions for medium and high res as shown earlier give significant increases in speed at the expense of hardware dependance.

ASSEMBLY LANGUAGE INTERFACING

C programs may call assembly language routines (or vsvs). These notes concentrate on interfacing a machine code program which generates sound for a C program.

Assembly language routines which are called from C must preserve the 8088/86 BP, SI and DI registers. The recommended sequence for saving and restoring registers is,

```
entry: push bp
mov bp, sp
push di
push si
exit: pop si
pop di
mov sp, bp
pop bp
ret
```

The assembly language routine should be declared as **extern**al inside the C program. In writing the asm routine, it should be declared as public and the routine name should be preceeded by an underscore '_' character. The C program, however, does not use the underscore when calling the asm routine. The asm routine should also be declared as a PROC FAR type, in order for the linker to function correctly. The memory model used must be large, else the reference to FAR procedures will generate an error.

The following code illustrates how this is all done by way of a practical example.

```
;ASOUND.ASM Makes machine gun sound, firing a number of shots
;This is a stand-alone assembly language program!
;*******************************************************
stck segment stack ;define stack segment
db 20 dup ('stack ')
stck ends
;*******************************************************
code segment ;define code segment
main proc far ;main part of program
assume cs:code,ds:code
;set up stack for return to DOS
start: push ds ;save old data segment
sub ax,ax ;put zero in AX
push ax ;save it on stack
mov cx,20d ;set number of shots
new_shot:
push cx ;save count
call shoot ;sound of shot
mov cx,400h ;set up silent delay
silent:loop silent ;silent delay
pop cx ;get shots count back
loop new_shot ;loop till all shots done
ret ;return to DOS
main endp ;end of main part of program
;
;SUBROUTINE to make brief noise
shoot proc near
mov dx,140h ;initial value of wait
mov bx,20h ;set count
```

```
in al,61h ;get port 61h
and al,11111100b ;AND off bits 0 and 1
sound: xor al,2 ;toggle bit 1 into al
out 61h,al ;output to port 61
add dx,9248h ;add random bit pattern
mov cl,3 ;set to rotate 3 bits
ror dx,cl ;rotate it
mov cx,dx ;put in CX
and cx,1ffh ;mask off upper 7 bits
or cx,10 ;ensure not too short
wait: loop wait ;wait
dec bx ;done enough
jnz sound ;jump if not yet done
;turn off sound
and al,11111100b ;AND off bits 0 and 1
out 61h,al ;turn off bits 0 and 1
ret ;return from subroutine
shoot endp
;
code ends ;end of code segment
end start ;end assembly
```

The above program, called **ASOUND.ASM**, is a stand-alone machine code program. In order to interface this to a C program as a function which accepts the number of shots fired, the following changes are made.

```
;SOUND.ASMMakes machine gun sound, firing a number of shots
NAME SOUND
PUBLIC _bang
;
stck segment stack
db 200 dup ('stack ')
stck ends
;
BUZZ segment byte PUBLIC 'CODE' ;segment name
assume cs:BUZZ,ds:BUZZ
_bang PROC FAR ;main part of program
push bp ;save current bp
mov bp,sp ;get stack pointer
push di ;save register variables from c
push si
mov cx,[bp+6] ;get passed int value into CX
new_shot:
push cx ;save count
call shoot ;sound of shot
mov cx,400h ;set up silent delay
silent:loop silent ;silent delay
pop cx ;get shots count back
loop new_shot ;loop till all shots done
;return to DOS
pop si ;restore register varaibles
pop di ;restore register variables
mov sp,bp ;recover stack pointer
pop bp
ret ;back to C program
```

```
_bang endP ;end of main part of program
;
;SUBROUTINE to make brief noise
shoot PROC NEAR
mov dx,140h ;initial value of wait
mov bx,20h ;set count
in al,61h ;get port 61h
and al,11111100b ;AND off bits 0 and 1
sound: xor al,2 ;toggle bit 1 into al
out 61h,al ;output to port 61
add dx,9248h ;add random bit pattern
mov cl,3 ;set to rotate 3 bits
ror dx,cl ;rotate it
mov cx,dx ;put in CX
and cx,1ffh ;mask off upper 7 bits
or cx,10 ;ensure not too short
wait: loop wait ;wait
dec bx ;done enough
jnz sound ;jump if not yet done
;turn off sound
and al,11111100b ;AND off bits 0 and 1
out 61h,al ;turn off bits 0 and 1
ret ;return from subroutine
shoot endP
BUZZ ends
END
```

The above shows a listing for **SOUND.ASM**, the converted routine from ASOUND.ASM, which will interface to a C program. Note that **_bang()** has been declared as PUBLIC and a procedure of type FAR. It is assembled using MASM V4.00 using the following command line,

```
A>MASM /MX SOUND;
```

This creates a file called **SOUND.OBJ** which will be linked with the C object code shortly. The purpose of the switch /MX is to preserve case sensitivity for public names.

The C Compiler uses the DI and SI registers for variables declared register based, so these are saved by the asm routine. If the direction flag is altered by the machine code program then the instruction cld should be executed before returning. C programs pass parameters on the stack. These are accessed as follows,

```
                                NEAR CALL        FAR CALL
1st argument Single Word        [ bp + 4 ]       [ bp + 6 ]
Next arg, Single Word           [ bp + 6 ]       [ bp + 8 ]
Double Word                     [ bp + 8 ]       [ bp + 10 ]
```

Single words occupy two bytes, double words four bytes.

The C program which calls the assembly language routine _bang() is,

```
extern far bang(); /* CSOUND.C */
main() {
        int fires = 0, sam = 0;
        printf("Enter in the number of times you wish to fire the gun:\n");
        scanf("%d", &fires);
```

```
        for( ; sam < fires; sam++ ) {
                printf("Entering bang to fire the gun %d times.\n", sam);
                bang( fires );
        }
}
```

At linking time, the file sound.obj is added to csound.obj, ie,

```
; for Microsoft C Compiler
; LINK csound.obj+sound.obj
; for TurboC Compiler
; tlink c0l csound sound, csound, csound, cl
```

NOTE: The assembly language routine _bang() is declared as an external function of type FAR. When referenced in the C program, the asm routine _bang() is done so without the leading underscore character.

RETURN VALUES FROM ASM ROUTINES

```
Return Data Type                Register(s) used
Characters                        AX
Short, Unsigned                   AX
Integers                          AX
Long Integers                     DX = High, AX = Low
Unsigned Long                     DX = High, AX = Low
Structure or Union                Address in AX
Float or double type              Address in AX
Near Pointer                      Offset in AX
Far Pointer                       Segment = DX, Offset = AX
```

The following section deals with how parameters are passed between C functions at the machine code level.

PARAMETER PASSING AND ASSEMBLY LANGUAGE

ACCESSING THE STACK FRAME INSIDE A MODULE Lets look at how a module handles the stack frame. Because each module will use the BP register to access any parameters, its first chore is to save the contents of BP.

```
push bp
```

It will then transfer the address of SP into BP, so that BP now points to the top of the stack.

```
mov bp,sp
```

thus the first two instructions in a module will be the combination,

```
push bp
mov bp,sp
```

ALLOCATION OF LOCAL STORAGE INSIDE A MODULE Local variables are allocated onto the stack using a

```
sub sp, n
```

instruction. This decrements the stack pointer by the number of bytes specified by n. For example, a C program might contain the declaration

```
auto int i;
```

as defining a local variable called **i**. Variables of type auto are created on the stack, and assuming an integer occupies two bytes, then the above declaration equates to the machine code instruction

```
sub sp, 2
```

Pictorially, the stack frame looks like,

```
|----------|
|  ihigh   | < SP
|----------|
|  ilow    |
|----------|
|  BPhigh  | < BP
|----------|
|  BPlow   |
|----------|
```

The local variable i can be accessed using SS:BP - 2, so the C statement,

```
i = 24; is equivalent to mov [bp - 2], 18
```

Note that twenty-four decimal is eighteen hexadecimal.

DEALLOCATION OF LOCAL VARIABLES WHEN THE MODULE TERMINATES When the module terminates, it must deallocate the space it allocated for the variable i on the stack. Referring to the above diagram, it can be seen that BP still holds the top of the stack as it was when the module was first entered. BP has been used for two purposes,

```
- to access parameters relative to it
- to remember where SP was upon entry to the module
```

The deallocation of any local variables (in our case the variable i) will occur with the following code sequence,

```
mov sp, bp ;this recovers SP, deallocating i
pop bp ;SP now is the same as on entry to module
```

THE PASSING OF PARAMETERS TO A MODULE Consider the following function call in a C program.

```
add_two( 10, 20 );
```

C pushes parameters (the values 10 and 20) right to left, thus the sequence of statements which implement this are,

```
push ax ;assume ax contains 2nd parameter, ie, integer value 20
push cx ;assume cx contains 1st parameter, ie, integer value 10
call add_two
```

The stack frame now looks like,

```
|----------|
|  return  | < SP
|----------|
|  address |
|----------|
|    0A    | 1st parameter; integer value 10
|----------|
|    00    |
|----------|
|    14    | 2nd parameter; integer value 20
|----------|
|    00    |
|----------|
```

Remembering that the first two statements of module add_two() are,

```
add_two: push bp
mov bp, sp
```

The stack frame now looks like (after those first two instructions inside add_two)

```
|----------|
|  BP high | <- BP <- SP
|----------|
|  BP low  |
|----------|
|  return  |
|----------|
|  address |
|----------|
|    0A    | 1st parameter; integer value 10
|----------|
|    00    |
|----------|
|    14    | 2nd parameter; integer value 20
|----------|
|    00    |
|----------|
```

ACCESSING OF PASSED PARAMETERS WITHIN THE CALLED MODULE It should be clear that the passed parameters to module add_two() are accessed relative to BP, with the 1st parameter residing at [BP + 4], and the 2nd parameter residing at [BP + 6].

DEALLOCATION OF PASSED PARAMETERS The two parameters passed in the call to module add_two() were pushed onto the stack frame before the module was called. Upon return from the module, they are still on the stack frame, so now they must be deallocated. The instruction which does this is, add sp, 4 where SP is adjusted upwards four bytes (ie, past the two integers).

EXAMPLE C PROGRAM AND CORRESPONDING ASSEMBLER CODE Consider the following C program and equivalent machine code instructions.

```
add_two: push bp                  add_two( numb1, numb2 )
mov bp, sp                        int numb1, numb2;
```

```
add sp, 2                         {       auto int result;
mov ax, [bp + 4]                          result = numb1 + numb2;
add ax, [bp + 6]
mov [bp - 2], ax
mov sp, bp                        }
pop bp
ret
main: push bp                     main()
mov bp, sp                        {
sub sp, 4                                 int num1, num2;
mov [bp - 2], 0A                          num1 = 10;
mov [bp - 4], 14                          num2 = 20;
push wptr [bp - 4]                        add_two( 10, 20 );
push wptr [bp - 2]
call add_two
add sp, 4                         }
mov sp, bp
pop bp
ret
```

KEYBOARD BIOS CALLS

INT 16H This interrupt in the ROM BIOS provides for minimal character transfer from the keyboard. It is entered by first specifying the desired task to perform in the AH register.

```
AH = 0 Get Key
        Returns with AH = scan code
        AL = ascii char, 0 = non-ascii, ie Func key
AH = 1 Get status
        Returns with zflag = 0, valid key in queue
        = 1, no key in queue
AH = scan code
        AL = ascii char, 0 = non-ascii, ie, Func key
AH = 2 Get shift status
        Returns with AL = 7 Right shift 1 = pressed
        6 Left shift
        5 Ctrl
        4 Alt
        3 Scroll Lock 1 = on
        2 Num Lock
        1 Caps Lock
        0 Ins
```

Lets develop routines similar to those found in some libraries.

```
#include <dos.h>

int bioskey( cmd )
int cmd;
{
        union REGS regs;
        regs.h.ah = cmd;
```

```
        int86( 0x14, &regs, &regs );
        return( regs.x.ax );
}

int kbhit()
{
        union REGS regs;
        regs.h.ah = 1;
        int86( 0x16, &regs, &regs );
        return( regs.x.cflags & 0x0040 ); /* return Z flag only */
}
```

EQUIPMENT BIOS CALLS

INT 11H This interrupt is used to determine the hardware attached to the computer system. It returns a value in register AX, which is comprised as follows,

```
Bits     Description
15,14    Number of printers
13       Not used
12       Game I/O attached
11,10,9  Number of RS232 cards attached
8        Not used
7,6      Number of disk drives 00=1, 01=2, 10=3, 11=4
5,4      Initial video mode 00=40, 01=80, 11=Mono
3,2      Ram Size
1        Co-Processor
0        IPL from disk 1=disk, 0=None
Bits 0 - 7 correspond to SW1 settings on motherboard (port 60h)
```

Lets demonstrate one use of this. First, lets create a similar function to that provided by the TurboC compiler.

```
#include <dos.h>
int biosequip()
{
        union REGS regs;
        int86( 0x11, &regs, &regs );
        return( regs.x.ax );
}
```

Now, using this, lets develop a function to test if the machine has a serial card connected.

```
int is_serial_card()
{
        if( (biosequip() & 0xE00) == 0 )
                return 0;
        else
                return 1; /* Serial card is present */
}
```

MEMORY SIZE BIOS CALLS

INT 12H This interrupt returns the number of 1kbyte memory blocks present in the system. This value is returned in the AX register. Lets develop routines similar to those in TurboC.

```
        #include <dos.h>

        int biosmemory()
        {
                union REGS regs;
                int86( 0x12, &regs, &regs );
                return( regs.x.ax );
        }
```

or, what about this version, utilising TurboC's register variables.

```
        int biosmemory()
        {
                geninterrupt( 0x12 );
                return _AX;
        }
```

RS232 BIOS CALLS

INT 14H This interrupt provides support for RS232 communications. The AH register value, on entry, determines the function to be performed.

```
        AH = 0 reset port, DX = 0 = com1
                return value in AL
                        7,6,5 Baud rate 000 = 110 100 = 1200
                        001 = 150 101 = 2400
                        010 = 300 110 = 4800
                        011 = 600 111 = 9600
                        4,3 Parity (00,10=off) (01=odd, 11=even)
                        2 Stops (0=One, 1=Two stops)
                        1,0 Word Size (10=7,11=8)
        AH = 1 xmit a character, character in AL
        AH = 2 recieve a character, returns character in AL
        AH = 3 return status
```

Now, lets develop routines in C to program the rs232 card using the int 14 BIOS call, ie, the bioscom() function in TurboC.

```
        #include <dos.h>
        int bioscom( cmd, byte, port )
        int cmd;
        char byte;
        int port;
        {
                union REGS regs;
                regs.x.dx = port;
                regs.h.ah = cmd;
```

```
        regs.h.al = byte;
        int86( 0x14, &regs, &regs );
        return( regs.x.ax );
}
```

Now, lets develop routines to initialise the specified comport, and to transmit and recieve characters, without resorting to using int 14h. These types of routines directly program the rs232 card, thus are ideal for embedded applications, ie, ROMMABLE code.

```
/*- Initiliase the RS232 serial card -*/
#define INP inportb
#define OUTP outportb
/* Defines for RS232 communications */
#define DLL 0 /* divisor latch low byte */
#define DLH 1 /* divisor latch high byte */
#define THR 0 /* transmit hold register */
#define RBR 0 /* recieve buffer register */
#define IER 1 /* interrupt enable register */
#define LCR 3 /* line control register */
#define MCR 4 /* modem control register */
#define LSR 5 /* line status register */
#define MSR 6 /* modem status register */
#define RTS 0x02 /* request to send */
#define CTS 0x10 /* clear to send */
#define DTR 0x01 /* data terminal ready */
#define DSR 0x20 /* data set ready */
#define RBF 0x01 /* bit 0 of LSR, rec buf full */
#define THRE 0x20 /* bit 5 of LSR, trans reg 0 */
#define DISINT 0x00 /* disable interrupts in IER */
#define ABRG 0x83 /* access baud rate generator */
/**/

void rs232_init( com_port, baud rate, parity, stops, word_size )
int com_port, baud_rate, word_size, stops;
char *parity;
{
        unsigned int divisorh, divisorl, format, acia[2];
        int far *bios = 0x00400000l;
        acia[0] = *bios; /* pick up address of com1 routine */
        acia[1] = *(bios + 1); /* pick up address of com2 routine */
        OUTP(acia[com_port] + IER, DISINT ); /* disable ints */
        OUTP(acia[com_port] + LCR, ABRG ); /* access baud rate gen*/
        switch( baud_rate ) {
                /* rem case 75, 110, 135, 150, 200, 1800, 19200 */
                case 300 : divisorh = 01; divisorl = 0x80; break;
                case 600 : divisorh = 00; divisorl = 0xc0; break;
                case 1200 : divisorh = 00; divisorl = 0x60; break;
                case 2400 : divisorh = 00; divisorl = 0x30; break;
                case 4800 : divisorh = 00; divisorl = 0x18; break;
                case 9600 : divisorh = 00; divisorl = 0x0c; break;
                default: printf("\nrs232_init: Error: Baud rate invalid.\n");
                        return -1;
        } /* end of switch */
        OUTP(acia[com_port] + DLL, divisorl );
```

```
        OUTP(acia[com_port] + DLH, divisorh );
        format = 0; /* This sets bit 6 and 7 to zero */
        if( (strcmp( parity, "E" ) == 0) || (strcmp( parity, "O" ) == 0) ) {
                format = format  0x28; /* set bit 3 and 5 */
                if( strcmp( parity, "E" ) == 0 )
                        format = format  0x10; /* set bit 4 */
        }
        if( stops == 2 )
                format = format  0x04;
        switch( word_size ) {
                case 8 : format = format  0x03; break;
                case 7 : format = format  0x02; break;
                case 6 : format = format  0x01; break;
                case 5 : break;
                default: printf("\nrs232_init: Unsupported word length.\n");
                        return -1;
        } /* end of switch */
        OUTP(acia[com_port] + LCR, format );
        return 0;
}

/* Transmit a single character to RS232 card -*/
void transmit( byte )
char byte;
{
        OUTP(acia[comport] + MCR, (RTS | DTR) ); /* assert RTS and DTR */
        while((INP(acia[comport] + LSR) & THRE)==0) /* trans reg empty? */
                ;
        OUTP(acia[comport] + THR, byte ); /* write character to THR */
        OUTP(acia[comport] + MCR, 0 );
}

/* Receive a single character from RS232 card */
char receive()  {
        char byte;
        OUTP(acia[comport] + MSR, (RTS | DTR) );
        while((INP(acia[comport]+LSR)&RBF)==0) /* has Data arrived? */
                ;
        OUTP(acia[comport] + MCR,0); /* stop all data */
        byte = INP(acia[comport] + RBR); /* get byte RBR */
        return( byte );
}
```

PRINTER SERVICES

INT 17H This interrupt provides support for the parallel printer. The AH register value, on entry, determines the function to be performed.

```
AH = 0 Write Character
        AL = character
        DX = printer number (0-2)
        Returns with AH = status code
                Bit 7 = printer not busy
```

```
                        6 = acknowledge
                        5 = out of paper
                        4 = printer selected
                        3 = I/O error
                        2,1 = unused
                        0 = time-out
        AH = 1 Initialise Printer
                DX = printer number (0-2)
                Returns with AH = status code
        AH = 2 Get Printer Status
                DX = printer number (0-2)
                Returns with AH = status code
```

Now lets develop a few routines which illustrate this,

```
        int biosprint( command, ch, printer )
        int command;
        char ch;
        int printer;
        {
                _AH = command;
                _DX = printer;
                _AL = ch;
                geninterrupt( 0x10 );
                _AX = _AX >> 8;
                return( _AX );
        }
```

Copyright Brian Brown, 1986-2000. All rights reserved.

INDEX Previous Next

Advanced C, part 3 of 3

INDEX Previous

Copyright Brian Brown, 1986-2000. All rights reserved.

Comprehensive listing of interrupts and hardware details.

CONTENTS OF PART THREE

- Floppy Disk Calls
- Sound Generation
- LongJump and ctrl-break handling
- Interrupt handling
- Embedded Code
- Make utility
- DOS Interrupts
- Interfacing to mouse.sys
- Interfacing to a joystick
- Video Library Routines

FLOPPY DISK SERVICES

INT 13H This interrupt provides for the management of the floppy disk drive and controller unit. The AH register value, on entry, determines the desired function.

```
        AH = 0 Reset Floppy Disk System No return value
        AH = 1 Get Status Returns AH as a status byte,
                Bit 7 = time-out
                6 = seek failure
                5 = controller error
                4 = CRC error
                3 = DMA overrun
                2 = sector not found
                1 = write-protected disk
                0 = illegal command
        AH = 2 Read Sector(s)
                AL = number of sectors to transfer (1-9)
                ES:BX = segment:offset of disk I/O buffer
                CH = track number (0-39)
                CL = sector number (1-9)
                DH = head number (0-1)
                DL = drive number (0-3)
                Returns on success cflag = clear
                                AH = 0
                                AL = number of disk sectors transferred
                        on failure cflag = set
                                AH = status byte
        AH = 3 Write Sector(s) same as for read sector(s)
```

```
        AH = 4 Verify Sector(s)
                AL = number of sectors to transfer (1-9)
                CH = track number (0-39)
                CL = sector number (1-9)
                DH = head number (0-1)
                DL = drive number (0-3)
                Returns on success cflag = clear
                                AH = 0
                                AL = number of disk sectors transferred
                        on failure cflag = set
                                AH = status byte
        AH = 5 Format Track
                ES:BX = segment:offset of address field list
                No return value
```

SOUND GENERATION

Port 43h provides access to the registers of port 42h. First store the magic value 0xb6 to port 43h, then load two 8 bit values into port 42h, which specify the frequency to generate. Once this is done, turning on bits 1 and 0 of port 61h will enable the circuitry and produce the tone. Summarising the steps, they are

1: Output 0xb6 to port 43h

2: Send each of the 8 bit values to port 42h

3: Enable bits 0 and 1 of port 61h

Step 1 is achieved by the C statement outportb(0x43, 0xb6);

Step 2 is achieved by converting the frequency into two eight bit values, then outputting them to port 42h. The Most Significant Byte is sent last.

```
        Frequency required to generate = 512 hertz
        16 bit value = 120000h / frequency
```

so, this is achieved by the C statements

```
        outportb(0x42,0)
        outportb(0x42,6)
```

Step 3 is achieved by the C statements

```
        byte = inportb(0x61);
        byte |= 3;
        outportb(0x61, byte);
```

Connecting this together into a function, it becomes,

```
        void tone_512() {
                char byte;
                outportb(0x43, 0xb6);
                outportb(0x42, 0);
```

```
                outportb(0x42, 6);
                byte = inportb(0x61);
                byte |= 3;
                outportb(0x61, byte);
        }
```

There follows two routines to generate sound using the timer chip. The first, beep(), sounds a note of 1000hz for a short duration. The second, note() allows you to specify the frequency and duration of the desired note.

```
        #include <dos.h>
        void beep() {
                int delay;
                _AL = 0xb6;
                outportb(0x43,_AL); /* write to timer mode register */
                _AX = 0x533; /* divisor for 1000hz */
                outportb(0x42,_AL); /* write LSB */
                _AL = _AH;
                outportb(0x42,_AL); /* write MSB */
                _AL = inportb(0x61); /* get current port setting */
                _AH = _AL; /* save it in _AH */
                _AL |= 3; /* turn speaker on */
                outportb(0x61,_AL);
                for(delay = 0; delay < 20000; delay++)
                        ;
                _AL = _AH;
                outportb(0x61,_AL); /* restore original settings */
        }

        int note( frequency, duration )
        unsigned int frequency, duration;
        {
                unsigned long delay;
                if( frequency > 5000 ) return 1;
                _AL = 0xb6;
                outportb(0x43,_AL); /* write to timer mode register */
                _AX = 0x120000L / frequency; /* calculate divisor required */
                outportb(0x42,_AL); /* write LSB */
                _AL = _AH;
                outportb(0x42,_AL); /* write MSB */
                _AL = inportb(0x61); /* get current port setting */
                _AL |= 3; /* turn speaker on */
                outportb(0x61,_AL);
                for(delay = 0L; delay < (long) duration * 45; delay++)
                        ;
                _AL = inportb(0x61); /* turn off sound */
                _AL &= 0xfc; /* set bits 1 and 0 off */
                outportb(0x61,_AL); /* restore original settings */
                return 0;
        }

        main() {
                unsigned int f;
                for(f = 100; f < 250; f += 10 ) {
                        note( f, (unsigned int) 1000 ); /* 1000 = 1 second */
```

```
                }
        }
```

LONGJUMP/SETJUMP/CTRL-BRK

The purpose of this section is to illustrate the techniques involved in taking over the control-break and control-c routines. We will show you how to enable and disable control-c checking. As well, features of the longjmp/setjmp routines will be demonstrated.

Control/Break

The routine which controls the detection of control-break resides in the ROM BIOS chip (int 16h), thus cannot be disabled. The keyboard interrupt routine, upon detecting a Ctrl-Break, generates an interrupt into DOS (type 1bh). It is thus possible to re-direct this DOS interrupt to your own routine.

Control/C

Ctrl-C is detected by DOS. This may be disabled or enabled using the setcbrk() in TurboC. The function ctrlbrk() allows redirection of the Ctrl-C interrupt (int 23h) to a particular function.

Lets build up some routines similar to those found in the TurboC library.

```
        #include  <dos.h>

        int setcbrk( value ) /* set control c checking, 0 = off, 1 = on */
        int value;
        {
                union REGS regs;
                regs.h.ah = 0x33;
                regs.h.al = 01;
                regs.h.dl = value;
                intdos( &regs, &regs );
        }

        int getcbrk() /* get control C status, 0 =0ff, 1 = on */
        {
                union REGS regs;
                regs.h.ah = 0x33;
                regs.h.al = 00;
                intdos( &regs, &regs );
                return( regs.h.dl & 1 );
        }
```

The following program illustrates the use of re-directing the ctrl-c interrupt 0x23 to a user supplied routine (TurboC only).

```
        #include <dos.h>

        int ctc() /* exits to DOS if return value is 0 */
        {
                static int times=0;
                printf("ctc is activated %d.\n", times);
                ++times;
                if(times >= 5) return(0);
                else return(1);
```

```
        }

        main() {
                int value;
                value = getcbrk();
                printf("Control-C checking is ");
                if( value )
                        printf("on");
                else
                        printf("off");
                printf(".\nRedirecting ctrl-c to user routine ctc.\n");
                ctrlbrk(ctc);
                for( ; ; )
                        printf("Press ctrl-c to exit (5)\n");
        }
```

LONGJMP and SETJMP

These routines allow processing to restart from a designated part of the program. Examples of this might be a menu driven system, which has many layers. A user, accessing a low level menu, by pressing ctrl-c, could immediately be placed into the highest level menu as though the package had just been restarted! Lets show how this is done by way of an example.

```
        #include <setjmp.h> /* for setjmp and longjmp */
        #include <stdio.h>

        jmp_buf jumper; /* for an image of the stack frame entries */

        void getkey() {
                printf("Press any key to continue.\n"); getchar();
        }

        int ctrlbreak() {
                printf("\n. Returning to main menu now.\n"); getkey();
                longjmp( jumper, 1 );
        }

        main() {
                ctrlbrk( ctrlbreak ); /* set up control-c handler routine */
                setjmp( jumper ); /* remember this as the entry point */
                for( ; ; ) { /* will return here when user press's ctrl-brk */
                        printf("Top menu.\nPress any key"); if( getchar()=='E')
exit(0);
                        for( ; ; ) {
                                printf("Menu 2.\nPress any key"); getchar();
                                for( ; ; ) {
                                        printf("Menu 3.\nPress any key"); getchar();
                                }
                        }
                }
        }
```

INTERRUPT ROUTINES

This section concentrates upon writing interrupt routines to replace those resident by either DOS or the ROM BIOS. By way of an illustration, we will show you how to take over the shift/prtscrn interrupt, which dumps the screen to the printer. This may be useful on a machine which does not have a printer. TurboC will be used to demonstrate this technique. Once this has been done, we will also show you how to modify it so that it stays resident in memory, rather than just lasting whilst the program lasts.

```
#include <dos.h>

void interrupt (*old_int5)();

void interrupt my_int5( unsigned bp, unsigned di, unsigned si,
unsigned ds, unsigned es, unsigned dx,
unsigned cx, unsigned bx, unsigned ax )
{
        /* normally, place nothing here, just a dummy routine */
        _AH = 0x0a;
        _AL = '5';
        _CX = 80;
        geninterrupt( 0x10 );
}

int ctrlbreak() {
        printf("\n. Returning to DOS now.\n");
        setvect( 5, old_int5 ); /* restore original vector */
        return( 0 );
}

main() {
        ctrlbrk( ctrlbreak ); /* set up control-c handler routine */
        old_int5 = getvect( 5 );
        printf("Resetting int_5 now.\n");
        setvect( 5, my_int5);
        for( ; ; )
                printf("Press ctrl-c to exit, shift-prtscrn to test.\n");
}
```

Be very careful about the use of DOS routines inside your interrupt routines. Calls to printf(), scanf() etc will probably result in a system crash. Now, lets present the above program as a terminate and stay resident program.

```
/* compiled in TurboC V1.0, using Large Memory Model */
#include <dos.h>

extern void far *_heapbase;

void interrupt my_int5( unsigned bp, unsigned di, unsigned si,
unsigned ds, unsigned es, unsigned dx,
unsigned cx, unsigned bx, unsigned ax )
{
}

main() {
        setvect( 5, my_int5);
```

```
        keep( 0, FP_SEG(_heapbase) - _psp );
}
```

Programs which Terminate and Stay Resident (TSR) are not simple. How-ever, there have been some good articles written recently concerning this.

```
Writing TSR's in TurboC : Al Stevens
(Computer Language, Feb 1988)
Converting TC programs to a TSR : M Young
(DOS Programmers Journal 1988, v6.2)
```

Now lets develop a program which is slighly more sophisticated. This program displays the time in the left top corner of the video screen.

```
#include <stdio.h> /* timer.c, (c) B Brown, 1988 */
#include <dos.h> /* TurboC V1.0, Large memory model */
#include <string.h>

extern void far *_heapbase;
static unsigned int TCSS;
static unsigned int TCSP;
static unsigned int TCDS;
static unsigned int OLDSS;
static unsigned int OLDSP;
static unsigned int OLDDS;
static int far *tbase = (int far *) 0x0000046cl;
static void interrupt (*oldtimer)();
static char buffer[20];
static int loop, xpos, ypos, vpage = 0;

static struct t {
        unsigned int sec, min, hor;
} tme;

void interrupt mytime( unsigned bp, unsigned di, unsigned si,
unsigned ds, unsigned es, unsigned dx,
unsigned cx, unsigned bx, unsigned ax )
{
        /* save old values of registers, get programs stack */
        disable();
        OLDSS = _SS;
        OLDSP = _SP;
        OLDDS = _DS;
        _DS = TCDS;
        _SS = TCSS;
        _SP = TCSP;
        /* get timer values */
        tme.hor = *(tbase + 1);
        tme.min = (*tbase / 18) / 60;
        tme.sec = (*tbase / 18) - (tme.min * 60);
        /* convert values to a character string */
        buffer[0] = (tme.hor / 10) + '0';
        buffer[1] = (tme.hor % 10) + '0';
```

```
        buffer[2] = ':';
        buffer[3] = ((tme.min / 10) | 0x30);
        buffer[4] = ((tme.min % 10) | 0x30) - 1;
        buffer[5] = ':';
        buffer[6] = ((tme.sec / 10) | 0x30);
        buffer[7] = (tme.sec % 10) + '0';
        buffer[8] = '\0';
        enable();
        /* save current cursor position */
        _AH = 3; _BH = vpage; geninterrupt(0x10); xpos = _DL; ypos = _DH;
        /* set cursor to row 0, column 0 */
        _AH = 2; _BH = vpage; _DX = 0; geninterrupt( 0x10 );
        /* print time on screen */
        for( loop = 0; loop < 8; loop++ ) {
                _AH = 0x0a; _AL = buffer[loop]; _BH = vpage;
                _CX = 1; geninterrupt( 0x10 );
                _AH = 2; _BH = vpage; _DX = loop + 1; geninterrupt(0x10);
        }
        /* restore original cursor position */
        _AH = 2; _BH = vpage; _DH = ypos, _DL = xpos; geninterrupt( 0x10 );
        /* chain to old timer interrupt */
        (*oldtimer)();
        /* restore register values, calling stack etc */
        _SS = OLDSS;
        _SP = OLDSP;
        _DS = OLDDS;
}

main() {
        disable();
        oldtimer = getvect( 0x1c ); /* get original vector */
        disable();
        TCSS = _SS; /* save segment values etc of programs stack */
        TCSP = _SP;
        TCDS = _DS;
        setvect( 0x1c, mytime ); /* hook into timer routine vector */
        enable();
        keep( 0, FP_SEG(_heapbase) - _psp ); /* tsr */
}
```

PRODUCING EMBEDDED CODE

Using the PC as a stand-alone system such as a data-logger, terminal etc, poses several problems. Generally, the software will be written to reside at a specific place in memory, usually in an EPROM. When the PC is turned on, this software is activated. This means that DOS is probably not present. If the software is written with any calls to DOS (examples being printf(), scanf() etc), then it will certainly crash.

Should the ROM BIOS chip be left on board, then calls to it via int xxh will probably work okay. This depends very much upon where the software is located in memory. As I see it, there are several options open. On a 640k RAM machine, RAM goes from 00000 to 9ffff, with the graphic cards going from a0000 to bffff. This leaves user ROM space from c0000 up to f4000.

Depending upon the ROM BIOS chip, some routines are not executed if you place code between c0000 to c7fff. These routines are probably initialisation of the keyboard queue, video display and disk drive controller (which may be important if you intend to use int 16h, int 10h and int 13h). Manufacturers of EGA cards, hard disk drives, lan cards etc usually place their code between c8000 to f4000.

On power up, as the ROM BIOS is being executed, it first checks for ROM chips between c0000 to c7fff, at every 2k boundary. If it finds one, it will leap to the ROMS entry address and execute the code there. Upon return (or if it doesn't find a ROM chip), it initialises the keyboard queue and video display, then checks for ROM between c8000 to f4000. If it finds a ROM here, it again calls it to execute the code.

If no ROM chips are found, the computer will attempt an int19h (Bootstrap Loader routine). If this is unsuccessful, an int18h instruction will be generated (a call to F6000, ie, BASIC ROM). If there are no BASIC ROM chips on board, it's likely that the system will perform a reset.

BASIC ROM resides from f4000 up, the entry point is f6000. The BIOS ROM resides from fe000 to fffff (normally an 8k EPROM, type 2764).

The format of User ROM chips residing between c0000 to f4000
If you decide to create a program which resides in this address space, then download it into an EPROM for placement on a board, its important to adhere to special provisions concerning the format of the initial 5 bytes of code. A ROM chip must be identified with the bytes 55 and AA in the first 2 locations, followed by a byte which represents the length of the program divided by 512, then followed by an instruction or jump to the entry routine (initialisation code, which sets up the segment register values, stack space etc).

The process of generating ROMMABLE code.
Rommable code is created by either specifying the absolute segment addresses using assembler (segment at 0c800h), or using a LOCATOR program which assigns addresses to the various segment groups once the program has been linked. This creates an absolute image file which can be downloaded into an EPROM programmer unit, which then programs the actual EPROM chip.

Other Considerations
How-ever, there are many traps involved in writing embedded code. Lets look at some of these to start with.

Library Routines
The library routines supplied with most compilers use DOS to perform video, keyboard and diskette functions. Your own versions will need to be created instead. Access to the source code for library functions will be helpful. If the ROM BIOS chip is left in place, it should be easy to write routines which substitute for the library.

Segment register values
With plenty of interrupts running around, it is important that you initialise the segment registers (DS,ES and SS,SP) when the jump to your ROM code takes place. Failure to do so can result in stack overflow, and the inability to access any data. Create your own stack somewhere safe in memory first.

Copy data to RAM
Copy your initialised data to RAM, and don't forget to adjust the DS or ES register to point to the segment address! Zero out any RAM block used for uninitialised static variables (ie, having the value 0).

Plug into the interrupt vectors
You may safely take over most interrupt routines, including int 10h etc. You will need to write your own routines to do this, don't rely upon the library functions which come with your compiler. The final section of this booklet demonstrates this. Ensure that interrupt routines which are called are type FAR, and save all the registers. Interrupt routines should also set up segment register values for DS/ES, if they need to access data some-where else.

Here are a couple of ROMMable routines

```
        void set_vect( int_number, int_code )
        unsigned int int_number;
        long int *int_code;
        {
                unsigned int seg, off;
                int far *int_vector = (int far *) 0x00000000;
```

```
            seg = FP_SEG( int_code );
            off = FP_OFF( int_code );
            number &= 0xff;
            number <= 2;
            *int_vector[number] = off;
            *int_vector[number+2] = seg;
    }

    print_str proc near ; ROM Version of int21h, ah = 09
            push si ; use si for indexed addressing
            push ax ; save character
            mov ax,dx ; establish addressibility
            mov si,ax ; dx has offset of string from DS
            pstr1: mov al,[si] ; get character
            cmp al,24h ; is it end of string
            je pstr2 ; yes then exit
            call pc_out ; no, print it then
            inc si ; point to next character
            jmp pstr1 ; repeat
            pstr2: pop ax
            pop si
            ret
    print_str endp

    pc_out proc near ; print character on video screen
            mov ah,0eh ; write tty call
            push bx
            push cx
            mov bh,0 ; assume page zero
            mov cx,1 ; one character to write
            int 10h
            pop cx
            pop bx
            ret
    pc_out endp
```

Keyboard Removal
Some BIOS routines check for keyboard existance prior to checking for user EPROM. If intending to run with the keyboard removed, and the BIOS chips present, either the keyboard must be present during a system reset, or the BIOS will need to be modified.

Running without the BIOS chips
You will need to initialise

- the interrupt vectors (maybe to an iret instruction)
- timer 0 to perform refreshing of dynamic ram
- the DMA and Priority Interrupt Controller (PIC) devices
- RAM to zero so that the parity generator doesn't get confused
- the segment registers SS, DS and ES to their respective areas
- any additional cards or devices used, video, keyboard, disk, A/D

You will need to test

- CPU
- for top of memory

- RAM and ROM
- interrupts working (PIC)
- timer channel operation

MAKE

This is a facility which offers project management of multiple source and object files. A special file (makefile), contains the files which the runtime code is dependant upon. When make is invoked, it checks the date of each file, and decides which files need re-compiling or re-linking.

Create a makefile Use an editor to create makefile, eg

```
$vi makefile
```

In makefile, place the following, ensuring that tab stops are placed between myprog.exe and myprog.obj, and between the left margin and tlink.

```
myprog.exe: myprog.obj f1.obj f2.obj f3.obj
tlink c0l myprog f1 f2 f3, myprog, myprog, cl
myprog.obj: myprog.c f1.c f2.c f3.c
tcc -c -ml -f- myprog.c
f1.obj: f1.c
tcc -c -ml -f- f1.c
f2.obj: f2.c
tcc -c -ml -f- f2.c
f3.obj: f3.c
tcc -c -ml -f- f3.c
```

Now, create the following modules f1.c f2.c f3.c and myprog.c

```
void print_mess1() /* Module f1.c */
{
        printf("This is module f1\n");
}

void print_mess2() /* Module f2.c */
{
        printf("This is module f2\n");
}

void print_mess3() /* Module f3.c */
{
        printf("This is module f3\n");
}

extern void print_mess1(), print_mess2(), print_mess3();
main() /* Module myprog.c */
{
        print_mess1();
        print_mess2();
        print_mess3();
        printf("and this is main\n");
```

```
}
```

Compile each of the above modules, using the tcc stand-alone compiler.

```
$tcc -c -ml -f- myprog.c
$tcc -c -ml -f- f1.c
$tcc -c -ml -f- f2.c
$tcc -c -ml -f- f3.c
```

Now you are in a position to try out the make function.

```
$make
```

This runs the make utility, which will recieve as its input the file contents of makefile, and generate the required runfile myprog.exe

```
$myprog
This is module f1
This is module f2
This is module f3
and this is main
$
```

If changes are made to any of the source files from this point on, you only need to re-run make. This helps to automate the process of program maintenance. It is possible to specify a command file other than makefile, which contains inter-dependancies, eg,

```
$make myprog
```

will perform a make on the inter-dependant commands specified in the file myprog.

DOS INTERRUPT ROUTINES

To maintain compatibility across different hardware machines and interfaces, calls to the DOS are preferrable to the low level access provided by the ROM BIOS routines. Two routines allow access to the DOS interrupt interface. They are intdos() and intdosx(). Both functions generate a DOS interrupt type 0x21.

```
intdos( union REGS *regs, union REGS *regs)
intdosx( union REGS *regs, union REGS *regs, struct SREGS *segregs )
```

The function intdosx() also copies the values for the segregs.x.ds and segregs.x.es into the DS and ES registers. Both functions copy the register values returned from the DOS call into the associated structure, as well as the status of the carry flag. If the carry flag is set, this indicates an error. The following functions illustrate calls using the DOS interrupt interface.

```
#include <dos.h>

union REGS regs;

char rs232_read() {
```

```
        regs.h.ah = 3; intdos( &regs, &regs ); return( regs.h.al );
}

void rs232_write( byte )
char byte;
{
        regs.h.ah = 4; intdos( &regs, &regs );
}
```

When a C program is run, DOS opens five pre-defined streams for the program to access. The streams are used with the C functions open(), read(), write(), and close(). The five pre-defined streams are,

```
0=CON stdin 1=CON stdout
2=CON stderr 3=AUX stdaux COM1
4=PRN stdprn LPT1
```

A program may access these opened streams, however, direct reads and writes can fail due to DOS re-direction. It is best to re-open device first, before performing any operations. This will prevent any re-direction.

The following code portion shows how to write to the prn device from within a C program.

```
#include <fcntl.h>
#include <string.h>
#include <io.h>

main() {
        char *message = "This is a message for the prn device.";
        int prnhandle;
        if( (prnhandle = open( "PRN", O_WRONLY, O_BINARY) ) == -1 ) {
                printf("Couldn't open prn device.\n");
                exit( 1 );
        }
        printf("Printer is on-line. Now printing the message.\n");
        if( (write( prnhandle, message, strlen(message) )) == -1 ) {
                printf("write to prn device failed.\n");
                exit(2);
        }
        printf("Message has been printed. Lets close and exit to DOS.\n");
        close( prnhandle );
}
```

INTERFACING TO MOUSE.SYS

The mouse driver locates itself in memory at boot time. It takes over both int 33h and int 10h. The driver is identified by an eight character sequence, in the case of the microsoft mouse, it is the sequence MS$MOUSE. Before issuing any calls to the mouse driver, you should first establish its presence. There are two methods of accomplishing this. First, you can test to see if the driver is installed by checking for the device name, or use a mouse call to int 33h. The routine which follows returns 0 if the mouse driver does not exist, -1 if it is present.

```
#include <dos.h>
```

```
        int mouse_exist2( void ) {
                _AX = 0;
                geninterrupt( 0x33 );
                return _AX;
                /* _BX will also contain the number of buttons */
        }
```

The mouse_exist2() call also initialises the mouse system to the default parameters, if it is present.

Mouse Function Calls
INT 33h

```
        AX = 0 Mouse Installed Flag and RESET
                Returns AX as a status byte, 0 = not present, -1 = present (and
RESET)
                The default parameters for a RESET are,
                cursor position = screen centre
                internal cursor flag = -1 (not displayed)
                graphics cursor = arrow (-1, -1)
                text cursor = inverting box
                interrupt mask call = all 0 (no interrupts)
                light pen emulation = enabled
                mouse/pixel ratio (H)= 8 to 8
                mouse/pixel ratio (V)= 16 to 8
                min/max cursor pos H = Depends upon card/mode
                min/max cursor pos V = Depends upon card/mode
        AX = 1 Show Cursor
                Increments the internal cursor flag, and if zero, displays the cursor
on the screen. If the                     cursor flag is already zero, this function
does nothing.
        AX = 2 Hide Cursor
                Decrements the internal cursor flag, and removes the cursor from the
screen.
        AX = 3 Get Mouse Position and Button Status
                Returns the state of the left and right mouse buttons, as well as the
horizontal and vertical           co-ordinates of the cursor.
                BX bit 0 = left button (1=pressed, 0=released)
                BX bit 1 = right button
                CX = cursor position, horizontal
                DX = cursor position, vertical
        AX = 4 Set Mouse Cursor Position
                Upon entry, CX = new horizontal position
                DX = new vertical position
        AX = 5 Get Mouse Button Press Information
                Upon entry, BX = which button to check for, (0=lft,1=rght)
                Returns the following information.
                        AX = button status, bit 0 = left button
                        bit 1 = right button (1=pressed, 0=released)
                        BX = count of button presses (0 to 32767, reset to 0 after
this call)
                        CX = cursor position, horizontal, at last press
                        DX = cursor position, vertical, at last press
        AX = 6 Get Button Release Information
                Upon entry, BX = which button to check for, (0=lft,1=rght)
```

```
                Returns the following information.
                        AX = button status, bit 0 = left button
                        bit 1 = right button (1=pressed, 0=released)
                        BX = count of button releases (0 to 32767, reset to 0 after
this call)
                        CX = cursor position, horizontal, at last release
                        DX = cursor position, vertical, at last release
        AX = 7 Set Minimum and Maximum Horizontal Position
                Upon entry, CX = minimum position
                DX = maximum position
        AX = 8 Set Minimum and Maximum Vertical Position
                Upon entry, CX = minimum position
                DX = maximum position
        AX = 9 Set Graphics Cursor Block
                Upon entry, BX = cursor hot spot (horizontal)
                CX = cursor hot spot (vertical)
                DX = pointer to screen and cursor masks
        AX = 10 Set Text Cursor
                Upon entry, BX = cursor select (0=software, 1=hardware)
                CX = screen mask or scan line start
                DX = cursor mask or scan line end
        AX = 11 Read Mouse Motion Counters
                Return values,
                        CX = horizontal count
                        DX = vertical count
        AX = 12 Set User-Defined Subroutine Input Mask
                Upon entry, CX = call mask
                DX = address offset to subroutine
                ES = address segment to subroutine
                Each bit of the call mask corresponds to
                        0 = Cursor position change
                        1 = Left button pressed
                        2 = Left button released
                        3 = Right button pressed
                        4 = Right button released
                        5-15 = Not used
                To enable an interrupt, set the corresponding bit to a 1.
                When the event occurs, the mouse driver will call your subroutine.
        AX = 13 Light Pen Emulation Mode ON
        AX = 14 Light Pen Emulation Mode OFF
        AX = 15 Set Mickey/Pixel Ratio
                Upon entry, CX = horizontal ratio
                DX = vertical ratio
                The ratios specify the number of mickeys per 8 pixels. The values
                must be within the range 1 to 32767. The default horizontal ratio
                is 8:8, whilst the default ratio for the vertical is 16:8
        AX = 16 Conditional OFF
                Upon entry, CX = upper x screen co-ordinate
                DX = upper y screen co-ordinate
                SI = lower x screen co-ordinate
                DI = lower y screen co-ordinate
                This function defines a region on the screen for updating. If the
mouse
                moves to the defined region, it will be hidden while the region is
updated.
```

```
                After calling this function, you must call function 1 again to show
the
                cursor.
        AX = 19 Set Double Speed Threshold
                Upon entry, DX = threshold speed in mickeys/second
                This function can be used to double the cursors motion on the screen.
                The default value is 64 mickeys/second.
```

Mouse Demonstration Program

```
        /* mousedem.c, an illustration of how to interface to mouse.sys */
        /* by B Brown, 1988 */
        #include <dos.h>

        static unsigned int arrow[2][16] = {
                { 0xfff0, 0xffe0, 0xffc0, 0xff81, 0xff03, 0x607, 0xf, 0x1f, 0xc03f,
                0xf07f, 0xffff, 0xffff, 0xffff, 0xffff, 0xffff, 0xffff } ,
                { 0, 6, 0x0c, 0x18, 0x30, 0x60, 0x70c0, 0x1d80, 0x700, 0, 0, 0, 0,
                0, 0, 0 }
        } ;

        void set640_200() {
                _AH = 0; _AL = 6; geninterrupt( 0x10 );
        }

        int mouse_exist() {
                _AX = 0; geninterrupt( 0x33 ); return _AX;
                /* _BX will also contain the number of buttons */
        }

        void show_cursor() {
                _AX = 1; geninterrupt( 0x33 );
        }

        void shape_cursor( buffer, hchs, vchs )
        unsigned int *buffer;
        unsigned int hchs, vchs;
        {
                _AX = 9; _BX = hchs; _CX = vchs;
                _DX = FP_OFF( buffer ); _ES = FP_SEG( buffer );
                geninterrupt( 0x33 );
        }

        main() {
                if( mouse_exist() == 0 ) {
                        printf("Mouse driver is not loaded. Returning to DOS.\n");
                        exit(1);
                }
                set640_200();
                shape_cursor( arrow, 0, 0 );
                show_cursor();
                while( 1 )
                        ;
        }
```

APPENDIX A

```
VIDEO LIBRARY ROUTINES, C Source code
/* C_UTILITIES FOR IBM-PC, MICROSOFT C COMPILER V3.0 */
/* */
/* Written by B. Brown */
/* Central Institute of Technology, */
/* Private Bag, Trentham */
/* Wellington, New Zealand, */
/* */
/* The routines are listed as follows, */
/* */
/* getmode() returns screen mode setting into */
/* 'screenmode', the number of columns */
/* into 'columns', and screen page number */
/* into 'activepage' */
/* */
/* setmode() sets the video mode based on the value */
/* of screenmode, where */
/* 0 =40x25BW 1 =40x25CO 2 =80x25BW */
/* 3 =80x25CO 4 =320x200CO 5 =320x200BW */
/* 6 =640x200BW 7 =80x25 monitor */
/* */
/* setcurs(col, row) sets the cursor position */
/* indicated by 'col' and 'row' */
/* */
/* rcurspos() returns the current cursor position */
/* */
/* rcharattr() returns the character and attribute at */
/* the current cursor location */
/* */
/* rchar() returns the character at the current */
/* cursor position */
/* */
/* rattr() returns the attribute at the current */
/* cursor position */
/* */
/* wcharattr(c, color) writes character and its */
/* attribute to the current cursor loc */
/* */
/* wcharonly(c) writes character to the current */
/* cursor position */
/* */
/* wdot(x,y,color) writes a dot specified by x,y in */
/* color */
/* */
/* rdot(x,y) returns color of dot located at x,y */
/* */
/* setborder(color) sets the border color */
/* */
/* BLACK 0 RED 4 DARK_GREY 8 LIGHT_RED 12 */
/* BLUE 1 MAGENTA 5 LIGHT_BLUE 9 LIGHT_MAGENTA 13 */
```

```
/* GREEN 2 BROWN 6 LIGHT_GREEN 10 YELLOW 14 */
/* CYAN 3 LIGHT_GREY 7 LIGHT_CYAN 11 WHITE 15 */
/* */
/* setpalette( palette) sets palette color in medium */
/* resolution color mode */
/* */
/* medcolor( bckgnd, border ) sets the background and */
/* border colors in medres mode */
/* Only works on active page, and */
/* sets entire screen */
/* */
/* selectpage(page) selects active display page */
/* */
/* wstr( message, color ) */
/* writes the characters or text string pointed to by */
/* the pointer message, using the foreground color */
/* specified. ( Actually, the background color can */
/* also be specified. The various modes are, */
/* */
/* bits 0 - 2 specify the foreground color */
/* bit 3 specifies the intensity, 0 = normal */
/* bits 4 - 6 specify the background color */
/* bit 7 specifies blinking, 0 = non-blinking*/
/* */
/* The cursor is moved after each character is written.*/
/* The text should not contain any control characters, */
/* eg, don't use /n */
/* Typical use is, */
/* */
/* static char *text = "Have a nice morning."; */
/* */
/* wstr( text, 4 ); */
/* puts("/n"); */
/* */
/* */
/* scrollup(tlr,tlc,brr,brc,attr,lines) scrolls up */
/* active display area given by topleftrow,*/
/* topleftcolumn, bottomrightrow, */
/* bottomrightcolumn, using attr as a */
/* color for the bottom line, scrolling */
/* the number of lines (0=all) */
/* */
/* scrolldown(tlr,tlc,brr,brc,attr,lines) scrolls down */
/* the active display area. */
/* */
/* clear_window(tlr,tlc,brr,brc,attr) clears the */
/* display window area */
/* */
/* line(x1,y1,x2,y2,lcolor) draws a line between */
/* co-ordinates in the color specified */
/* */
/* circle ( xcentre, ycentre, radius, color ) draws */
/* a circle in the specified color. Works */
/* only in screen mode 4, 320*200Color */
/* */
```

```
/* NOTES ON THE USE OF VIDEO.LIB */
/* This has been implemented as a library. All the */
/* functions listed here can be called from your C */
/* programs. To do this however, the following guide */
/* outlines should be adhered to! */
/* */
/* 1: Incorporate the following declarations at the */
/* start of your C program. */
/* */
/* extern union REGS regs; */
/* extern unsigned char activepage; */
/* extern unsigned char columns; */
/* extern unsigned char screenmode; */
/* */
/* 2: At linking time, specify the inclusion of */
/* CHELP.LIB */
#include <stdio.h> /* for the screen rout's */
#include <conio.h> /* used for outp() */
#include <dos.h> /* used for int86() */
#include <math.h> /* for circle routine */
union REGS regs; /* programming model 8088 */
unsigned char activepage; /* current video screen */
unsigned char columns=79; /* number of columns */
unsigned char screenmode=2;/* display mode, 80x25 col */
/* getmode() is a function that finds out the current */
/* display page number (any one of eight), the screen */
/* mode currently in use, and the number of columns */
/* (40, 80 etc) */

getmode() {
        regs.h.ah = 15; int86( 0x10, &regs, &regs);
        activepage = regs.h.bh; screenmode = regs.h.al;
        columns = regs.h.ah;
}

/* setmode() is a function that sets the display mode */
/* of the video display. First change the value of the */
/* global variable 'screenmode', then call the function*/
/* 'setmode()'. It clears the display page. */
setmode() {
        regs.h.ah=0; regs.h.al=screenmode & 7;
        int86(0x10, &regs, &regs);
}

setcurs(col, row)
unsigned int col, row;
{
        getmode(); regs.h.ah = 2; regs.h.dh = row;
        regs.h.dl = col; regs.h.bh = activepage;
        int86( 0x10, &regs, &regs);
}

rcurspos() {
        getmode(); regs.h.ah = 3; regs.h.bh = activepage;
        int86( 0x10, &regs, &regs); return( regs.x.dx );
```

```
        /* row=regs.h.dh, column=regs.h.dl */
}

rcharattr() {
        getmode(); regs.h.ah = 8;
        int86(0x10, &regs, &regs); return( regs.x.ax );
        /* attribute=regs.h.ah, character=regs.h.al */
}

rchar() {
        getmode(); regs.h.ah = 8;
        int86(0x10, &regs, &regs); return( regs.h.al );
}

rattr() {
        getmode(); regs.h.ah = 8;
        int86(0x10, &regs, &regs); return( regs.h.ah );
}

wcharattr(c, color)
char c;
unsigned int color;
{
        getmode(); regs.h.ah = 9;
        regs.h.bh = activepage; regs.x.cx = 1;
        regs.h.al = c; regs.h.bl = color;
        int86( 0x10, &regs, &regs);
}

wcharonly(c)
char c;
{
        getmode(); regs.h.ah = 10;
        regs.h.bh = activepage; regs.x.cx = 1;
        regs.h.al = c; int86(0x10, &regs, &regs);
}

wdot( x, y, color )
unsigned int x, y, color;
{
        getmode();
        switch( screenmode ) {
                case 4:
                case 5:
                case 6:
                        regs.h.ah = 12; regs.h.bh = 0;
                        regs.x.dx = y; regs.x.cx = x;
                        regs.h.al = color; int86( 0x10, &regs, &regs);
                        break;
                default:
                        break;
        }
}

rdot( x, y)
```

```
unsigned int x, y;
{
        getmode();
        switch( screenmode ) {
                case 4:
                case 5:
                case 6:
                        regs.h.ah = 13; regs.h.bh = 0;
                        regs.x.dx = y; regs.x.cx = x;
                        int86(0x10, &regs, &regs);
                        return ( regs.h.al );
                        break;
                default:
                        return ( -1 );
                        break;
        }
}

setborder(color)
unsigned int color;
{
        outp( 0x3d9, color & 0x0f );
}

setpalette( palette )
int palette;
{
        getmode();
        if( screenmode <> 4 )
                return( -1 );
        regs.h.ah = 0x0b; regs.h.bh = 1; regs.h.bl = palette & 1;
        int86( 0x10, &regs, &regs );
}

medcolor( bckgnd, border )
int bckgnd, border;
{
        getmode();
        if( screenmode <> 4 )
                return( -1 );
                regs.h.ah = 0x0b; regs.h.bh = 0;
                regs.h.bl = (bckgnd << 4) + border;
                int86( 0x10, &regs, &regs );
}

selectpage(page)
unsigned int page;
{
        getmode();
        switch( screenmode ) {
                case 0:
                case 1:
                        page = page & 7;
                        break;
                case 2:
```

```
                case 3:
                case 7:
                        page = page & 3;
                        break;
                default:
                        page = 0;
                        break;
        }
        regs.h.ah = 5; regs.h.al = page;
        int86(0x10, &regs, &regs);
}

wstr( message, color )
char *message;
unsigned char color;
{
        unsigned int rowpos, colpos;
        getmode(); rcurspos();
        colpos = regs.h.dl; rowpos = regs.h.dh;
        if ( screenmode != 1 && screenmode != 3 )
                return ( -1 );
        while ( *message ) {
                wcharattr( *message, color );
                ++colpos;
                if(colpos > columns) /* check for edge of screen */
                {
                        colpos = 0; /* set to beginning of line */
                        ++rowpos; /* increment row count */
                        if( rowpos > 24 ) /* do we need to scroll? */
                        {
                                rowpos = 24;
                                regs.h.ah = 6; /* scroll up function call */
                                regs.h.al = 1; /* scroll entire screen */
                                regs.h.ch = 0; /* upper left corner */
                                regs.h.cl = 0; regs.h.dl = columns;
                                regs.h.dh = 24; regs.h.bh = color;
                                int86(0x10,&regs,&regs); /* scroll screen */
                        }
                }
                setcurs(colpos, rowpos); /* update cursor */
                ++message; /* next character in string */
        }
}

scrollup( tlr, tlc, brr, brc, attr, lines )
unsigned int tlr, tlc, brr, brc, attr, lines;
{
        union REGS regs;
        regs.h.ah = 6; regs.h.al = lines;
        regs.h.bh = attr; regs.h.ch = tlr;
        regs.h.cl = tlc; regs.h.dh = brr;
        regs.h.dl = brc; int86( 0x10, &regs, &regs );
}

scrolldown( tlr, tlc, brr, brc, attr, lines )
```

```
unsigned int tlr, tlc, brr, brc, attr, lines;
{
        union REGS regs;
        regs.h.ah = 7; regs.h.al = lines;
        regs.h.bh = attr; regs.h.ch = tlr;
        regs.h.cl = tlc; regs.h.dh = brr;
        regs.h.dl = brc; int86( 0x10, &regs, &regs );
}

clear_window( tlr, tlc, brr, brc, attr )
unsigned int tlr, tlc, brr, brc, attr;
{
        union REGS regs;
        regs.h.ah = 6; regs.h.al = 0; regs.h.bh = attr;
        regs.h.ch = tlr; regs.h.cl = tlc; regs.h.dh = brr;
        regs.h.dl = brc; int86( 0x10, &regs, &regs );
}

line( x1, y1, x2, y2, lcolor )
int x1, y1, x2, y2, lcolor;
{
        int xx, yy, delta_x, delta_y, si, di;
        getmode();
        switch( screenmode ) {
                case 0: case 1: case 2: case 3: case 7: return(-1); break;
                default: break;
        }
        if( x1 > x2 ) {
                xx = x2; x2 = x1; x1 = xx;
                yy = y2; y2 = y1; y1 = yy;
        }
        delta_y = y2 - y1;
        if ( delta_y >= 0 )
                si = 1;
        else {
                si = -1; delta_y = -delta_y;
        }
        delta_x = x2 - x1;
        if ( delta_x >= 0 )
                di = 1;
        else {
                di = 0; delta_x = -delta_x;
        }
        if( (delta_x - delta_y) < 0 )
                steep( x1, y1, delta_x, delta_y, si, di, lcolor );
        else
                easy ( x1, y1, delta_x, delta_y, si, di, lcolor);
        return ( 0 );
}

steep( x1, y1, delta_x, delta_y, si, di, color )
int x1, y1, delta_x, delta_y, si, di, color;
{
        int half_delta_y, cx, dx, bx, ax, count;
        half_delta_y = delta_y / 2;
```

```
        cx = x1; dx = y1; bx = 0; count = delta_y;
        newdot2: wdot( cx, dx, color );
                dx = dx + si; bx = bx + delta_x;
                if ( bx - half_delta_y <= 0 )
                        goto dcount2;
                bx = bx - delta_y; cx = cx + di;
        dcount2: --count;
                if ( count >= 0 )
                        goto newdot2;
}

easy( x1, y1, delta_x, delta_y, si, di, color )
int x1, y1, delta_x, delta_y, si, di, color;
{
        int half_delta_x, cx, dx, bx, ax, count;
        half_delta_x = delta_x / 2;
        cx = x1; dx = y1; bx = 0; count = delta_x;
        newdot:
                wdot( cx, dx, color );
                cx = cx + di; bx = bx + delta_y;
                if ( bx - half_delta_x <= 0 )
                        goto dcount;
                bx = bx - delta_x; dx = dx + si;
        dcount:
                --count;
                if ( count >= 0 )
                        goto newdot;
}

circle ( xcentre, ycentre, radius, color )
int xcentre, ycentre, radius, color;
{
        int xfirst, yfirst, xsecond, ysecond, totalpoints = 16;
        float angle, DA;
        getmode();
        if ( screenmode != 4 )
                return (-1 );
        DA = 6.28318 / totalpoints;
        xfirst = xcentre + radius;
        yfirst = ycentre;
        for( angle = DA; angle <= 6.28318; angle = angle + DA ) {
                xsecond = xcentre + radius * cos(angle);
                ysecond = ycentre + radius * sin(angle) * .919999;
                line( xfirst, yfirst, xsecond, ysecond, color );
                xfirst = xsecond;
                yfirst = ysecond;
        }
        line( xfirst, yfirst, xcentre+radius, ycentre, color );
        return ( 1 );
}
```

Copyright Brian Brown, 1986-2000. All rights reserved.

C Programming

An introduction

INDEX Next

A SIMPLE C PROGRAM
The following program is written in the C programming language.

```
#include <stdio.h>

main()
{
        printf("Programming in C is easy.\n");
}
```

Sample Program Output

```
Programming in C is easy.
_
```

A NOTE ABOUT C PROGRAMS
In C, lowercase and uppercase characters are very important! All commands in C must be lowercase. The C programs starting point is identified by the word

```
main()
```

This informs the computer as to where the program actually starts. The brackets that follow the keyword *main* indicate that there are no arguments supplied to this program (this will be examined later on).

The two braces, { and }, signify the begin and end segments of the program. The purpose of the statment

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

is to allow the use of the *printf* statement to provide program output. Text to be displayed by *printf()* must be enclosed in double quotes. The program has only one statement

```
printf("Programming in C is easy.\n");
```

printf() is actually a function (procedure) in C that is used for printing variables and text. Where text appears in double quotes "", it is printed without modification. There are some exceptions however. This has to do with the \ and % characters. These characters are modifier's, and for the present the \ followed by the n character represents a newline character. Thus the program prints

```
        Programming in C is easy.
```

and the cursor is set to the beginning of the next line. As we shall see later on, what follows the \ character will determine what is printed, ie, a tab, clear screen, clear line etc. Another important thing to remember is that all C statements are terminated by a semi-colon ;

Click here for a pascal comparison.

Summary of major points so far

- program execution begins at *main()*
- keywords are written in lower-case
- statements are terminated with a semi-colon
- text strings are enclosed in double quotes
- C is case sensitive, use lower-case and try not to capitalise variable names
- \n means position the cursor on the beginning of the next line
- *printf()* can be used to display text to the screen
- the curly braces {} define the beginning and end of a program block

---

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Next

C Programming

INDEX Previous Next

A SIMPLE C PROGRAM

This shows a C and Pascal program side by side, for comparison purposes.

```
#include <stdio.h>

main()
{
    printf("Programming in C is easy.\n");
}
```

```
program One (output);

begin
    writeln('Programming in C is easy')
end.
```

Can you see some similar styles?

```
{                 begin
}                 end
printf            writeln
"textstring"      'textstring'
```

Note how the braces are similar in usage to the *begin* and *end* statements in Pascal. Note also that C encloses strings in double quotes, whereas Pascal uses single quotes.

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

INDEX Previous Next

CLASS EXERCISE C1
What will the following program output?

```
#include <stdio.h>

main()
{
        printf("Programming in C is easy.\n");
        printf("And so is Pascal.\n");
}
```

And this program?

```
#include <stdio.h>

main()
{
        printf("The black dog was big. ");
        printf("The cow jumped over the moon.\n");
}
```

Another thing about programming in C is that it is not necessary to repeatedly call the *printf* routine, so try and work out what the following program displays,

```
#include <stdio.h>

main()
{
        printf("Hello...\n..oh my\n...when do i stop?\n");
}
```

Answers

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

Previous

ANSWERS TO CLASS EXERCISE C1

```
        #include <stdio.h>

        main()
        {
                printf("Programming in C is easy.\n");
                printf("And so is Pascal.\n");
        }
```

```
        Programming in C is easy.
        And so is Pascal.
        _
```

```
#include <stdio.h>
        main()
        {
                printf("The black dog was big. ");
                printf("The cow jumped over the moon.\n");
        }
```

```
        The black dog was big. The cow jumped over the moon.
        _
```

```
#include <stdio.h>

        main()
        {
                printf("Hello...\n..oh my\n...when do i stop?\n");
        }
```

```
        Hello...
        ..oh my
```

...when do i stop?

_

©Copyright B Brown. 1984-2000. All rights reserved.

Previous

C Programming

WHAT ABOUT VARIABLES

C provides the programmer with FOUR basic data types. User defined variables must be declared before they can be used in a program.

Get into the habit of declaring variables using lowercase characters. Remember that C is case sensitive, so even though the two variables listed below have the same name, they are considered different variables in C.

```
sum
Sum
```

The declaration of variables is done after the opening brace of *main()*,

```
#include <stdio.h>

main()
{
        int sum;

        sum = 500 + 15;
        printf("The sum of 500 and 15 is %d\n", sum);
}
```

```
Sample Program Output
The sum of 500 and 15 is 515
```

It is possible to declare variables elsewhere in a program, but lets start simply and then get into variations later on.

The basic format for declaring variables is

```
data_type   var, var, ... ;
```

where *data_type* is one of the four basic types, an *integer*, *character*, *float*, or *double* type.

The program declares the variable *sum* to be of type INTEGER (int). The variable *sum* is then assigned the value of 500 + 15 by using the assignment operator, the = sign.

```
sum = 500 + 15;
```

Now lets look more closely at the *printf()* statement. It has two arguments, separated by a comma. Lets look at the first argument,

```
"The sum of 500 and 15 is %d\n"
```

The *%* sign is a special character in C. It is used to display the value of variables. When the program is executed, C starts printing the text until it finds a *%* character. If it finds one, it looks up for the next argument (in this case *sum*), displays its value, then continues on.

The *d* character that follows the *%* indicates that a decimal integer is expected. So, when the *%d* sign is reached, the next argument to the *printf()* routine is looked up (in this case the variable *sum*, which is 515), and displayed. The *\n* is then executed which prints the newline character.

The output of the program is thus,

```
The sum of 500 and 15 is 515
_
```

Some of the formatters for *printf* are,

```
Cursor Control Formatters
\n      newline
\t      tab
\r      carriage return
\f      form feed
\v      vertical tab

Variable Formatters
%d      decimal integer
%c      character
%s      string or character array
%f      float
%e      double
```

The following program prints out two integer values separated by a TAB

It does this by using the \t cursor control formatter

```
#include <stdio.h>

main()
{
        int sum, value;

        sum = 10;
        value = 15;
        printf("%d\t%d\n", sum, value);
```

```
}
```

Program output looks like

```
10      15
_
```

© Copyright B Brown. 1984-2000. All rights reserved. INDEX Previous Next

C Programming

INDEX Previous Next

CLASS EXERCISE C2

What is the output of the following program?

```
#include <stdio.h>

main()
{
        int   value1, value2, sum;

        value1 = 35;
        value2 = 18;
        sum = value1 + value2;
        printf("The sum of %d and %d is %d\n", value1, value2, sum);
}
```

Note that the program declares three variables, all integers, on the same declaration line. This could've been done by three separate declarations,

```
int  value1;
int  value2;
int  sum;
```

Answers

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

Previous

ANSWER TO CLASS EXERCISE C2

```
#include <stdio.h>

main()
{
        int   value1, value2, sum;

        value1 = 35;
        value2 = 18;
        sum = value1 + value2;
        printf("The sum of %d and %d is %d\n", value1, value2, sum);
}


The sum of 35 and 18 is 53
_
```

©Copyright B Brown. 1984-2000. All rights reserved.

Previous

C Programming

INDEX Previous Next

COMMENTS
The addition of comments inside programs is desirable. These may be added to C programs by enclosing them as follows,

```
        /* bla bla bla bla bla bla */
```

Note that the **/*** opens the comment field and ***/** closes the comment field. Comments may span multiple lines. Comments may not be nested one inside another.

```
        /* this is a comment. /* this comment is inside */ wrong */
```

In the above example, the first occurrence of ***/** closes the comment statement for the entire line, meaning that the text ***wrong*** is interpreted as a C statement or variable, and in this example, generates an error.

What Comments Are Used For

- documentation of variables and their usage
- explaining difficult sections of code
- describes the program, author, date, modification changes, revisions etc
- copyrighting

Basic Structure of C Programs
C programs are essentially constructed in the following manner, as a number of well defined sections.

```
        /* HEADER SECTION                         */
        /* Contains name, author, revision number*/

        /* INCLUDE SECTION                        */
        /* contains #include statements           */

        /* CONSTANTS AND TYPES SECTION            */
        /* contains types and #defines            */

        /* GLOBAL VARIABLES SECTION               */
        /* any global variables declared here     */

        /* FUNCTIONS SECTION                      */
        /* user defined functions                 */
```

```
/* main() SECTION                         */

int main()
{

}
```

Adhering to a well defined structured layout will make your programs

- easy to read
- easy to modify
- consistent in format
- self documenting

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

INDEX Previous Next

MORE ABOUT VARIABLES
Variables must begin with a character or underscore, and may be followed by any combination of characters, underscores, or the digits 0 - 9. The following is a list of valid variable names,

```
summary
exit_flag
i
Jerry7
Number_of_moves
_valid_flag
```

You should ensure that you use meaningful names for your variables. The reasons for this are,

- meaningful names for variables are self documenting (see what they do at a glance)
- they are easier to understand
- there is no correlation with the amount of space used in the .EXE file
- makes programs easier to read

CLASS EXERCISE C3
Why are the variables in the following list invalid,

```
value$sum
exit flag
3lotsofmoney
char
```

Answers

VARIABLE NAMES AND PREFIXES WHEN WRITING WINDOWS OR OS/2 PROGRAMS
During the development of OS/2, it became common to add prefix letters to variable names to indicate the data type of variables.

This enabled programmers to identify the data type of the variable without looking at its declaration, thus they could easily check to see if they were performing the correct operations on the data type and hopefully, reduce the number of errors.

Prefix	Purpose or Type
b	a byte value

```
c                   count or size
clr                 a variable that holds a color
f                   bitfields or flags
h                   a handle
hwnd                a window handle
id                  an identity
l                   a long integer
msg                 a message
P                   a Pointer
rc                  return value
s                   short integer
ul                  unsigned long integer
us                  unsigned short integer
sz                  a null terminated string variable
psz                 a pointer to a null terminated string variable
```

In viewing code written for Windows or OS/2, you may see variables written according to this convention.

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

Previous

ANSWER TO CLASS EXERCISE C3

```
value$sum       contains a $
exit flag       contains a space
3lotsofmoney    begins with a digit
char            is a reserved keyword
```

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

INDEX | Previous | Next

DATA TYPES AND CONSTANTS

The four basic data types are

- **INTEGER**
  These are whole numbers, both positive and negative. Unsigned integers (positive values only) are supported. In addition, there are short and long integers.

  The keyword used to define integers is,

  ```
  int
  ```

  An example of an integer value is 32. An example of declaring an integer variable called **sum** is,

  ```
  int sum;
  sum = 20;
  ```

- **FLOATING POINT**
  These are numbers which contain fractional parts, both positive and negative. The keyword used to define float variables is,

  ```
  float
  ```

  An example of a float value is 34.12. An example of declaring a float variable called **money** is,

  ```
  float money;
  money = 0.12;
  ```

- **DOUBLE**
  These are exponetional numbers, both positive and negative. The keyword used to define double variables is,

  ```
  double
  ```

  An example of a double value is 3.0E2. An example of declaring a double variable called **big** is,

  ```
  double big;
  big = 312E+7;
  ```

- **CHARACTER**
  These are single characters. The keyword used to define character variables is,

  ```
  char
  ```

  An example of a character value is the letter **A**. An example of declaring a character variable called **letter** is,

  ```
  char letter;
  letter = 'A';
  ```

  Note the assignment of the character *A* to the variable *letter* is done by enclosing the value in **single quotes**. Remember the golden rule: Single character - Use single quotes.

Sample program illustrating each data type

```
#include < stdio.h >

main()
{
        int   sum;
        float money;
        char  letter;
        double pi;

        sum = 10;               /* assign integer value */
        money = 2.21;           /* assign float value */
        letter = 'A';           /* assign character value */
        pi = 2.01E6;            /* assign a double value */

        printf("value of sum = %d\n", sum );
        printf("value of money = %f\n", money );
        printf("value of letter = %c\n", letter );
        printf("value of pi = %e\n", pi );
}


Sample program output
value of sum = 10
value of money = 2.210000
value of letter = A
value of pi = 2.010000e+06
```

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

INITIALIZING DATA VARIABLES AT DECLARATION TIME
Unlike PASCAL, in C variables may be initialized with a value when they are declared. Consider the following declaration, which declares an integer variable *count* which is initialized to 10.

```
int  count = 10;
```

SIMPLE ASSIGNMENT OF VALUES TO VARIABLES
The **=** operator is used to assign values to data variables. Consider the following statement, which assigns the value 32 an integer variable *count*, and the letter **A** to the character variable *letter*

```
count = 32;
letter = 'A';
```

THE VALUE OF VARIABLES AT DECLARATION TIME
Lets examine what the default value a variable is assigned when its declared. To do this, lets consider the following program, which declares two variables, *count* which is an integer, and *letter* which is a character.

Neither variable is pre-initialized. The value of each variable is printed out using a *printf()* statement.

```
#include <stdio.h>

main()
{
        int   count;
        char  letter;

        printf("Count = %d\n", count);
        printf("Letter = %c\n", letter);
}
```

```
Sample program output
Count = 26494
Letter = f
```

It can be seen from the sample output that the values which each of the variables take on at declaration time are **no-zero**. In C, this is common, and programmers must ensure that variables are assigned values before using them.

If the program was run again, the output could well have different values for each of the variables. We can never assume that variables declare in the manner above will take on a specific value.

Some compilers may issue warnings related to the use of variables, and Turbo C from Borland issues the following warning,

```
possible use of 'count' before definition in function main
```

RADIX CHANGING

Data numbers may be expressed in any base by simply altering the modifier, e.g., decimal, octal, or hexadecimal. This is achieved by the letter which follows the % sign related to the *printf* argument.

```
#include <stdio.h>

main() /* Prints the same value in Decimal, Hex and Octal */
{
        int   number = 100;

        printf("In decimal the number is %d\n", number);
        printf("In hex the number is %x\n", number);
        printf("In octal the number is %o\n", number);
        /* what about %X\n as an argument?  */
}
```

```
Sample program output
In decimal the number is 100
In hex the number is 64
In octal the number is 144
```

Note how the variable *number* is initialized to 100 at the time of its declaration.

DEFINING VARIABLES IN OCTAL AND HEXADECIMAL

Often, when writing systems programs, the programmer needs to use a different number base rather than the default decimal.

Integer constants can be defined in octal or hex by using the associated prefix, e.g., to define an integer as an octal constant use *0* (zero)

```
        int     sum = 0567;
```

To define an integer as a hex constant use *0x* (zero followed by x or X)

```
        int     sum = 0x7ab4;
        int     flag = 0x7AB4;     /* Note upper or lowercase hex ok */
```

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

MORE ABOUT FLOAT AND DOUBLE VARIABLES
C displays both float and double variables to six decimal places. This does NOT refer to the precision (accuracy) of which the number is actually stored, only how many decimal places *printf()* uses to display these variable types.

The following program illustrates how the different data types are declared and displayed,

```
#include <stdio.h>

main()
{
        int     sum = 100;
        char    letter = 'Z';
        float   set1 = 23.567;
        double  num2 = 11e+23;

        printf("Integer variable is %d\n", sum);
        printf("Character is %c\n", letter);
        printf("Float variable is %f\n", set1);
        printf("Double variable is %e\n", num2);
}
```

```
Sample program output
Integer variable is 100
Character variable is Z
Float variable is 23.567000
Double variable is 11.000000e23
```

To change the number of decimal places printed out for float or double variables, modify the %f or %e to include a precision value, eg,

```
printf("Float variable is %.2f\n", set1 );
```

In this case, the use of %.2f limits the output to two decimal places, and the output now looks like

```
Sample program output
Integer variable is 100
Character variable is Z
Float variable is 23.56
Double variable is 11.000000e23
```

SPECIAL NOTE ABOUT DATA TYPE CONVERSION
Consider the following program,

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

```
	main()
	{
		int  value1 = 12, value2 = 5;
		float answer = 0;

		answer = value1 / value2;
		printf("The value of %d divided by %d is %f\n",value1,value2,answer
);
	}
```

Sample program output

```
	The value of 12 divided by 5 is 2.000000
```

Even though the above declaration seems to work, it is not always 100% reliable. **Note** how *answer* does not contain a proper fractional part (ie, all zero's).

To ensure that the correct result always occurs, the data type of *value1* and *value2* should be converted to a float type before assigning to the float variable *answer*. The following change illustrates how this can be done,

```
	answer = (float)value1 / (float)value2;
```

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

INDEX Previous Next

DIFFERENT TYPES OF INTEGERS

A normal integer is limited in range to +-32767. This value differs from computer to computer. It is possible in C to specify that an integer be stored in four memory locations instead of the normal two. This increases the effective range and allows very large integers to be stored. The way in which this is done is as follows,

```
long int big_number = 245032L;
```

To display a long integer, use %l, ie,

```
printf("A larger number is %l\n", big_number );
```

Short integers are also available, eg,

```
short int   small_value = 114h;
printf("The value is %h\n", small_value);
```

Unsigned integers (positive values only) can also be defined.

The size occupied by integers varies upon the machine hardware. ANSI C (American National Standards Institute) has tried to standardise upon the size of data types, and hence the number range of each type.

The following information is from the on-line help of the Turbo C compiler,

```
Type: int
  Integer data type

Variables of type int are one word in length.
They can be signed (default) or unsigned,
which means they have a range of -32768 to
32767 and 0 to 65535, respectively.


Type modifiers: signed, unsigned, short, long

A type modifier alters the meaning of the base
type to yield a new type. Each of the above
can be applied to the base type int. The
modifiers signed and unsigned can be applied
```

```
to the base type char. In addition, long can
be applied to double. When the base type is
ommitted from a declaration, int is assumed.

Examples:
long              x;  /* int is implied */
unsigned char     ch;
signed int        i;  /* signed is default */
unsigned long int l;  /* int ok, not needed */
```

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

PREPROCESSOR STATEMENTS

The **define** statement is used to make programs more readable. Consider the following examples,

```
        #define TRUE    1    /* Don't use a semi-colon , # must be first character on line */
        #define FALSE   0
        #define NULL    0
        #define AND     &
        #define OR      |
        #define EQUALS  ==

        game_over = TRUE;
        while( list_pointer != NULL )
                ................
```

Note that preprocessor statements begin with a # symbol, and are NOT terminated by a semi-colon. Traditionally, preprocessor statements are listed at the beginning of the source file.

Preprocessor statements are handled by the compiler (or preprocessor) before the program is actually compiled. All # statements are processed first, and the symbols (like TRUE) which occur in the C program are replaced by their value (like 1). Once this substitution has taken place by the preprocessor, the program is then compiled.

In general, preprocessor constants are written in **UPPERCASE**.

Click here for more information of preprocessor statements, including macros.

Class Exercise C4

Use pre-processor statements to replace the following constants

```
        0.312
        W
        37
```

Click here for answers

LITERAL SUBSTITUTION OF SYMBOLIC CONSTANTS USING #define

Lets now examine a few examples of using these symbolic constants in our programs. Consider the following program which defines a constant called TAX_RATE.

```
        #include <stdio.h>

        #define TAX_RATE  0.10

        main()
        {
                float balance;
                float tax;
```

```
        balance = 72.10;
        tax = balance * TAX_RATE;
        printf("The tax on %.2f is %.2f\n", balance, tax );
}
```

The pre-processor first replaces all symbolic constants before the program is compiled, so after preprocessing the file (and before its compiled), it now looks like,

```
#include <stdio.h>

#define TAX_RATE  0.10

main()
{
        float balance;
        float tax;

        balance = 72.10;
        tax = balance * 0.10;
        printf("The tax on %.2f is %.2f\n", balance, tax );
}
```

YOU CANNOT ASSIGN VALUES TO THE SYMBOLIC CONSTANTS
Considering the above program as an example, look at the changes we have made below. We have added a statement which tries to change the TAX_RATE to a new value.

```
#include <stdio.h>

#define TAX_RATE  0.10

main()
{
        float balance;
        float tax;

        balance = 72.10;
        TAX_RATE = 0.15;
        tax = balance * TAX_RATE;
        printf("The tax on %.2f is %.2f\n", balance, tax );
}
```

This is **illegal**. You cannot re-assign a new value to a symbolic constant.

ITS LITERAL SUBSTITUTION, SO BEWARE OF ERRORS
As shown above, the preprocessor performs literal substitution of symbolic constants. Lets modify the previous program slightly, and introduce an error to highlight a problem.

```
#include <stdio.h>

#define TAX_RATE  0.10;
```

```
main()
{
        float balance;
        float tax;

        balance = 72.10;
        tax = (balance * TAX_RATE )+ 10.02;
        printf("The tax on %.2f is %.2f\n", balance, tax );
}
```

In this case, the error that has been introduced is that the *#define* is terminated with a semi-colon. The preprocessor performs the substitution and the offending line (which is flagged as an error by the compiler) looks like

```
        tax = (balance * 0.10; )+ 10.02;
```

However, you do not see the output of the preprocessor. If you are using TURBO C, you will only see

```
        tax = (balance * TAX_RATE )+ 10.02;
```

flagged as an error, and this actually looks okay (but its not! after substitution takes place).

MAKING PROGRAMS EASY TO MAINTAIN BY USING #define
The whole point of using *#define* in your programs is to make them easier to read and modify. Considering the above programs as examples, what changes would you need to make if the TAX_RATE was changed to 20%.

Obviously, the answer is once, where the *#define* statement which declares the symbolic constant and its value occurs. You would change it to read

```
#define TAX_RATE = 0.20
```

Without the use of symbolic constants, you would hard code the value 0.20 in your program, and this might occur several times (or tens of times).

This would make changes difficult, because you would need to search and replace every occurrence in the program. However, as the programs get larger, **what would happen if you actually used the value 0.20 in a calculation that had nothing to do with the TAX_RATE!**

SUMMARY OF #define

- allow the use of symbolic constants in programs
- in general, symbols are written in uppercase
- are not terminated with a semi-colon
- generally occur at the beginning of the file
- each occurrence of the symbol is replaced by its value
- makes programs readable and easy to maintain

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

CLASS EXERCISE C4

Use pre-processor statements to replace the following constants

```
0.312
W
37
```

```
#define smallvalue  0.312
#define letter      'W'
#define smallint    37
```

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

INDEX | Previous | Next

HEADER FILES
Header files contain definitions of functions and variables which can be incorporated into any C program by using the pre-processor *#include* statement. Standard header files are provided with each compiler, and cover a range of areas, string handling, mathematical, data conversion, printing and reading of variables.

To use any of the standard functions, the appropriate header file should be included. This is done at the beginning of the C source file. For example, to use the function *printf()* in a program, the line

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

should be at the beginning of the source file, because the definition for *printf()* is found in the file *stdio.h* All header files have the extension .h and generally reside in the /include subdirectory.

```
        #include <stdio.h>
        #include "mydecls.h"
```

The use of angle brackets <> informs the compiler to search the compilers include directory for the specified file. The use of the double quotes "" around the filename inform the compiler to search in the current directory for the specified file.

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX | Previous | Next

C Programming

INDEX Previous Next

Practise Exercise 1: Defining Variables

JavaScript compatible inter-active version of this test.

1. Declare an integer called sum

2. Declare a character called letter

3. Define a constant called TRUE which has a value of 1

4. Declare a variable called money which can be used to hold currency

5. Declare a variable called arctan which will hold scientific notation values (+e)

6. Declare an integer variable called total and initialise it to zero.

7. Declare a variable called loop, which can hold an integer value.

8. Define a constant called GST with a value of .125

Answers

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

Previous

Answers to Practise Exercise 1: Defining Variables

1. Declare an integer called sum

```
int sum;
```

2. Declare a character called letter

```
char letter;
```

3. Define a constant called TRUE which has a value of 1

```
#define TRUE 1
```

4. Declare a variable called money which can be used to hold currency

```
float money;
```

5. Declare a variable called arctan which will hold scientific notation values (+e)

```
double arctan;
```

6. Declare an integer variable called total and initialise it to zero.

```
int total;
total = 0;
```

7. Declare a variable called loop, which can hold an integer value.

```
int loop;
```

8. Define a constant called GST with a value of .125

```
#define GST 0.125
```

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

INDEX Previous Next

Practise Exercise 1: Defining Variables

Only use this if you have a JavaScript compatible browser

1. The statement that correctly defines an integer called *sum* is

- sum : integer;
- integer sum;
- int sum;
- sum int;

2. The statement that correctly defines a character called *letter* is

- letter := char;
- char letter;
- letter : char;
- character letter;

3. The correct define statement for a constant called **TRUE**, which has a value of 1 is

- int TRUE = 1;
- #define TRUE = 1
- #define TRUE 1;
- #define TRUE 1

4. The correct definition for a variable called *money* which can be used to hold currency, is

- money : real;
- real money;
- float money;
- money float;

5. The correct definition of a variable called *arctan* which will hold scientific notation values (+e), is

- arctan : float;
- real arctan;
- double arctan;
- arctan float;

6. The correct definition of an integer variable called *total* initialized to zero, is

- total : integer = 0;

- total = 0, int;
- int total = 0;
- int = 0, total;

7. The correct definition of a variable called *loop*, which can hold an integer value, is

- loop : integer;
- integer loop;
- int loop;
- loop int;

8. The correct define statement for a constant called **GST** with a value of .125, is

- #define GST 0.125
- GST .125;
- float GST=0.125;
- #define GST .125;

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

ARITHMETIC OPERATORS

The symbols of the arithmetic operators are:-

Operation	Operator	Comment	Value of Sum before	Value of sum after
Multiply	*	sum = sum * 2;	4	8
Divide	/	sum = sum / 2;	4	2
Addition	+	sum = sum + 2;	4	6
Subtraction	-	sum = sum -2;	4	2
Increment	++	++sum;	4	5
Decrement	--	--sum;	4	3
Modulus	%	sum = sum % 3;	4	1

The following code fragment adds the variables *loop* and *count* together, leaving the result in the variable *sum*

```
sum = loop + count;
```

Note: If the modulus **%** sign is needed to be displayed as part of a text string, use two, ie %%

```
#include <stdio.h>

main()
{
        int sum = 50;
        float modulus;

        modulus = sum % 10;
        printf("The %% of %d by 10 is %f\n", sum, modulus);
}
```

```
Sample Program Output
The % of 50 by 10 is 0.000000
```

CLASS EXERCISE C5

What does the following change do to the printed output of the previous program?

```
printf("The %% of %d by 10 is %.2f\n", sum, modulus);
```

Answers

Increment

The increment operator adds one to the value of the variable. The following code fragment (part of a program) adds one to the value of count, so that after the statement is executed, count has a value of 5.

```
int count = 4;
count++;
```

Decrement

The decrement operator subtracts one from the value of the variable. The following code fragment (part of a program) subtracts one from the value of count, so that after the statement is executed, count has a value of 3.

```
int count = 4;
count--;
```

Modulus

The modulus operator assigns the remainder left over after a division the value of the variable. The following code fragment (part of a program) uses the modulus operator to calculate the modulus of 20 % 3. To work this out, divide 20 by 3. Now 3 divides into 20 six times, with a remainder left over of 2. So the value 2 (the remainder) is assigned to count.

```
int count;
count = 20 % 3;
```

© Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

Previous

ANSWERS: CLASS EXERCISE C5

```
#include <stdio.h>

main()
{
        int sum = 50;
        float modulus;

        modulus = sum % 10;
        printf("The %% of %d by 10 is %.2f\n", sum, modulus);
}


The % of 50 by 10 is 0.00
_
```

©Copyright B Brown. 1984-2000. All rights reserved.

Previous

C Programming

INDEX | Previous | Next

Practise Exercise 2: Assignments

JavaScript compatible inter-active version of this test.

1. Assign the value of the variable number1 to the variable total

2. Assign the sum of the two variables loop_count and petrol_cost to the variable sum

3. Divide the variable total by the value 10 and leave the result in the variable discount

4. Assign the character W to the char variable letter

5. Assign the result of dividing the integer variable sum by 3 into the float variable costing. Use type casting to ensure that the remainder is also held by the float variable.

Answers

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX | Previous | Next

C Programming

Previous

Answers: Practise Exercise 2: Assignments

1. Assign the value of the variable number1 to the variable total

```
total = number1;
```

2. Assign the sum of the two variables loop_count and petrol_cost to the variable sum

```
sum = loop_count + petrol_cost;
```

3. Divide the variable total by the value 10 and leave the result in the variable discount

```
discount = total / 10;
```

4. Assign the character W to the char variable letter

```
letter = 'W';
```

5. Assign the result of dividing the integer variable sum by 3 into the float variable costing. Use type casting to ensure that the remainder is also held by the float variable.

```
costing = (float) sum / 3;
```

©Copyright B Brown. 1984-2000. All rights reserved.

Previous

C Programming

INDEX | Previous | Next

Practise Exercise 2: Assignments

To run this test requires a JavaScript enabled browser

1. The statement which correctly assigns the value of the variable *number1* to the variable *total*, is

- total := number1;
- number1 = total;
- total = number1;
- number1 := total;

2. The statement that correctly assigns the sum of the two variables *loop_count* and *petrol_cost* to the variable *sum*, is

- loop_count = sum + petrol_cost;
- petrol_cost = sum - loop_count;
- sum = petrol_cost / loop_count;
- sum = loop_count + petrol_cost;

3. The correct statement which divides the variable *total* by the value 10 and leaves the result in the variable *discount*, is

- discount = total / 10;
- discount = 10 / total;
- total = discount / 10;
- total = 10 / discount;

4. The correct statement which assigns the character *W* to the char variable *letter*, is

- letter = "W";
- letter = 'W';
- char letter = "W";
- strcpy(letter, "W");

5. The correct statement which assign the decimal result of dividing the integer variable sum by 3 into the float variable costing, is (Use type casting to ensure that floating point division is performed)

Given: int sum = 7; float costing;

- (float) costing = sum / 3;
- costing = (float) (sum / 3);

- ○ costing = (float) sum / 3;
- ○ costing = float (sum / 3);

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX | Previous | Next

C Programming

INDEX Previous Next

PRE/POST INCREMENT/DECREMENT OPERATORS
PRE means do the operation first followed by any assignment operation. POST means do the operation after any assignment operation. Consider the following statements

```
++count;         /* PRE Increment, means add one to count */
count++;         /* POST Increment, means add one to count */
```

In the above example, because the value of *count* is not assigned to any variable, the effects of the PRE/POST operation are not clearly visible.

Lets examine what happens when we use the operator along with an assignment operation. Consider the following program,

```
#include <stdio.h>

main()
{
        int count = 0, loop;

        loop = ++count;  /* same as count = count + 1; loop = count;  */
        printf("loop = %d, count = %d\n", loop, count);

        loop = count++;  /* same as loop = count;  count = count + 1;  */
        printf("loop = %d, count = %d\n", loop, count);
}
```

```
Sample Program Output
loop = 1, count = 1
loop = 1; count = 2
```

If the operator precedes (is on the left hand side) of the variable, the operation is performed first, so the statement

```
loop = ++count;
```

really means increment *count* first, then assign the new value of *count* to *loop*.

Which way do you write it?
Where the increment/decrement operation is used to adjust the value of a variable, and is not involved in an assignment operation, which should you use,

```
        ++loop_count;
or
        loop_count++;
```

The answer is, it really does not matter. It does seem that there is a preference amongst C programmers to use the post form.

Something to watch out for
Whilst we are on the subject, do not get into the habit of using a space(s) between the variable name and the pre/post operator.

```
loop_count ++;
```

Try to be explicit in *binding* the operator tightly by leaving no gap.

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

INDEX Previous Next

GOOD FORM

Perhaps we should say *programming style* or *readability*. The most common complaints we would have about beginning C programmers can be summarized as,

- they have poor layout
- their programs are hard to read

Your programs will be quicker to write and easier to debug if you get into the habit of actually formatting the layout correctly as you write it.

For instance, look at the program below

```
#include<stdio.h>
main()
   {
    int sum,loop,kettle,job;
    char Whoknows;

         sum=9;
   loop=7;
 whoKnows='A';
printf("Whoknows=%c,kettle=%d\n",whoknows,kettle);
}
```

It is our contention that the program is hard to read, and because of this, will be difficult to debug for errors by an inexperienced programmer. It also contains a few deliberate mistakes!

Okay then, lets rewrite the program using good form.

```
#include <stdio.h>

main()
{
        int sum, loop, kettle = 0, job;
        char whoknows;

        sum = 9;
        loop = 7;
        whoknows = 'A';
        printf( "Whoknows = %c, kettle = %d\n", whoknows, kettle );
}
```

We have also corrected the mistakes. The major differences are

- the { and } braces directly line up underneath each other
  This allows us to check ident levels and ensure that statements belong to the correct block of code. This becomes vital as programs become more complex
- spaces are inserted for readability
  We as humans write sentences using spaces between words. This helps our comprehension of what we read (if you dont believe me, try reading the following sentence. wishihadadollarforeverytimeimadeamistake. The insertion of spaces will also help us identify mistakes quicker.
- good indentation
  Indent levels (tab stops) are clearly used to block statements, here we clearly see and identify functions, and the statements which belong to each { } program body.
- initialization of variables
  The first example prints out the value of *kettle*, a variable that has no initial value. This is corrected in the second example.

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

KEYBOARD INPUT

There is a function in C which allows the programmer to accept input from a keyboard. The following program illustrates the use of this function,

```
#include <stdio.h>

main()     /* program which introduces keyboard input */
{
        int  number;

        printf("Type in a number \n");
        scanf("%d", &number);
        printf("The number you typed was %d\n", number);
}
```

```
Sample Program Output
Type in a number
23
The number you typed was 23
```

An integer called *number* is defined. A prompt to enter in a number is then printed using the statement

```
printf("Type in a number \n:");
```

The *scanf* routine, which accepts the response, has two arguments. The first ("%d") specifies what type of data type is expected (ie char, int, or float). List of formatters for scanf() found here.

The second argument (&number) specifies the variable into which the typed response will be placed. In this case the response will be placed into the memory location associated with the variable *number*.

This explains the special significance of the & character (which means the address of).

Sample program illustrating use of scanf() to read integers, characters and floats

```
#include < stdio.h >

main()
{
        int sum;
        char letter;
        float money;

        printf("Please enter an integer value ");
        scanf("%d", &sum );
```

```
                printf("Please enter a character ");
                /* the leading space before the %c ignores space characters in the
input */
                scanf("  %c", &letter );

                printf("Please enter a float variable ");
                scanf("%f", &money );

                printf("\nThe variables you entered were\n");
                printf("value of sum = %d\n", sum );
                printf("value of letter = %c\n", letter );
                printf("value of money = %f\n", money );
        }
```

```
Sample Program Output
Please enter an integer value
34
Please enter a character
W
Please enter a float variable
32.3
The variables you entered were
value of sum = 34
value of letter = W
value of money = 32.300000
```

This program illustrates several important points.

- the c language provides no error checking for user input. The user is expected to enter the correct data type. For instance, if a user entered a character when an integer value was expected, the program may enter an infinite loop or abort abnormally.
- its up to the programmer to validate data for correct type and range of values.

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

INDEX | Previous | Next

Practise Exercise 3: printf() and scanf()

JavaScript compatible inter-active version of this test.

1. Use a printf statement to print out the value of the integer variable sum

2. Use a printf statement to print out the text string "Welcome", followed by a newline.

3. Use a printf statement to print out the character variable letter

4. Use a printf statement to print out the float variable discount

5. Use a printf statement to print out the float variable dump using two decimal places

6. Use a scanf statement to read a decimal value from the keyboard, into the integer variable sum

7. Use a scanf statement to read a float variable into the variable discount_rate

8. Use a scanf statement to read a single character from the keyboard into the variable operator. Skip leading blanks, tabs and newline characters.

Answers

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX | Previous | Next

C Programming

Previous

Answers: Practise Exercise 3: printf() and scanf()

1. Use a printf statement to print out the value of the integer variable sum

```
printf("%d", sum);
```

2. Use a printf statement to print out the text string "Welcome", followed by a newline.

```
printf("Welcome\n");
```

3. Use a printf statement to print out the character variable letter

```
printf("%c", letter);
```

4. Use a printf statement to print out the float variable discount

```
printf("%f", discount);
```

5. Use a printf statement to print out the float variable dump using two decimal places

```
printf("%.2f", dump);
```

6. Use a scanf statement to read a decimal value from the keyboard, into the integer variable sum

```
scanf("%d", &sum);
```

7. Use a scanf statement to read a float variable into the variable discount_rate

```
scanf("%f", &discount_rate);
```

8. Use a scanf statement to read a single character from the keyboard into the variable operator. Skip leading blanks, tabs and newline characters.

```
scanf(" %c", &operator);
```

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

INDEX | Previous | Next

Practise Exercise 3: printf() and scanf()

To run this test requires a JavaScript enabled browser

1. The statement which prints out the value of the integer variable *sum*, is

- printf("%s", sum);
- print("%i", sum);
- printf("%d", sum);
- printf("%d", &sum);

2. The statement which prints out the text string "Welcome", followed by a newline, is.

- printf("Welcome\n");
- printf(Welcome, '\n');
- printf(Welcome\n);
- printf('Welcome', '\n');

3. The statement which prints out the value of the character variable *letter*, is

- printf(letter);
- printf("%c", &letter);
- printf("%d", letter);
- printf("%c", letter);

4. The statement which prints out the value of the float variable *discount*, is

- printf("%s", discount);
- print('discount');
- printf("%f", discount);
- printf("%d", discount);

5. The statement which prints out the value of the float variable *dump* using two decimal places, is

- printf("%f", dump);
- printf("%.2f", dump);
- printf("%2f", dump);
- printf("%f", &dump);

6. The statement to read a decimal value from the keyboard, into the integer variable *sum*, is

- scanf("%d", &sum);

- scanf(sum);
- scanf("%s", sum);
- scanf("%f", &sum);

7. The statement to read a float value into the variable *discount_rate* is

- scanf("%f", discount_rate);
- scanf("%d", &discount_rate);
- scanf(discount_rate);
- scanf("%f", &discount_rate);

8. The statement to read a single character from the keyboard into the variable *operator*, skipping leading blanks, tabs and newline characters, is

- scanf("%s", operator);
- scanf("%c", &operator);
- scanf(" %c", &operator);
- scanf("%c", operator);

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

INDEX | Previous | Next

THE RELATIONAL OPERATORS
These allow the comparision of two or more variables.

Operator	Meaning
==	*equal to*
!=	*not equal*
<	*less than*
<=	*less than or equal to*
>	*greater than*
>=	*greater than or equal to*

In the next few screens, these will be used in *for* loops and *if* statements.

The operator

<>

may be legal in Pascal, **but is illegal in C.**

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX | Previous | Next

C Programming

ITERATION, FOR LOOPS

The basic format of the for statement is,

```
for( start condition; continue condition; re-evaulation )
        program statement;
```

```
/* sample program using a for statement */
#include <stdio.h>

main()   /* Program introduces the for statement, counts to ten */
{
        int  count;

        for( count = 1; count <= 10; count = count + 1 )
                printf("%d ", count );

        printf("\n");
}
```

Sample Program Output

```
1 2 3 4 5 6 7 8 9 10
```

The program declares an integer variable *count*. The first part of the *for* statement

```
for( count = 1;
```

initialises the value of *count* to 1. The *for* loop continues whilst the condition

```
count <= 10;
```

evaluates as TRUE. As the variable *count* has just been initialised to 1, this condition is TRUE and so the program statement

```
        printf("%d ", count );
```

is executed, which prints the value of *count* to the screen, followed by a space character.

Next, the remaining statement of the *for* is executed

```
count = count + 1 );
```

which adds one to the current value of *count*. Control now passes back to the conditional test,

```
count <= 10;
```

which evaluates as true, so the program statement

```
        printf("%d ", count );
```

is executed. *Count* is incremented again, the condition re-evaluated etc, until count reaches a value of 11.

When this occurs, the conditional test

```
count <= 10;
```

evaluates as FALSE, and the *for* loop terminates, and program control passes to the statement

```
printf("\n");
```

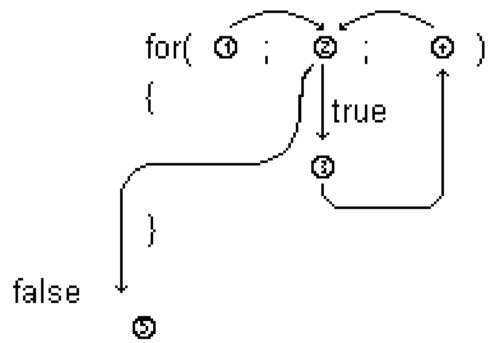
which prints a newline, and then the program terminates, as there are no more statements left to execute.

```
/* sample program using a for statement */
#include <stdio.h>

main()
{
        int  n, t_number;

        t_number = 0;
        for( n = 1; n <= 200; n = n + 1 )
                t_number = t_number + n;

        printf("The 200th triangular_number is %d\n", t_number);
}
```

```
Sample Program Output
The 200th triangular_number is 20100
```

The above program uses a *for* loop to calculate the sum of the numbers from 1 to 200 inclusive (said to be the *triangular number*).

The following diagram shows the order of processing each part of a *for*



An example of using a for loop to print out characters

```
#include <stdio.h>

main()
{
        char letter;
        for( letter = 'A'; letter <= 'E'; letter = letter + 1 ) {
                printf("%c ", letter);
        }
}
```

```
Sample Program Output
A B C D E
```

An example of using a for loop to count numbers, using two initialisations

```
#include <stdio.h>

main()
{
        int total, loop;
        for( total = 0, loop = 1; loop <= 10; loop = loop + 1 ){
                total = total + loop;
        }
        printf("Total = %d\n", total );
}
```

```
Sample Program Output
Total = 55
```

In the above example, the variable *total* is initialised to 0 as the first part of the for loop. The two statements,

```
for( total = 0, loop = 1;
```

are part of the initialisation. This illustrates that more than one statement is allowed, as long as they are separated by **commas**.

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

INDEX Previous Next

Graphical Animation of *for* loop
To demonstrate the operation of the *for* statement, lets consider a series of animations.

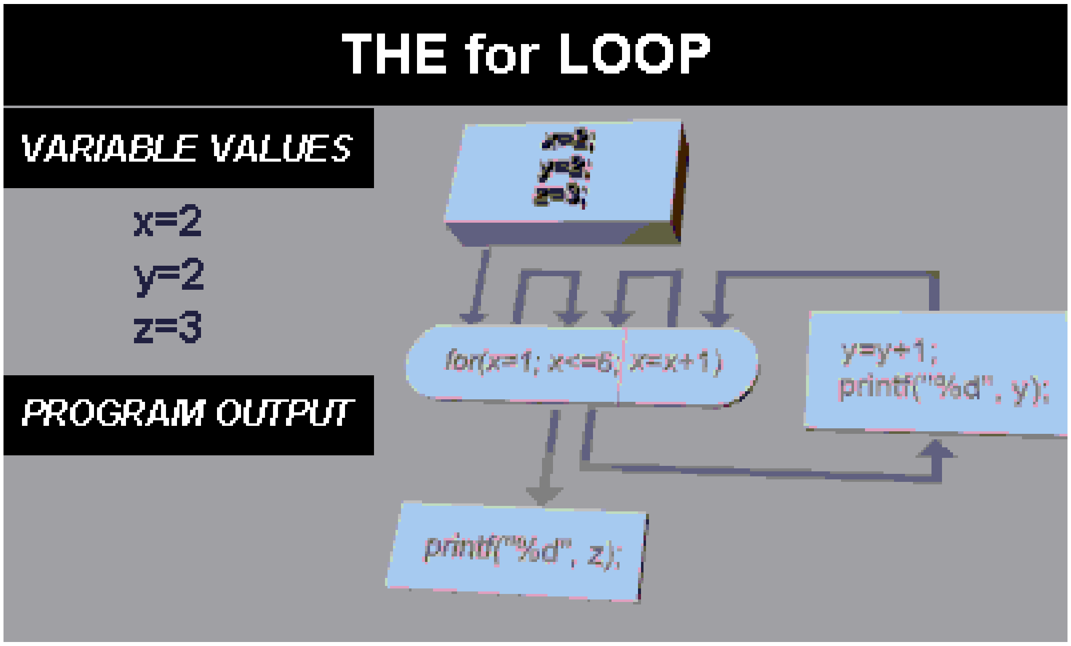
The code we will be using is

```
#include <stdio.h>

main() {
        int x, y, z;

        x = 2;
        y = 2;
        z = 3;

        for( x = 1; x <= 6; x = x + 1 ) {
                printf("%d", y );
                y = y + 1;
        }
        printf("\n%d", z );
}
```

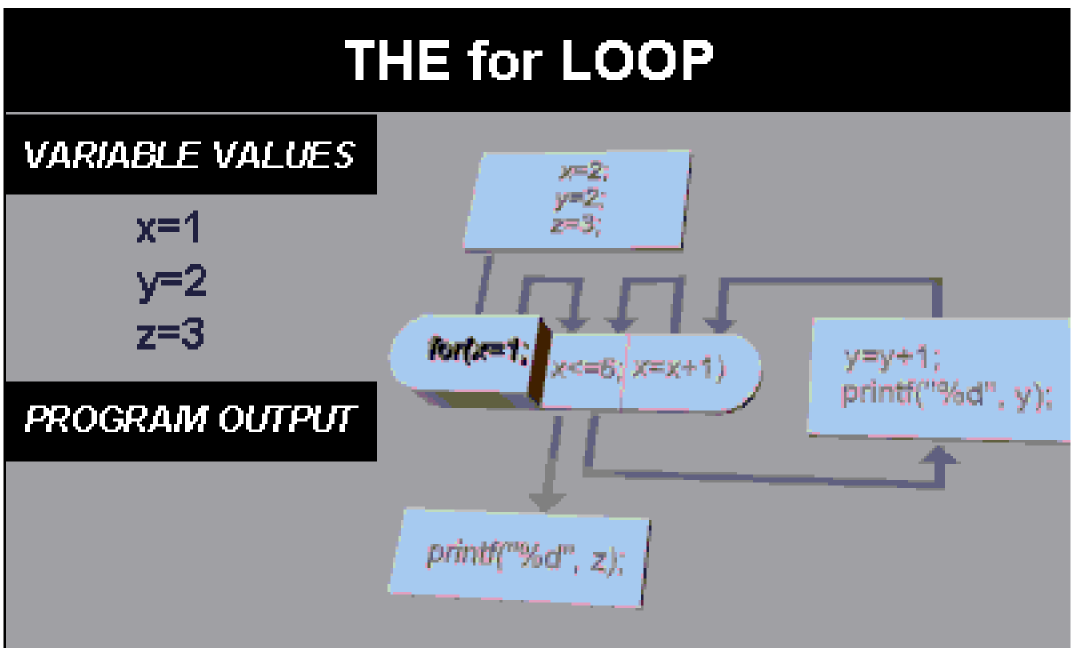
Sample Program Output

```
2 3 4 5 6 7
3
```

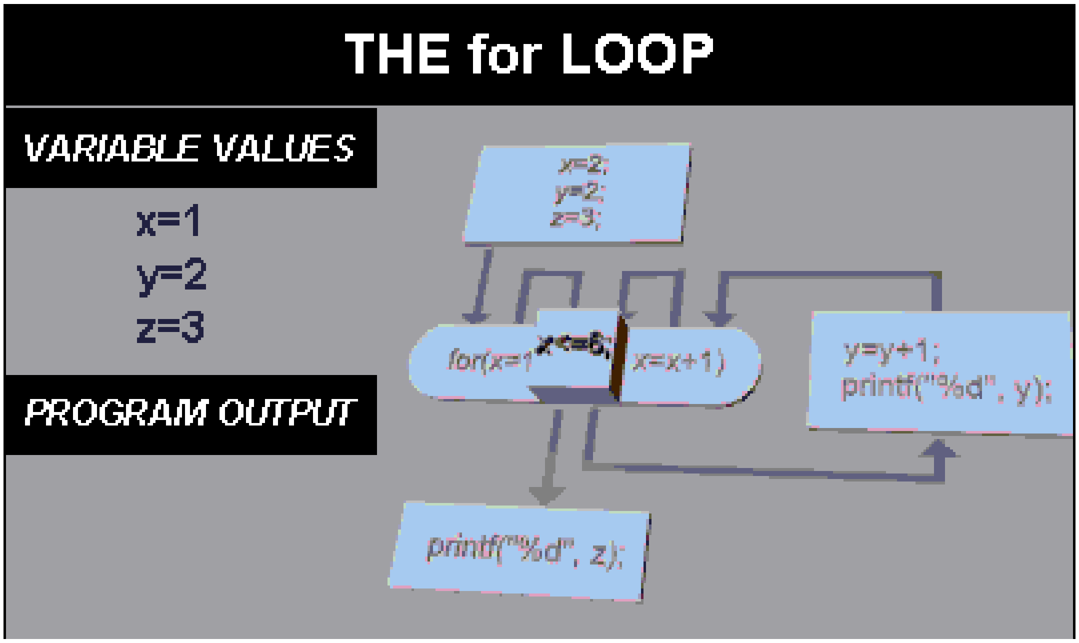
The following diagram shows the initial state of the program, after the initialization of the variables *x*, *y*, and *z*.



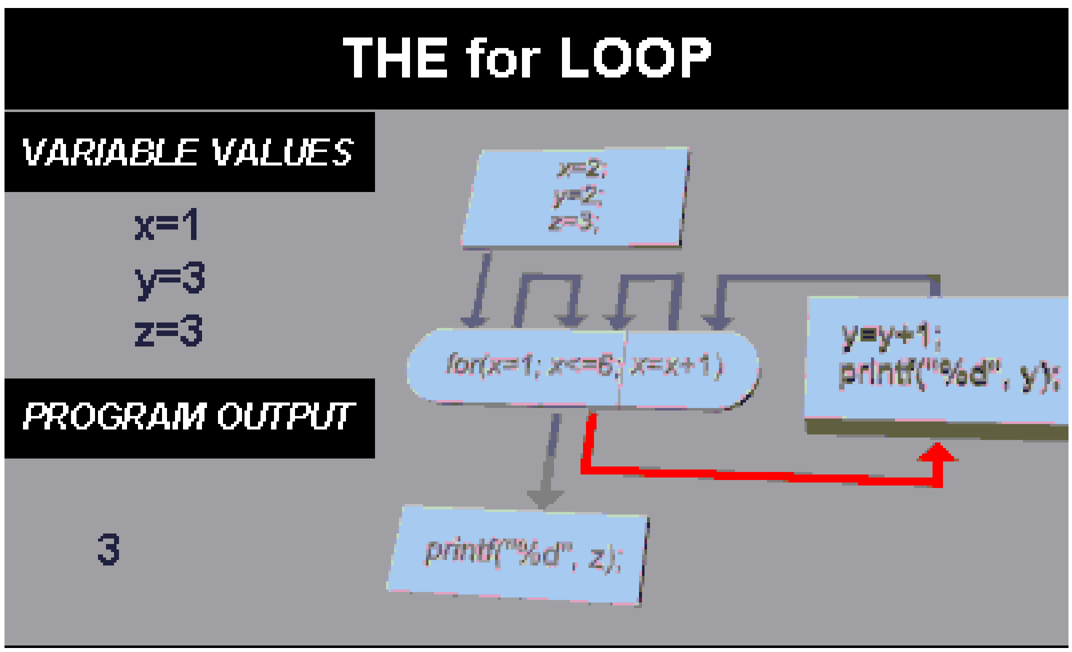
On entry to the *for* statement, the first expression is executed, which in our example assigns the value 1 to *x*. This can be seen in the graphic shown below (Note: see the Variable Values: section)



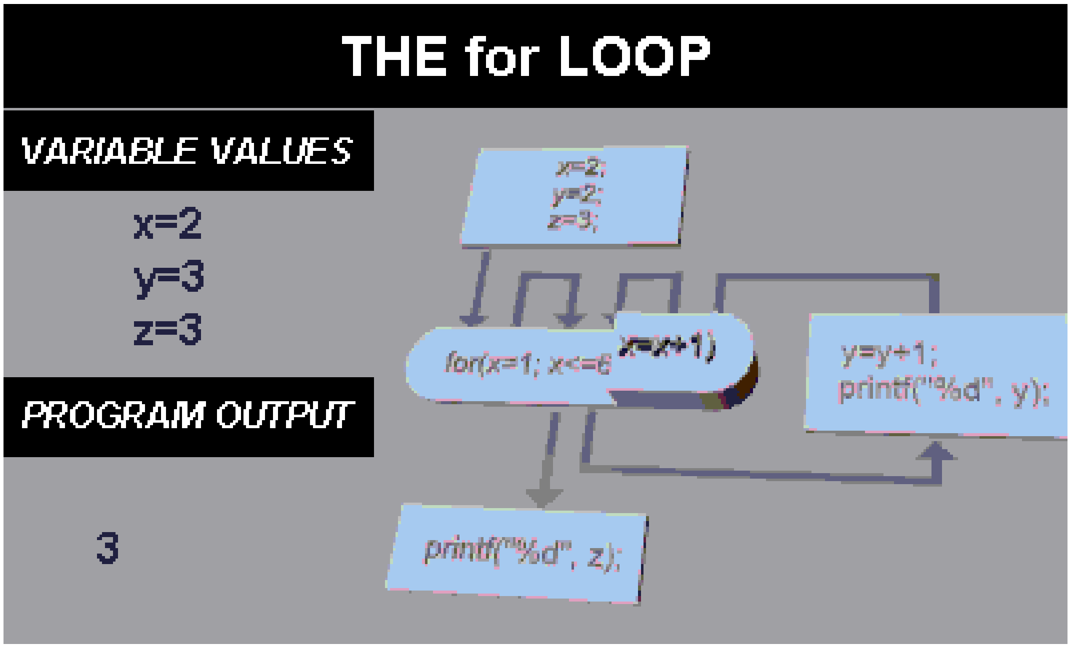
The next part of the *for* is executed, which tests the value of the loop variable *x* against the constant **6**.



It can be seen from the variable window that *x* has a current value of 1, so the test is successful, and program flow branches to execute the statements of the *for body*, which prints out the value of *y*, then adds 1 to *y*. You can see the program output and the state of the variables shown in the graphic below.

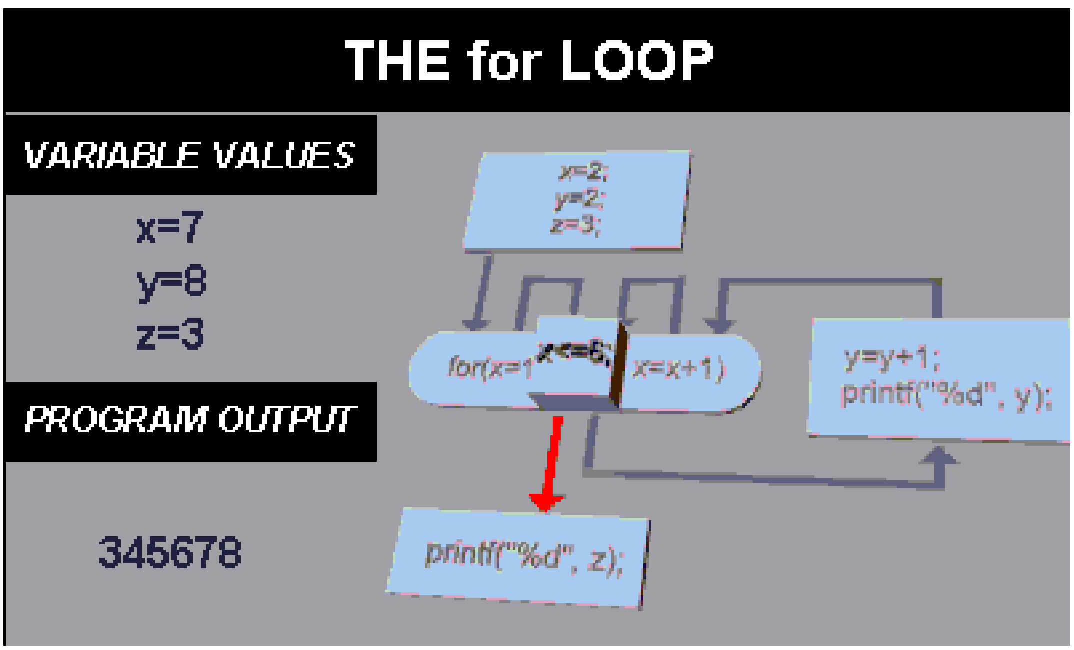


After executing the statements of the *for body*, execution returns to the last part of the *for* statement. Here, the value of *x* is incremented by 1. This is seen by the value of *x* changing to 2.



Next, the condition of the *for* variable is tested again. It continues because the value of it (2) is less than 6, so the body of the loop is executed again.

Execution continues till the value of *x* reaches 7. Lets now jump ahead in the animation to see this. Here, the condition test will fail, and the *for* statement finishes, passing control to the statement which follows.



Play "for" animation [AVI, 4.4MB]
Play "for" animation [Real Video, 740KB]

© Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

This video shows a graphical animation of how a for loop works in C.

Previous

C Programming

INDEX Previous Next

EXERCISE C6
Rewrite the previous program by calculating the 200th triangular number, and make the program shorter (if possible).

CLASS EXERCISE C7
What is the difference between the two statements,

```
a == 2
a = 2
```

CLASS EXERCISE C8
Change the printf line of the above program to the following,

```
printf(" %2d            %2d\n",n,t_number);
```

What does the inclusion of the 2 in the %d statements achieve?

EXERCISE C9
Create a C program which calculates the triangular number of the users request, read from the keyboard using **scanf()**. A triangular number is the sum of the preceding numbers, so the triangular number 7 has a value of

7 + 6 + 5 + 4 + 3 + 2 + 1

Answers

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

Previous

Answer: EXERCISE C6

```
#include <stdio.h>

main()
{
        int  n = 1, t_number = 0;

        for( ; n <= 200; n++ )
                t_number = t_number + n;

        printf("The 200th triangular_number is %d\n", t_number);
}
```

Answer: CLASS EXERCISE C7

```
a == 2          equality test
a = 2           assignment
```

Answer: CLASS EXERCISE C8
The inclusion of the 2 in the %d statements achieves a field width of two places, and prints a leading 0 where the value is less than 10.

Answer: EXERCISE C9

```
#include <stdio.h>

main()
{
        int  n = 1, t_number = 0, input;

        printf("Enter a number\n");
        scanf("%d", &input);
        for( ; n <= input; n++ )
                t_number = t_number + n;

        printf("The triangular_number of %d is %d\n", input, t_number);
}
```

©Copyright B Brown. 1984-2000. All rights reserved.

Previous

C Programming

Practise Exercise 4: for loops

JavaScript compatible inter-active version of this test.

1. Write a for loop to print out the values 1 to 10 on separate lines.

2. Write a for loop which will produce the following output (hint: use two nested for loops)

```
1
22
333
4444
55555
```

3. Write a for loop which sums all values between 10 and 100 into a variable called *total*. Assume that *total* has NOT been initialized to zero.

4. Write a for loop to print out the character set from A-Z.

Answers

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

PRACTISE EXERCISE 4

for loops

1. Write a for loop to print out the values 1 to 10 on separate lines.

```
for( loop = 1; loop <= 10; loop = loop + 1 )
        printf("%d\n", loop) ;
```

2. Write a for loop which will produce the following output (hint: use two nested for loops)

```
1
22
333
4444
55555
```

```
for( loop = 1; loop <= 5; loop = loop + 1 )
{
        for( count = 1; count <= loop; count  = count + 1 )
                printf("%d", loop );
        printf("\n");
}
```

3. Write a for loop which sums all values between 10 and 100 into a variable called *total*. Assume that *total* has NOT been initialized to zero.

```
for( loop = 10, total = 0; loop <= 100; loop = loop + 1 )
        total = total + loop;
```

4. Write a for loop to print out the character set from A-Z.

```
for( ch = 'A'; ch <= 'Z'; ch = ch + 1 )
        printf("%c", ch );
printf("\n");
```

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

INDEX | Previous | Next

PRACTISE EXERCISE 4

for loops

To run this test requires a JavaScript enabled browser

1. The statement which prints out the values 1 to 10 on separate lines, is

○ **Statement 1**

```
for( count = 1; count <= 10; count = count + 1)
   printf("%d\n", count);
```

○ **Statement 2**

```
for( count = 1; count < 10; count = count + 1)
   printf("%d\n", count);
```

○ **Statement 3**

```
for( count = 0; count <= 9; count = count + 1)
   printf("%d ", count);
```

○ **Statement 4**

```
for( count = 1; count <> 10; count = count + 1)
   printf("%d\n", count);
```

---

2. The statement which produces the following output is, (hint: use two nested for loops)

```
1
22
333
4444
55555
```

○ **Statement 1**

```
for(a = 1; a <= 5; a = a + 1) {
```

```
   for( b = 1; b <= 5; b = b + 1)
     printf("%d", b);
   printf("\n");
}
```

Statement 2

```
for( a = 1; a <= 5; a = a + 1) {
   for( b = 1; b <= a; b = b + 1)
      printf("%d", a);
   printf("\n");
}
```

Statement 3

```
for( a = 1; a <= 5; a = a + 1) {
   for( b = a; b <= 5; b = b + 1)
      printf("%d", b);
   printf("\n");
}
```

Statement 4

```
for( a = 1; a <= 5; a = a + 1) {
   for( b = 1; b < a; b = b + a)
      printf("%d", b);
   printf("\n");
}
```

---

3. The statement which sums all values between 10 and 100 into a variable called *total* is, assuming that *total* has NOT been initialised to zero.

Statement 1

```
for( a = 10; a <= 100; a = a + 1)
   total = total + a;
```

Statement 2

```
for( a = 10; a < 100; a = a + 1, total = 0)
   total = total + a;
```

Statement 3

```
for( a = 10; a <= 100, total = 0; a = a + 1)
   total = total + a;
```

Statement 4

```
for( a = 10, total = 0; a <= 100; a = a + 1)
   total = total + a;
```

---

4. The statement that prints out the character set from A-Z, is

○ **Statement 1**

```
for( a = 'A'; a < 'Z'; a = a + 1)
   printf("%c", a);
```

○ **Statement 2**

```
for( a = 'a'; a <= 'z'; a = a + 1)
   printf("%c", &a);
```

○ **Statement 3**

```
for( a = 'A'; a <= 'Z'; a = a + 1)
   printf("%c", a);
```

○ **Statement 4**

```
for( a = 'Z'; a <= 'A'; a = a + 1)
   printf("%c", a);
```

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

INDEX | Previous | Next

THE WHILE STATEMENT
The *while* provides a mechanism for repeating C statements whilst a condition is true. Its format is,

```
	while( condition )
		program statement;
```

Somewhere within the body of the *while* loop a statement must alter the value of the condition to allow the loop to finish.

```
	/* Sample program including while  */
	#include <stdio.h>

	main()
	{
		int  loop = 0;

		while( loop <= 10 ) {
			printf("%d\n", loop);
			++loop;
		}
	}


	Sample Program Output
	0
	1
	...
	10
```

The above program uses a *while* loop to repeat the statements

```
	printf("%d\n", loop);
	++loop;
```

whilst the value of the variable *loop* is less than or equal to 10.

Note how the variable upon which the *while* is dependant is initialised prior to the *while* statement (in this case the previous line), and also that the value of the variable is altered within the loop, so that eventually the conditional test will succeed and the *while* loop will terminate.

This program is functionally equivalent to the earlier *for* program which counted to ten.

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

INDEX | Previous | Next

THE DO WHILE STATEMENT

The *do { } while* statement allows a loop to continue whilst a condition evaluates as TRUE (non-zero). The loop is executed as least once.

```
        /* Demonstration of DO...WHILE    */
        #include <stdio.h>

        main()
        {
                int  value, r_digit;

                printf("Enter the number to be reversed.\n");
                scanf("%d", &value);
                do {
                        r_digit = value % 10;
                        printf("%d", r_digit);
                        value = value / 10;
                } while( value != 0 );
                printf("\n");
        }
```

The above program reverses a number that is entered by the user. It does this by using the modulus *%* operator to extract the right most digit into the variable *r_digit*. The original number is then divided by 10, and the operation repeated whilst the number is not equal to 0.

It is our contention that this programming construct is improper and should be avoided. It has potential problems, and you should be aware of these.

One such problem is deemed to be *lack of control*. Considering the above program code portion,

```
                do {
                        r_digit = value % 10;
                        printf("%d", r_digit);
                        value = value / 10;
                } while( value != 0 );
```

there is **NO** choice whether to execute the loop. Entry to the loop is automatic, as you only get a choice to continue.

Another problem is that the loop is always executed at least once. This is a by-product of the lack of control. This means its possible to enter a *do { } while* loop with invalid data.

Beginner programmers can easily get into a whole heap of trouble, so our advice is to avoid its use. This is the only time that you will encounter it in this course. Its easy to avoid the use of this construct by replacing it with the following algorithms,

```
initialise loop control variable
while( loop control variable is valid ) {
        process data
        adjust control variable if necessary
}
```

Okay, lets now rewrite the above example to remove the *do { } while* construct.

```
/* rewritten code to remove construct */
#include <stdio.h>

main()
{
        int  value, r_digit;

        value = 0;
        while( value <= 0 ) {
                printf("Enter the number to be reversed.\n");
                scanf("%d", &value);
                if( value <= 0 )
                        printf("The number must be positive\n");
        }

        while( value != 0 )
        {
                r_digit = value % 10;
                printf("%d", r_digit);
                value = value / 10;
        }
        printf("\n");
}
```

Sample Program Output

```
Enter the number to be reversed.
-43
The number must be positive
Enter the number to be reversed.
423
324
```

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

MAKING DECISIONS

INDEX Previous Next

SELECTION (IF STATEMENTS)
The *if* statements allows branching (decision making) depending upon the value or state of variables. This allows statements to be executed or skipped, depending upon decisions. The basic format is,

```
if( expression )
        program statement;
```

Example;

```
if( students < 65 )
        ++student_count;
```

In the above example, the variable *student_count* is incremented by one only if the value of the integer variable *students* is less than 65.

The following program uses an *if* statement to validate the users input to be in the range 1-10.

```
#include <stdio.h>

main()
{
        int number;
        int valid = 0;

        while( valid == 0 ) {
                printf("Enter a number between 1 and 10 -->");
                scanf("%d", &number);
                /* assume number is valid */
                valid = 1;
                if( number < 1 ) {
                        printf("Number is below 1. Please re-enter\n");
                        valid = 0;
                }
                if( number > 10 ) {
                        printf("Number is above 10. Please re-enter\n");
                        valid = 0;
                }
        }
        printf("The number is %d\n", number );
}
```

```
Sample Program Output
Enter a number between 1 and 10 --> -78
Number is below 1. Please re-enter
Enter a number between 1 and 10 --> 4
The number is 4
```

EXERCISE C10
Write a C program that allows the user to enter in 5 grades, ie, marks between 0 - 100. The program must calculate the average mark, and state the number of marks less than 65.

Answer

Consider the following program which determines whether a character entered from the keyboard is within the range A to Z.

```
#include <stdio.h>

main()
{
        char letter;

        printf("Enter a character -->");
        scanf(" %c", &letter );

        if( letter >= 'A' ) {
                if( letter <= 'Z' )
                        printf("The character is within A to Z\n");
        }
}
```

```
Sample Program Output
Enter a character --> C
The character is within A to Z
```

The program does not print any output if the character entered is not within the range A to Z. This can be addressed on the following pages with the *if else* construct.

Please note use of the leading space in the statement (before %c)

```
scanf(" %c", &letter );
```

This enables the skipping of leading TABS, Spaces, (collectively called whitespaces) and the ENTER KEY. If the leading space was not used, then the first entered character would be used, and *scanf* would not ignore the whitespace characters.

COMPARING float types FOR EQUALITY
Because of the way in which float types are stored, it makes it very difficult to compare float types for equality. Avoid trying to compare float variables for equality, or you may encounter unpredictable results.

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

Exercise C10: Answer

Write a C program that allows the user to enter in 5 grades, ie, marks between 1 - 100. The program must calculate the average mark, and state the number of marks less than 65.

```
#include <stdio.h>

main()
{
        int grade;      /* to hold the entered grade */
        float average;  /* the average mark */
        int loop;       /* loop count */
        int sum;        /* running total of all entered grades */
        int valid_entry;        /* for validation of entered grade */
        int failures;   /* number of people with less than 65 */

        sum = 0;        /* initialise running total to 0 */
        failures = 0;

        for( loop = 0; loop < 5; loop = loop + 1 )
        {
                valid_entry = 0;
                while( valid_entry == 0 )
                {
                        printf("Enter mark (1-100):");
                        scanf(" %d", &grade );
                        if ((grade > 1 ) {
                                if( grade < 100 )
                                     valid_entry = 1;
                        }
                }
                if( grade < 65 )
                        failures++;
                sum = sum + grade;
        }
        average = (float) sum / loop;
        printf("The average mark was %.2f\n", average );
        printf("The number less than 65 was %d\n", failures );
}
```

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

INDEX | Previous | Next

if else
The general format for these are,

```
if( condition 1 )
   statement1;
else if( condition 2 )
   statement2;
else if( condition 3 )
   statement3;
else
   statement4;
```

The *else* clause allows action to be taken where the condition evaluates as false (zero).

The following program uses an *if else* statement to validate the users input to be in the range 1-10.

```
#include <stdio.h>

main()
{
        int number;
        int valid = 0;

        while( valid == 0 ) {
                printf("Enter a number between 1 and 10 -->");
                scanf("%d", &number);
                if( number < 1 ) {
                        printf("Number is below 1. Please re-enter\n");
                        valid = 0;
                }
                else if( number > 10 ) {
                        printf("Number is above 10. Please re-enter\n");
                        valid = 0;
                }
                else
                        valid = 1;
        }
        printf("The number is %d\n", number );
}
```

Sample Program Output

```
Enter a number between 1 and 10 --> 12
Number is above 10. Please re-enter
Enter a number between 1 and 10 --> 5
The number is 5
```

This program is slightly different from the previous example in that an *else* clause is used to set the variable *valid* to 1. In this program, the logic should be easier to follow.

```
	/* Illustates nested if else and multiple arguments to the scanf function.
*/
	#include <stdio.h>

	main()
	{
		int    invalid_operator = 0;
		char   operator;
		float  number1, number2, result;

		printf("Enter two numbers and an operator in the format\n");
		printf(" number1 operator number2\n");
		scanf("%f %c %f", &number1, &operator, &number2);

		if(operator == '*')
			result = number1 * number2;
		else if(operator == '/')
			result = number1 / number2;
		else if(operator == '+')
			result = number1 + number2;
		else if(operator == '-')
			result = number1 - number2;
		else
			invalid_operator = 1;

		if( invalid_operator != 1 )
			printf("%f %c %f is %f\n", number1, operator, number2, result
);
		else
			printf("Invalid operator.\n");
	}


	Sample Program Output
	Enter two numbers and an operator in the format
	number1 operator number2
	23.2 + 12
	23.2 + 12 is 35.2
```

The above program acts as a simple calculator.

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

MAKING DECISIONS

Practise Exercise 5: while loops and if else

JavaScript compatible inter-active version of this test.

1. Use a while loop to print the integer values 1 to 10 on the screen

```
12345678910
```

2. Use a nested while loop to reproduce the following output

```
1
22
333
4444
55555
```

3. Use an if statement to compare the value of an integer called sum against the value 65, and if it is less, print the text string "Sorry, try again".

4. If total is equal to the variable good_guess, print the value of total, else print the value of good_guess.

Answers

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

MAKING DECISIONS

Answers: Practise Exercise 5: while loops and if else

1. Use a while loop to print the integer values 1 to 10 on the screen

```
12345678910
```

```
#include <stdio.h>

main()
{
        int loop;
        loop = 1;
        while( loop <= 10 ) {
                printf("%d", loop);
                loop++;
        }
        printf("\n");
}
```

2. Use a nested while loop to reproduce the following output

```
1
22
333
4444
55555
```

```
#include <stdio.h>

main()
{
        int loop;
```

```
            int count;
            loop = 1;
            while( loop <= 5 ) {
                    count = 1;
                    while( count <= loop ) {
                            printf("%d", count);
                            count++;
                    }
                    loop++;
            }
            printf("\n");
    }
```

3. Use an if statement to compare the value of an integer called sum against the value 65, and if it is less, print the text string "Sorry, try again".

```
    if( sum < 65 )
            printf("Sorry, try again.\n");
```

4. If total is equal to the variable good_guess, print the value of total, else print the value of good_guess.

```
    if( total == good_guess )
            printf("%d\n", total );
    else
            printf("%d\n", good_guess );
```

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

INDEX | Previous | Next

Practise Exercise 5: while loops and if else

To run this test requires a JavaScript enabled browser

---

1. The statement which prints the integer values 1 to 10 on the screen, is

```
12345678910
```

Statement 1

```
count = 1;
while( count <= 10 ) {
   printf("%d", count);
   count = count + 1;
}
```

Statement 2

```
count = 1;
while( count <= 10 ) {
   printf("%d", &count);
   count = count + 1;
}
```

Statement 3

```
count = 1;
while( count < 10 ) {
  printf("%d\n", count);
  count = count + 1;
}
```

Statement 4

```
count = 1;
while( count <= 10 ) {
  printf("%d\n", count);
  count = count + 1;
```

```
}
```

---

2. The statement which reproduces the following output, is

```
1
22
333
4444
55555
```

○ **Statement 1**

```
a = 1;
while( a <= 5 ) {
  while( b <= a ) {
    printf("%d\n", a);
    b = b + 1;
  }
  a = a + 1;
}
```

○ **Statement 2**

```
a = 1;
while( a <= 5 ) {
  b = 1;
  while( b <= a ) {
    printf("%d", a);
    b = b + 1;
  }
  printf("\n");
  a = a + 1;
}
```

○ **Statement 3**

```
a = 1;
while( a <= 5 ) {
  while( b <= 5 ) {
    printf("%d", a);
    b = b + 1;
  }
  a = a + 1;
  printf("\n");
}
```

○ **Statement 4**

```
a = 1;
while( a <= 5 ) {
  printf("\n");
  b = 1;
  while( a <= b ) {
    printf("%d", a);
    b = b + 1;
  }
  a = a + 1;
}
```

---

3. The statement that compares the value of an integer called *sum* against the value 65, and if it is less, prints the text string "Sorry, try again", is

○ **Statement 1**

```
if( sum < "65" )
   printf("Sorry, try again" );
```

○ **Statement 2**

```
if( sum <= 65 )
   printf("Sorry, try again" );
```

○ **Statement 3**

```
if( 65 == sum )
   printf("Sorry, try again" );
```

○ **Statement 4**

```
if( sum < 65 )
   printf("Sorry, try again" );
```

---

4. The statement that compares *total* for equality to *good_guess*, and if equal prints the value of *total*, and if not equal prints the value of *good_guess*, is

○ **Statement 1**

```
if( total < good_guess )
   printf("%d", total );
else
   printf("%d", good_guess );
```

○ **Statement 2**

```
if( total == good_guess )
   printf("%d", good_guess );
else
   printf("%d", total );
```

○ **Statement 3**

```
if( total = good_guess )
   printf("%d", total );
else
   printf("%d", good_guess );
```

○ **Statement 4**

```
if( total == good_guess )
   printf("%d", total );
else
   printf("%d", good_guess );
```

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

COMPOUND RELATIONALS (AND, NOT, OR, EOR)

Combining more than one condition
These allow the testing of more than one condition as part of selection statements. The symbols are

```
        LOGICAL AND     &&
```

Logical and requires all conditions to evaluate as TRUE (non-zero).

```
        LOGICAL OR      ||
```

Logical or will be executed if any ONE of the conditions is TRUE (non-zero).

```
        LOGICAL NOT      !
```

logical not negates (changes from TRUE to FALSE, vsvs) a condition.

```
        LOGICAL EOR      ^
```

Logical eor will be excuted if either condition is TRUE, but NOT if they are all true.

The following program uses an *if* statement with logical OR to validate the users input to be in the range 1-10.

```
        #include <stdio.h>

        main()
        {
                int number;
                int valid = 0;

                while( valid == 0 ) {
                        printf("Enter a number between 1 and 10 -->");
                        scanf("%d", &number);
                        if( (number < 1 ) || (number > 10) ){
                                printf("Number is outside range 1-10. Please re-enter\n");
                                valid = 0;
                        }
                        else
                                valid = 1;
```

```
        }
        printf("The number is %d\n", number );
}
```

```
Sample Program Output
Enter a number between 1 and 10 --> 56
Number is outside range 1-10. Please re-enter
Enter a number between 1 and 10 --> 6
The number is 6
```

This program is slightly different from the previous example in that a LOGICAL OR eliminates one of the *else* clauses.

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

INDEX Previous Next

COMPOUND RELATIONALS (AND, NOT, OR, EOR)

NEGATION

```
#include <stdio.h>

main()
{
        int flag = 0;
        if( ! flag ) {
                printf("The flag is not set.\n");
                flag = ! flag;
        }
        printf("The value of flag is %d\n", flag);
}
```

```
Sample Program Output
The flag is not set.
The value of flag is 1
```

The program tests to see if *flag* is not (!) set; equal to zero. It then prints the appropriate message, changes the state of *flag*; *flag* becomes equal to not *flag*; equal to 1. Finally the value of *flag* is printed.

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

INDEX Previous Next

COMPOUND RELATIONALS (AND, NOT, OR, EOR)

Range checking using Compound Relationals
Consider where a value is to be inputted from the user, and checked for validity to be within a certain range, lets say between the integer values 1 and 100.

```
#include <stdio.h>

main()
{
        int number;
        int valid = 0;

        while( valid == 0 ) {
                printf("Enter a number between 1 and 100");
                scanf("%d", &number );
                if( (number < 1) || (number > 100) )
                        printf("Number is outside legal range\n");
                else
                        valid = 1;
        }
        printf("Number is %d\n", number );
}
```

```
Sample Program Output
Enter a number between 1 and 100
203
Number is outside legal range
Enter a number between 1 and 100
-2
Number is outside legal range
Enter a number between 1 and 100
37
Number is 37
```

The program uses *valid*, as a flag to indicate whether the inputted data is within the required range of allowable values. The while loop continues whilst *valid* is 0.

The statement

```
if( (number < 1) || (number > 100) )
```

checks to see if the number entered by the user is within the valid range, and if so, will set the value of *valid* to 1, allowing the while loop to exit.

Now consider writing a program which validates a character to be within the range A-Z, in other words *alphabetic*.

```
#include <stdio.h>

main()
{
        char ch;
        int valid = 0;

        while( valid == 0 ) {
                printf("Enter a character A-Z");
                scanf(" %c", &ch );
                if( (ch >= 'A') && (ch <= 'Z') )
                        valid = 1;
                else
                        printf("Character is outside legal range\n");
        }
        printf("Character is %c\n", ch );
}
```

```
Sample Program Output
Enter a character A-Z
a
Character is outside legal range
Enter a character A-Z
0
Character is outside legal range
Enter a character A-Z
R
Character is R
```

In this instance, the AND is used because we want validity between a range, that is all values between a low and high limit. In the previous case, we used an OR statement to test to see if it was outside or below the lower limit or above the higher limit.

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

INDEX | Previous | Next

switch() case:
The *switch case* statement is a better way of writing a program when a series of *if elses* occurs. The general format for this is,

```
switch ( expression ) {
        case  value1:
                program statement;
                program statement;
                ......
                break;
        case  valuen:
                program statement;
                .......
                break;
        default:
                .......
                .......
                break;
}
```

The keyword *break* must be included at the end of each case statement. The default clause is optional, and is executed if the cases are not met. The right brace at the end signifies the end of the case selections.

Rules for switch statements

```
values for 'case' must be integer or character constants
the order of the 'case' statements is unimportant
the default clause may occur first (convention places it last)
you cannot use expressions or ranges
```

```
#include <stdio.h>

main()
{
        int menu, numb1, numb2, total;

        printf("enter in two numbers -->");
        scanf("%d %d", &numb1, &numb2 );
        printf("enter in choice\n");
        printf("1=addition\n");
```

```
            printf("2=subtraction\n");
            scanf("%d", &menu );
            switch( menu ) {
                    case 1: total = numb1 + numb2; break;
                    case 2: total = numb1 - numb2; break;
                    default: printf("Invalid option selected\n");
            }
            if( menu == 1 )
                    printf("%d plus %d is %d\n", numb1, numb2, total );
            else if( menu == 2 )
                    printf("%d minus %d is %d\n", numb1, numb2, total );
    }
```

```
Sample Program Output
enter in two numbers --> 37 23
enter in choice
1=addition
2=subtraction
2
37 minus 23 is 14
```

The above program uses a *switch* statement to validate and select upon the users input choice, simulating a simple menu of choices.

EXERCISE C11
Rewrite the previous program, which accepted two numbers and an operator, using the *switch case* statement.

Answer

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

Previous

THE switch case STATEMENT

EXERCISE C11
Rewrite the previous program, which accepted two numbers and an operator, using the *switch case* statement.

```
	/* Illustates nested if else and multiple arguments to the scanf function.
*/
	#include <stdio.h>

	main()
	{
		int  invalid_operator = 0;
		char  operator;
		float  number1, number2, result;

		printf("Enter two numbers and an operator in the format\n");
		printf(" number1 operator number2\n");
		scanf("%f %c %f", &number1, &operator, &number2);

		if(operator == '*')
			result = number1 * number2;
		else if(operator == '/')
			result = number1 / number2;
		else if(operator == '+')
			result = number1 + number2;
		else if(operator == '-')
			result = number1 - number2;
		else
			invalid_operator = 1;

		if( invalid_operator != 1 )
			printf("%f %c %f is %f\n", number1, operator, number2, result
);
		else
			printf("Invalid operator.\n");
	}
```

Solution

```
	/* Illustates switch */
	#include <stdio.h>

	main()
	{
		int  invalid_operator = 0;
		char  operator;
		float  number1, number2, result;

		printf("Enter two numbers and an operator in the format\n");
```

```
                printf(" number1 operator number2\n");
                scanf("%f %c %f", &number1, &operator, &number2);

                switch( operator ) {
                        case '*' : result = number1 * number2; break;
                        case '/' : result = number1 / number2; break;
                        case '+' : result = number1 + number2; break;
                        case '-' : result = number1 - number2; break;
                        default : invalid_operator = 1;
                }
                switch( invalid_operator ) {
                        case 1 : printf("Invalid operator.\n"); break;
                        default : printf("%f %c %f is %f\n", number1, operator,
number2, result );
                }
        }
```

©Copyright B Brown. 1984-2000. All rights reserved.

Previous

C Programming

INDEX Previous Next

Practise Exercise 6

Compound Relationals and switch

JavaScript compatible inter-active version of this test.

1. if sum is equal to 10 and total is less than 20, print the text string "incorrect.".

2. if flag is 1 or letter is not an 'X', then assign the value 0 to exit_flag, else set exit_flag to 1.

3. rewrite the following statements using a switch statement

```
if( letter == 'X' )
        sum = 0;
else if ( letter == 'Z' )
        valid_flag = 1;
else if( letter == 'A' )
        sum = 1;
else
        printf("Unknown letter -->%c\n", letter );
```

Answers

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

Previous

Answers: Practise Exercise 6

Compound Relationals and switch

1. if sum is equal to 10 and total is less than 20, print the text string "incorrect.".

```
if( (sum == 10) && (total < 20) )
        printf("incorrect.\n");
```

2. if flag is 1 or letter is not an 'X', then assign the value 0 to exit_flag, else set exit_flag to 1.

```
if( (flag == 1) || (letter != 'X') )
        exit_flag = 0;
else
        exit_flag = 1;
```

3. rewrite the following statements using a switch statement

```
if( letter == 'X' )
        sum = 0;
else if ( letter == 'Z' )
        valid_flag = 1;
else if( letter == 'A' )
        sum = 1;
else
        printf("Unknown letter -->%c\n", letter );


switch( letter ) {
        case 'X' : sum = 0; break;
        case 'Z' : valid_flag = 1; break;
        case 'A' : sum = 1; break;
        default  : printf("Unknown letter -->%c\n", letter );
}
```

©Copyright B Brown. 1984-2000. All rights reserved.

Previous

C Programming

Practise Exercise 6

Compound Relationals and switch

To run this test requires a JavaScript enabled browser

---

1. The statement that tests to see if *sum* is equal to 10 and *total* is less than 20, and if so, prints the text string "incorrect.", is

○ **Statement 1**

```
if( (sum = 10) && (total < 20) )
   printf("incorrect.");
```

○ **Statement 2**

```
if( (sum == 10) && (total < 20) )
   printf("incorrect.");
```

○ **Statement 3**

```
if( (sum == 10) || (total < 20) )
   printf("incorrect.");
```

---

2. if flag is 1 or letter is not an 'X', then assign the value 0 to exit_flag, else set exit_flag to 1.

○ **Statement 1**

```
if( (flag = 1) || (letter != 'X') )
   exit_flag = 0;
else
   exit_flag = 1;
```

○ **Statement 2**

```
if( (flag == 1) || (letter <> 'X') )
   exit_flag = 0;
else
   exit_flag = 1;
```

○ **Statement 3**

```
if( (flag == 1) || (letter != 'X') )
   exit_flag = 0;
else
   exit_flag = 1;
```

---

3. rewrite the following statements using a switch statement

```
if( letter == 'X' )
        sum = 0;
else if ( letter == 'Z' )
        valid_flag = 1;
else if( letter == 'A' )
        sum = 1;
else
        printf("Unknown letter -->%c\n", letter );
```

○ **Statement 1**

```
switch( letter ) {
   case 'X' : sum = 0; break;
   case 'Z' : valid_flag = 1; break;
   case 'A' : sum = 1; break;
   default  : printf( "Unknown letter -->%c\n", letter ); break;
}
```

○ **Statement 2**

```
switch( letter ) {
   case 'X' : sum = 0;
   case 'Z' : valid_flag = 1;
   case 'A' : sum = 1;
   default  : printf( "Unknown letter -->%c\n", letter );
}
```

○ **Statement 3**

```
switch( letter ) {
   case "X" : sum = 0; break;
   case "Z" : valid_flag = 1; break;
   case "A" : sum = 1; break;
   default  : printf( "Unknown letter -->%c\n", letter ); break;
}
```

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

INDEX Previous Next

ACCEPTING SINGLE CHARACTERS FROM THE KEYBOARD

getchar
The following program illustrates this,

```
#include <stdio.h>

main()
{
        int  i;
        int ch;

        for( i = 1; i<= 5; ++i ) {
                ch = getchar();
                putchar(ch);
        }
}
```

```
Sample Program Output
AACCddEEtt
```

The program reads five characters (one for each iteration of the for loop) from the keyboard. Note that *getchar()* gets a single character from the keyboard, and *putchar()* writes a single character (in this case, *ch*) to the console screen.

The file ctype.h provides routines for manipulating characters.

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

BUILT IN FUNCTIONS FOR STRING HANDLING

string.h
You may want to look at the section on arrays first!. The following macros are built into the file string.h

```
strcat          Appends a string
strchr          Finds first occurrence of a given character
strcmp          Compares two strings
strcmpi         Compares two strings, non-case sensitive
strcpy          Copies one string to another
strlen          Finds length of a string
strlwr          Converts a string to lowercase
strncat         Appends n characters of string
strncmp         Compares n characters of two strings
strncpy         Copies n characters of one string to another
strnset         Sets n characters of string to a given character
strrchr         Finds last occurrence of given character in string
strrev          Reverses string
strset          Sets all characters of string to a given character
strspn          Finds first substring from given character set in string
strupr          Converts string to uppercase
```

To convert a string to uppercase

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

main()
{
        char name[80];  /* declare an array of characters 0-79 */

        printf("Enter in a name in lowercase\n");
        scanf( "%s", name );
        strupr( name );
        printf("The name is uppercase is %s", name );
}
```

```
Sample Program Output
Enter in a name in lowercase
samuel
The name in uppercase is SAMUEL
```

BUILT IN FUNCTIONS FOR CHARACTER HANDLING

The following character handling functions are defined in ctype.h

```
isalnum         Tests for alphanumeric character
isalpha         Tests for alphabetic character
isascii         Tests for ASCII character
iscntrl         Tests for control character
isdigit         Tests for 0 to 9
isgraph         Tests for printable character
islower         Tests for lowercase
isprint         Tests for printable character
ispunct         Tests for punctuation character
isspace         Tests for space character
isupper         Tests for uppercase character
isxdigit        Tests for hexadecimal
toascii         Converts character to ascii code
tolower         Converts character to lowercase
toupper         Converts character to uppercase
```

To convert a string array to uppercase a character at a time using *toupper()*

```
#include <stdio.h>
#include <ctype.h>
main()
{
        char name[80];
        int loop;

        printf("Enter in a name in lowercase\n");
        scanf( "%s", name );
        for( loop = 0; name[loop] != 0; loop++ )
                name[loop] = toupper( name[loop] );

        printf("The name is uppercase is %s", name );
}
```

```
Sample Program Output
Enter in a name in lowercase
samuel
The name in uppercase is SAMUEL
```

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

Validation Of User Input In C

Basic Rules

- Don't pass invalid data onwards.
- Validate data at input time.
- Always give the user meaningful feedback
- Tell the user what you expect to read as input

```
/* example one, a simple continue statement */
#include <stdio.h>
#include <ctype.h>

main()
{
        int     valid_input;    /* when 1, data is valid and loop is exited */
        char    user_input;     /* handles user input, single character menu choice
*/

        valid_input = 0;
        while( valid_input == 0 ) {
                printf("Continue (Y/N)?\n");
                scanf("  %c", &user_input );
                user_input = toupper( user_input );
                if((user_input == 'Y') || (user_input == 'N') )  valid_input = 1;
                else  printf("\007Error: Invalid choice\n");
        }
}
```

Sample Program Output

```
Continue (Y/N)?
b
Error: Invalid Choice
Continue (Y/N)?
N
```

```
/* example two, getting and validating choices */
#include <stdio.h>
#include <ctype.h>

main()
{
        int     exit_flag = 0, valid_choice;
        char    menu_choice;
```

```
        while( exit_flag == 0 ) {
                valid_choice = 0;
                while( valid_choice == 0 ) {
                        printf("\nC = Copy File\nE = Exit\nM = Move File\n");
                        printf("Enter choice:\n");
                        scanf("   %c", &menu_choice );
                        if((menu_choice=='C') || (menu_choice=='E') ||
(menu_choice=='M'))
                                valid_choice = 1;
                        else
                                printf("\007Error. Invalid menu choice selected.\n");
                }
                switch( menu_choice ) {
                        case 'C' : ....................();    break;
                        case 'E' : exit_flag = 1;  break;
                        case 'M' : ....................();  break;
                        default : printf("Error--- Should not occur.\n"); break;
                }
        }
}
```

Sample Program Output

```
C = Copy File
E = Exit
M = Move File
Enter choice:
X
Error. Invalid menu choice selected.
C = Copy File
E = Exit
M = Move File
Enter choice:
E
```

Other validation examples

© Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

INDEX Previous Next

THE CONDITIONAL EXPRESSION OPERATOR

This conditional expression operator takes THREE operators. The two symbols used to denote this operator are the ? and the :. The first operand is placed before the ?, the second operand between the ? and the :, and the third after the :. The general format is,

```
        condition ? expression1 : expression2
```

If the result of condition is TRUE (non-zero), expression1 is evaluated and the result of the evaluation becomes the result of the operation. If the condition is FALSE (zero), then expression2 is evaluated and its result becomes the result of the operation. An example will help,

```
        s = ( x < 0 ) ? -1 : x * x;

        If x is less than zero then s = -1
        If x is greater than zero then s = x * x
```

Example program illustrating conditional expression operator

```
        #include <stdio.h>

        main()
        {
                int input;

                printf("I will tell you if the number is positive, negative or
zero!\n");
                printf("please enter your number now--->");
                scanf("%d", &input );
                (input < 0) ? printf("negative\n") : ((input > 0) ?
printf("positive\n") : printf("zero\n"));
        }


        Sample Program Output
        I will tell you if the number is positive, negative or zero!
        please enter your number now---> 32
        positive
```

CLASS EXERCISE C12

Evaluate the following expression, where a=4, b=5

```
least_value = ( a < b ) ? a : b;
```

Answers

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

Previous

Answers: CLASS EXERCISE C12
Evaluate the following expression, where a=4, b=5

```
least_value = ( a < b ) ? a : b;
```

least_value = 4

©Copyright B Brown. 1984-2000. All rights reserved.

Previous

C Programming

INDEX Previous Next

ARRAYS

Little Boxes on the hillside
Arrays are a data structure which hold multiple variables of the same data type. Consider the case where a programmer needs to keep track of a number of people within an organisation. So far, our initial attempt will be to create a specific variable for each user. This might look like,

```
int name1 = 101;
int name2 = 232;
int name3 = 231;
```

It becomes increasingly more difficult to keep track of this as the number of variables increase. Arrays offer a solution to this problem.

An array is a multi-element box, a bit like a filing cabinet, and uses an indexing system to find each variable stored within it. In C, indexing starts at **zero**.

Arrays, like other variables in C, must be declared before they can be used.

The replacement of the above example using arrays looks like,

```
int names[4];
names[0] = 101;
names[1] = 232;
names[2] = 231;
names[3] = 0;
```

We created an array called *names*, which has space for four integer variables. You may also see that we stored 0 in the last space of the array. This is a common technique used by C programmers to signify the end of an array.

Arrays have the following syntax, using square brackets to access each indexed value (called an **element**).

```
x[i]
```

so that *x[5]* refers to the sixth element in an array called *x*. In C, array elements start with 0. Assigning values to array elements is done by,

```
x[10] = g;
```

and assigning array elements to a variable is done by,

```
g = x[10];
```

In the following example, a character based array named *word* is declared, and each element is assigned a character. The last element is filled with a zero value, to signify the end of the character string (in C, there is no string type, so character based arrays are used to hold strings). A printf statement is then used to print out all elements of the array.

```
/*  Introducing array's, 2  */
#include <stdio.h>

main()
{
        char word[20];

        word[0] = 'H';
        word[1] = 'e';
        word[2] = 'l';
        word[3] = 'l';
        word[4] = 'o';
        word[5] = 0;
        printf("The contents of word[] is -->%s\n", word );
}
```

```
Sample Program Output
The contents of word[] is Hello
```

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

DECLARING ARRAYS

Arrays may consist of any of the valid data types. Arrays are declared along with all other variables in the declaration section of the program.

```
/*  Introducing array's  */
#include <stdio.h>

main()
{
        int   numbers[100];
        float  averages[20];

        numbers[2] = 10;
        --numbers[2];
        printf("The 3rd element of array numbers is %d\n", numbers[2]);
}
```

```
Sample Program Output
The 3rd element of array numbers is 9
```

The above program declares two arrays, assigns 10 to the value of the 3rd element of array *numbers*, decrements this value (*--numbers[2]*), and finally prints the value. The number of elements that each array is to have is included inside the square brackets.

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

ASSIGNING INITIAL VALUES TO ARRAYS

The declaration is preceded by the word *static*. The initial values are enclosed in braces, eg,

```
#include <stdio.h>
main()
{
        int x;
        static int  values[] = { 1,2,3,4,5,6,7,8,9 };
        static char  word[] = { 'H','e','l','l','o' };
        for( x = 0; x < 9; ++x )
                printf("Values [%d] is %d\n", x, values[x]);
}
```

```
Sample Program Output
Values[0] is 1
Values[1] is 2
....
Values[8] is 9
```

The previous program declares two arrays, *values* and *word*. Note that inside the squarebrackets there is no variable to indicate how big the array is to be. In this case, C initializes the array to the number of elements that appear within the initialize braces. So values consist of 9 elements (numbered 0 to 8) and the char array *word* has 5 elements.

The following program shows how to initialise all the elements of an integer based array to the value 10, using a for loop to cycle through each element in turn.

```
#include <stdio.h>
main()
{
        int count;
        int  values[100];
        for( count = 0; count < 100; count++ )
                values[count] = 10;
}
```

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

INDEX Previous Next

MULTI DIMENSIONED ARRAYS
Multi-dimensioned arrays have two or more index values which specify the element in the array.

```
multi[i][j]
```

In the above example, the first index value *i* specifies a row index, whilst *j* specifies a column index.

Declaration and calculations

```
int         m1[10][10];
static int m2[2][2] = { {0,1}, {2,3} };

sum = m1[i][j] + m2[k][l];
```

NOTE the strange way that the initial values have been assigned to the two-dimensional array m2. Inside the braces are,

```
{ 0, 1 },
{ 2, 3 }
```

Remember that arrays are split up into row and columns. The first is the row, the second is the column. Looking at the initial values assigned to *m2*, they are,

```
m2[0][0] = 0
m2[0][1] = 1
m2[1][0] = 2
m2[1][1] = 3
```

EXERCISE C13
Given a two dimensional array, write a program that totals all elements, printing the total.

CLASS EXERCISE C14
What value is assigned to the elements which are not assigned initialised.

[Answers](#)

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

Previous

EXERCISE C13

Given a two dimensional array write a program that totals all elements printing the total.

```
#include <stdio.h>

main()
{
        static int m[][] = { {10,5,-3}, {9, 0, 0}, {32,20,1}, {0,0,8} };
        int row, column, sum;

        sum = 0;
        for( row = 0; row < 4; row++ )
                for( column = 0; column < 3; column++ )
                        sum = sum + m[row][column];
        printf("The total is %d\n", sum );
}
```

CLASS EXERCISE C14

They get initialised to **ZERO**.

©Copyright B Brown. 1984-2000. All rights reserved.

Previous

C Programming

CHARACTER ARRAYS [STRINGS]

Consider the following program,

```
#include <stdio.h>
main()
{
        static char name1[] = {'H','e','l','l','o'};
        static char name2[] = "Hello";
        printf("%s\n", name1);
        printf("%s\n", name2);
}
```

Sample Program Output

```
Helloxghifghjkloqw30-=kl`'
Hello
```

The difference between the two arrays is that *name2* has a null placed at the end of the string, ie, in name2[5], whilst *name1* has not. This can often result in garbage characters being printed on the end. To insert a null at the end of the name1 array, the initialization can be changed to,

```
static char name1[] = {'H','e','l','l','o','\0'};
```

Consider the following program, which initialises the contents of the character based array *word* during the program, using the function *strcpy*, which necessitates using the include file *string.h*

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

main()
{
        char word[20];

        strcpy( word, "hi there." );
        printf("%s\n", word );
}
```

Sample Program Output

```
hi there.
```

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

SOME VARIATIONS IN DECLARING ARRAYS

```
int  numbers[10];

static int numbers[10] = { 34, 27, 16 };

static int numbers[] = { 2, -3, 45, 79, -14, 5, 9, 28, -1, 0 };

static char text[] = "Welcome to New Zealand.";

static float radix[12] = { 134.362, 1913.248 };

double  radians[1000];
```

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

INDEX Previous Next

READING CHARACTER STRINGS FROM THE KEYBOARD

Character based arrays are often refered to in C as strings. C does not support a string type, so character based arrays are used in place of strings. The *%s* modifier to *printf()* and *scanf()* is used to handle character based arrays. This assumes that a 0 or NULL value is stored in the last element of the array. Consider the following, which reads a string of characters (excluding spaces) from the keyboard.

```
char string[18];
scanf("%s", string);
```

NOTE that the & character does not need to precede the variable name when the formatter %s is used! If the users response was

```
Hello<enterkey>
```

then

```
string[0] = 'H'
string[1] = 'e'
....
string[4] = 'o'
string[5] = '\0'
```

Note how the enterkey is not taken by *scanf()* and the text string is terminated by a NULL character '\0' after the last character stored in the array.

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

INDEX | Previous | Next

Practise Exercise 7: Arrays

JavaScript compatible inter-active version of this test.

1. Declare a character based array called letters of ten elements

2. Assign the character value 'Z' to the fourth element of the letters array

3. Use a for loop to total the contents of an integer array called numbers which has five elements. Store the result in an integer called total.

4. Declare a multidimensioned array of floats called balances having three rows and five columns.

5. Write a for loop to total the contents of the multidimensioned float array balances.

6. Assign the text string "Hello" to the character based array words at declaration time.

7. Assign the text string "Welcome" to the character based array stuff (not at declaration time)

8. Use a printf statement to print out the third element of an integer array called totals

9. Use a printf statement to print out the contents of the character array called words

10. Use a scanf statement to read a string of characters into the array words.

11. Write a for loop which will read five characters (use scanf) and deposit them into the character based array words, beginning at element 0.

Answers

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX | Previous | Next

C Programming

Previous

Answers: Practise Exercise 7: Arrays

1. Declare a character based array called letters of ten elements

```
char letters[10];
```

2. Assign the character value 'Z' to the fourth element of the letters array

```
letters[3] = 'Z';
```

3. Use a for loop to total the contents of an integer array called numbers which has five elements. Store the result in an integer called total.

```
for( loop = 0, total = 0; loop < 5; loop++ )
        total = total + numbers[loop];
```

4. Declare a multidimensioned array of floats called balances having three rows and five columns.

```
float balances[3][5];
```

5. Write a for loop to total the contents of the multidimensioned float array balances.

```
for( row = 0, total = 0; row < 3; row++ )
        for( column = 0; column < 5; column++ )
                total = total + balances[row][column];
```

6. Assign the text string "Hello" to the character based array words at declaration time.

```
static char words[] = "Hello";
```

7. Assign the text string "Welcome" to the character based array stuff (not at declaration time)

```
char stuff[50];

strcpy( stuff, "Welcome" );
```

8. Use a printf statement to print out the third element of an integer array called totals

```
printf("%d\n", totals[2] );
```

9. Use a printf statement to print out the contents of the character array called words

```
printf("%s\n", words);
```

10. Use a scanf statement to read a string of characters into the array words.

```
scanf("%s", words);
```

11. Write a for loop which will read five characters (use scanf) and deposit them into the character based array words, beginning at element 0.

```
        for( loop = 0; loop < 5; loop++ )
                scanf("%c", &words[loop] );
```

©Copyright B Brown. 1984-2000. All rights reserved.

Previous

C Programming

INDEX Previous Next

Practise Exercise 7: Arrays

To run this test requires a JavaScript enabled browser

---

1. The statement which declares a character based array called *letters* of ten elements is,

- letters : char[10];
- char[10] letters;
- char letters[10];
- char array letters[0..9];

---

2. Assign the character value 'Z' to the fourth element of the *letters* array

- letters[4] := "Z";
- letters[3] = 'Z';
- letters[3] = 'z';
- letters[4] = "Z";

---

3. Use a for loop to total the contents of an integer array called *numbers* which has five elements. Store the result in an integer called *total*.

Statement 1

```
for( loop = 0, total = 0; loop >= 4; loop++ )
        total = total + numbers[loop];
```

Statement 2

```
for( loop = 0, total = 0; loop < 5; loop++ )
        total = total + numbers[loop];
```

Statement 3

```
for( loop = 0, total = 0; loop <= 5; loop++ )
        total = total + numbers[loop];
```

---

4. Declare a multidimensioned array of floats called *balances* having three rows and five columns.

- float balances[3][5];

- balances[3][5] of float;
- float balances[5][3];
- array of float balances[0..2][0..5];

---

5. Write a for loop to total the contents of the multidimensioned float array *balances*, as declared in question 4.

- **Statement 1**

```
for( row = 0, total = 0; row < 3; row++ )
        for( column = 0, total = 0; column < 5; column++ )
                total = total + balances[row][column];
```

- **Statement 2**

```
for( row = 0, total = 0; row < 3; row++ )
        for( column = 0; column < 5; column++ )
                total = total + balances[row][column];
```

- **Statement 3**

```
for( row = 0, total = 0; row < 3; row++ )
        for( column = 0; column < row; column++ )
                total = total + balances[row][column];
```

---

6. Assign the text string "Hello" to the character based array *words* at declaration time.

- char words[10] = 'Hello';
- static char words[] = "Hello";
- static char words["hello"];
- static char words[] = { Hello };

---

7. Assign the text string "Welcome" to the character based array *stuff* (not at declaration time)

- strcpy(stuff, 'Welcome');
- stuff = "Welcome";
- stuff[0] = "Welcome";
- strcpy(stuff, "Welcome");

---

8. Use a printf statement to print out the third element of an integer array called *totals*

- printf("%d\n", &totals[3]);
- printf("%d\n", totals[3]);
- printf("%c\n", totals[2]);
- printf("%d\n", totals[2]);

---

9. Use a printf statement to print out the contents of the character array called *words*

- printf("%s\n", words);
- printf("%c\n", words);
- printf("%d\n", words);
- printf("%s\n", words[2]);

---

10. Use a scanf statement to read a string of characters into the array *words*.

- scanf("%s\n", words);
- scanf(" %c", words);
- scanf("%c", words);
- scanf("%s", words);

---

11. Write a for loop which will read five characters (use scanf) and deposit them into the character based array words, beginning at element 0.

Statement 1

```
        for( loop = 0; loop < 5; loop++ )
                scanf("%c", &words[loop] );
```

Statement 2

```
        for( loop = 0; loop <= 5; loop++ )
                scanf("%c", words );
```

Statement 3

```
        for( loop = 0; loop < 5; loop++ )
                scanf("%c", &words[0] );
```

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

INDEX Previous Next

FUNCTIONS
A function in C can perform a particular task, and supports the concept of modular programming design techniques.

We have already been exposed to functions. The main body of a C program, identified by the keyword *main*, and enclosed by the left and right braces is a function. It is called by the operating system when the program is loaded, and when terminated, returns to the operating system.

Functions have a basic structure. Their format is

```
        return_data_type  function_name  ( arguments, arguments )
        data_type_declarations_of_arguments;
        {
                function_body
        }
```

It is worth noting that a return_data_type is assumed to be type *int* unless otherwise specified, thus the programs we have seen so far imply that *main()* returns an integer to the operating system.

ANSI C varies slightly in the way that functions are declared. Its format is

```
        return_data_type function_name (data_type variable_name, data_type
variable_name, .. )
        {
                function_body
        }
```

This permits type checking by utilizing function prototypes to inform the compiler of the type and number of parameters a function accepts. When calling a function, this information is used to perform type and parameter checking.

ANSI C also requires that the return_data_type for a function which does not return data must be type *void*. The default return_data_type is assumed to be integer unless otherwise specified, but must match that which the function declaration specifies.

A simple function is,

```
        void print_message( void )
        {
                printf("This is a module called print_message.\n");
        }
```

Note the function name is *print_message*. No arguments are accepted by the function, this is indicated by the keyword *void* in the accepted parameter section of the function declaration. The return_data_type is void, thus data is not returned by the function.

An ANSI C function prototype for *print_message()* is,

```
        void print_message( void );
```

Function prototypes are listed at the beginning of the source file. Often, they might be placed in a users .h (header) file.

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

FUNCTIONS

Now lets incorporate this function into a program.

```
/* Program illustrating a simple function call */
#include <stdio.h>

void print_message( void );    /* ANSI C function prototype */

void print_message( void )     /* the function code */
{
        printf("This is a module called print_message.\n");
}

main()
{
        print_message();
}
```

Sample Program Output

```
This is a module called print_message.
```

To call a function, it is only necessary to write its name. The code associated with the function name is executed at that point in the program. When the function terminates, execution begins with the statement which follows the function name.

In the above program, execution begins at *main()*. The only statement inside the main body of the program is a call to the code of function *print_message()*. This code is executed, and when finished returns back to *main()*.

As there is no further statements inside the main body, the program terminates by returning to the operating system.

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

INDEX | Previous | Next

FUNCTIONS
In the following example, the function accepts a single data variable, but does not return any information.

```
/* Program to calculate a specific factorial number  */
#include <stdio.h>

void calc_factorial( int );    /* ANSI function prototype */

void calc_factorial( int n )
{
        int  i, factorial_number = 1;

        for( i = 1; i <= n; ++i )
                factorial_number *= i;

        printf("The factorial of %d is %d\n", n, factorial_number );
}

main()
{
        int  number = 0;

        printf("Enter a number\n");
        scanf("%d", &number );
        calc_factorial( number );
}
```

```
Sample Program Output
Enter a number
3
The factorial of 3 is 6
```

Lets look at the function *calc_factorial()*. The declaration of the function

```
void calc_factorial( int n )
```

indicates there is no return data type and a single integer is accepted, known inside the body of the function as *n*. Next comes the declaration of the local variables,

```
int  i, factorial_number = 0;
```

It is more correct in C to use,

```
        auto int  i, factorial_number = 0;
```

as the keyword *auto* designates to the compiler that the variables are local. The program works by accepting a variable from the keyboard which is then passed to the function. In other words, the variable *number* inside the main body is then copied to the variable *n* in the function, which then calculates the correct answer.

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

INDEX | Previous | Next

RETURNING FUNCTION RESULTS
This is done by the use of the keyword *return*, followed by a data variable or constant value, the data type of which must match that of the declared return_data_type for the function.

```
	float add_numbers( float n1, float n2 )
	{
		return n1 + n2;    /* legal */
		return 6;          /* illegal, not the same data type */
		return 6.0;        /* legal */
	}
```

It is possible for a function to have multiple return statements.

```
	int validate_input( char command )
	{
		switch( command ) {
			case '+' :
			case '-' : return 1;
			case '*' :
			case '/' : return 2;
			default  : return 0;
		}
	}
```

Here is another example

```
	/* Simple multiply program using argument passing */
	#include <stdio.h>

	int calc_result( int, int );          /* ANSI function prototype */

	int calc_result( int numb1, int numb2 )
	{
		auto int result;
		result = numb1 * numb2;
		return result;
	}

	main()
	{
		int  digit1 = 10, digit2 = 30, answer = 0;
		answer = calc_result( digit1, digit2 );
		printf("%d multiplied by %d is %d\n", digit1, digit2, answer );
	}
```

```
Sample Program Output
10 multiplied by 30 is 300
```

NOTE that the value which is returned from the function (ie result) must be declared in the function.

NOTE: The formal declaration of the function name is preceded by the data type which is returned,

```
int  calc_result ( numb1, numb2 )
```

EXERCISE C15
Write a program in C which incorporates a function using parameter passing and performs the addition of three numbers. The main section of the program is to print the result.

Answer

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

Previous

Answer: EXERCISE C15

Write a program in C which incorporates a function using parameter passing and performs the addition of three numbers. The main section of the program is to print the result.

```
#include <stdio.h>
int calc_result( int, int, int );

int calc_result( int var1, int var2, int var3 )
{
  int sum;

  sum = var1 + var2 + var3;
  return( sum );           /* return( var1 + var2 + var3 ); */
}

main()
{
  int numb1 = 2, numb2 = 3, numb3=4, answer=0;

  answer = calc_result( numb1, numb2, numb3 );
  printf("%d + %d + %d = %d\n", numb1, numb2, numb3, answer);
}
```

©Copyright B Brown. 1984-2000. All rights reserved.

Previous

C Programming

INDEX Previous Next

LOCAL AND GLOBAL VARIABLES

Local
These variables only exist inside the specific function that creates them. They are unknown to other functions and to the main program. As such, they are normally implemented using a stack. Local variables cease to exist once the function that created them is completed. They are recreated each time a function is executed or called.

Global
These variables can be accessed (ie known) by any function comprising the program. They are implemented by associating memory locations with variable names. They do not get recreated if the function is recalled.

DEFINING GLOBAL VARIABLES

```
        /* Demonstrating Global variables  */
        #include <stdio.h>
        int add_numbers( void );                /* ANSI function prototype */

        /* These are global variables and can be accessed by functions from this
point on */
        int  value1, value2, value3;

        int add_numbers( void )
        {
                auto int result;
                result = value1 + value2 + value3;
                return result;
        }

        main()
        {
                auto int result;
                value1 = 10;
                value2 = 20;
                value3 = 30;
                result = add_numbers();
                printf("The sum of %d + %d + %d is %d\n",
                        value1, value2, value3, final_result);
        }


        Sample Program Output
        The sum of 10 + 20 + 30 is 60
```

The scope of global variables can be restricted by carefully placing the declaration. They are visible from the declaration until the end of the current source file.

```
        #include <stdio.h>
        void no_access( void ); /* ANSI function prototype */
        void all_access( void );

        static int n2;          /* n2 is known from this point onwards */

        void no_access( void )
        {
                n1 = 10;        /* illegal, n1 not yet known */
                n2 = 5;         /* valid */
        }

        static int n1;          /* n1 is known from this point onwards */

        void all_access( void )
        {
                n1 = 10;        /* valid */
                n2 = 3;         /* valid */
        }
```

AUTOMATIC AND STATIC VARIABLES
C programs have a number of segments (or areas) where data is located. These segments are typically,

```
        _DATA   Static data
        _BSS    Uninitialized static data, zeroed out before call to main()
        _STACK  Automatic data, resides on stack frame, thus local to functions
        _CONST  Constant data, using the ANSI C keyword const
```

The use of the appropriate keyword allows correct placement of the variable onto the desired data segment.

```
        /* example program illustrates difference between static and automatic
variables */
        #include <stdio.h>
        void demo( void );              /* ANSI function prototypes */

        void demo( void )
        {
                auto int avar = 0;
                static int svar = 0;

                printf("auto = %d, static = %d\n", avar, svar);
                ++avar;
                ++svar;
        }


        main()
        {
```

```
        int i;

        while( i < 3 ) {
                demo();
                i++;
        }
}
```

```
Sample Program Output
auto = 0, static = 0
auto = 0, static = 1
auto = 0, static = 2
```

Sample program output

Static variables are created and initialized once, on the first call to the function. Subsequent calls to the function do not recreate or re-initialize the static variable. When the function terminates, the variable still exists on the _DATA segment, but cannot be accessed by outside functions.

Automatic variables are the opposite. They are created and re-initialized on each entry to the function. They disappear (are de-allocated) when the function terminates. They are created on the _STACK segment.

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

Previous

AUTOMATIC AND STATIC VARIABLES

```
        /* example program illustrates difference between static and automatic
variables */
        #include <stdio.h>
        void demo( void );               /* ANSI function prototypes */

        void demo( void )
        {
                auto int avar = 0;
                static int svar = 0;

                printf("auto = %d, static = %d\n", avar, svar);
                ++avar;
                ++svar;
        }


        main()
        {
                int i;

                while( i < 3 ) {
                        demo();
                        i++;
                }
        }
```

Program output

```
        auto = 0, static = 0
        auto = 0, static = 1
        auto = 0, static = 2
```

©Copyright B Brown. 1984-2000. All rights reserved.

Previous

C Programming

INDEX | Previous | Next

PASSING ARRAYS TO FUNCTIONS

The following program demonstrates how to pass an array to a function.

```
/* example program to demonstrate the passing of an array */
#include <stdio.h>
int maximum( int [] );          /* ANSI function prototype */

int  maximum( int values[5] )
{
        int  max_value, i;

        max_value = values[0];
        for( i = 0; i < 5; ++i )
                if( values[i] > max_value )
                        max_value = values[i];

        return max_value;
}

main()
{
        int values[5], i, max;

        printf("Enter 5 numbers\n");
        for( i = 0; i < 5; ++i )
                scanf("%d", &values[i] );

        max = maximum( values );
        printf("\nMaximum value is %d\n", max );
}
```

Sample Program Output

```
Enter 5 numbers
7 23 45 9 121
Maximum value is 121
```

Note: The program defines an array of five elements (values) and initializes each element to the users inputted values. The array *values* is then passed to the function. The declaration

```
        int  maximum( int values[5] )
```

defines the function name as *maximum*, and declares that an integer is passed back as the result, and that it accepts a data type called *values*, which is declared as an array of five integers. The values array in the main body is now known as the array values inside function maximum. **IT IS NOT A COPY, BUT THE ORIGINAL**.

This means any changes will update the original array.

A local variable *max_value* is set to the first element of values, and a for loop is executed which cycles through each element in values and assigns the lowest item to *max_value*. This number is then passed back by the return statement, and assigned to *max* in the main section.

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

Functions and Arrays

C allows the user to build up a library of modules such as the maximum value found in the previous example.

However, in its present form this module or function is limited as it only accepts ten elements. It is thus desirable to modify the function so that it also accepts the number of elements as an argument also. A modified version follows,

```
/* example program to demonstrate the passing of an array */
#include <stdio.h>

int findmaximum( int [], int );             /* ANSI function prototype */

int  findmaximum( int numbers[], int elements )
{
        int  largest_value, i;

        largest_value = numbers[0];

        for( i = 0; i < elements; ++i )
                if( numbers[i] > largest_value )
                        largest_value = numbers[i];

        return largest_value;
}

main()
{
        static int numb1[] = { 5, 34, 56, -12, 3, 19 };
        static int numb2[] = { 1, -2, 34, 207, 93, -12 };

        printf("maximum of numb1[] is %d\n", findmaximum(numb1, 6));
        printf("maximum is numb2[] is %d\n", findmaximum(numb2, 6));
}
```

Sample Program Output

```
maximum of numb1[] is 56
maximum of numb2[] is 207
```

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

PASSING OF ARRAYS TO FUNCTIONS
If an entire array is passed to a function, any changes made also occur to the original array.

PASSING OF MULTIDIMENSIONAL ARRAYS TO FUNCTIONS
If passing a multidimensional array, the number of columns must be specified in the formal parameter declaration section of the function.

EXERCISE C16
Write a C program incorporating a function to add all elements of a two dimensional array. The number of rows are to be passed to the function, and it passes back the total sum of all elements (Use at least a 4 x 4 array).

Answer

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

Previous

EXERCISE C16

Write a C program incorporating a function to add all elements of a two dimensional array. The number of rows are to be passed to the function, and it passes back the total sum of all elements (Use at least a 4 x 4 array).

```
#include <stdio.h>

int add2darray( int [][5], int );       /* function prototype */

int add2darray( int array[][5], int rows )
{
        int total = 0, columns, row;

        for( row = 0; row < rows; row++ )
                for( columns = 0; columns < 5; columns++ )
                        total = total + array[row][columns];
        return total;
}

main()
{
        int numbers[][] = { {1, 2, 35, 7, 10}, {6, 7, 4, 1, 0} };
        int sum;

        sum = add2darray( numbers, 2 );
        printf("the sum of numbers is %d\n", sum );
}
```

©Copyright B Brown. 1984-2000. All rights reserved.

Previous

C Programming

FUNCTION PROTOTYPES
These have been introduced into the C language as a means of provided type checking and parameter checking for function calls. Because C programs are generally split up over a number of different source files which are independently compiled, then linked together to generate a run-time program, it is possible for errors to occur.

Consider the following example.

```
/* source file add.c */
void add_up( int numbers[20] )
{
        ....
}

/* source file mainline.c */
static float values[] = { 10.2, 32.1, 0.006, 31.08 };

main()
{
        float result;
        ...
        result = add_up( values );
}
```

As the two source files are compiled separately, the compiler generates correct code based upon what the programmer has written. When compiling mainline.c, the compiler assumes that the function add_up accepts an array of float variables and returns a float. When the two portions are combined and ran as a unit, the program will definitely not work as intended.

To provide a means of combating these conflicts, ANSI C has function prototyping. Just as data types need to be declared, functions are declared also. The function prototype for the above is,

```
/* source file mainline.c */
void add_up( int numbers[20] );
```

NOTE that the function prototype ends with a semi-colon; in this way we can tell its a declaration of a function type, not the function code. If mainline.c was re-compiled, errors would be generated by the call in the main section which references *add_up()*.

Generally, when developing a large program, a separate file would be used to contain all the function prototypes. This file can then be included by the compiler to enforce type and parameter checking.

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

ADDITIONAL ASSIGNMENT OPERATOR
Consider the following statement,

```
numbers[loop] += 7;
```

This assignment += is equivalent to add equals. It takes the value of *numbers[loop]*, adds it by 7, then assigns the value to *numbers[loop]*. In other words it is the same as,

```
numbers[loop] = numbers[loop] + 7;
```

CLASS EXERCISE C17
What is the outcome of the following, assuming time=2, a=3, b=4, c=5

```
time -= 5;
a *= b + c;
```

Answers

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

CLASS EXERCISE C17

What is the outcome of the following, assuming time=2, a=3, b=4, c=5

```
time -= 5;
a *= b + c;
```

```
time = -3
a = 27
```

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

INDEX Previous Next

A SIMPLE EXCHANGE SORT ALGORITHM

The following steps define an algorithm for sorting an array,

```
1. Set i to 0
2. Set j to i + 1
3. If a[i] > a[j], exchange their values
4. Set j to j + 1. If j < n goto step 3
5. Set i to i + 1. If i < n - 1 goto step 2
6. a is now sorted in ascending order.

Note: n is the number of elements in the array.
```

EXERCISE C18

Implement the above algorithm as a function in C, accepting the array and its size, returning the sorted array in ascending order so it can be printed out by the calling module. The array should consist of ten elements.

Answer

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

Previous

A SIMPLE EXCHANGE SORT ALGORITHM
The following steps define an algorithm for sorting an array,

```
1. Set i to 0
2. Set j to i + 1
3. If a[i] > a[j], exchange their values
4. Set j to j + 1. If j < n goto step 3
5. Set i to i + 1. If i < n - 1 goto step 2
6. a is now sorted in ascending order.

Note: n is the number of elements in the array.
```

EXERCISE C18
Implement the above algorithm as a function in C, accepting the array and its size, returning the sorted array in ascending order so it can be printed out by the calling module. The array should consist of ten elements.

```
#include <stdio.h>

void sort( int [], int );

void sort( int a[], int elements )
{
        int i, j, temp;

        i = 0;
        while( i < (elements - 1) ) {
                j = i + 1;
                while( j < elements ) {
                        if( a[i] > a[j] ) {
                                temp = a[i];
                                a[i] = a[j];
                                a[j] = temp;
                        }
                        j++;
                }
                i++;
        }
}

main()
```

```
{
        int numbers[] = { 10, 9, 8, 23, 19, 11, 2, 7, 1, 13, 12 };
        int loop;

        printf("Before the sort the array was \n");
        for( loop = 0; loop < 11; loop++ )
                printf(" %d ", numbers[loop] );
        sort( numbers, 11 );
        printf("After the sort the array was \n");
        for( loop = 0; loop < 11; loop++ )
                printf(" %d ", numbers[loop] );
}
```

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

INDEX | Previous | Next

RECURSION

This is where a function repeatedly calls itself to perform calculations. Typical applications are games and Sorting trees and lists.

Consider the calculation of 6! (6 factorial)

```
 ie 6! = 6 * 5 * 4 * 3 * 2 * 1
    6! = 6 * 5!
    6! = 6 * ( 6 - 1 )!
    n! = n * ( n - 1 )!
```

```
        /* bad example for demonstrating recursion */
        #include <stdio.h>

        long int factorial( long int );         /* ANSI function prototype */

        long int  factorial( long int n )
        {
                long int result;

                if( n == 0L )
                        result = 1L;
                else
                        result = n * factorial( n - 1L );
                return ( result );
        }

        main()
        {
                int j;

                for( j = 0; j < 11; ++j )
                        printf("%2d! = %ld\n", factorial( (long) j) );
        }
```

EXERCISE C19

Rewrite example c9 using a recursive function.

Answer

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

Previous

RECURSIVE PROGRAMMING: EXERCISE C19
Rewrite example c9 using a recursive function.

```
#include <stdio.h>
long int triang_rec( long int );

long int triang_rec( long int number )
{
    long int result;

    if( number == 0l )
      result = 0l;
    else
      result = number + triang_rec( number - 1 );
    return( result );
}

main ()
{
  int request;
  long int triang_rec(), answer;

  printf("Enter number to be calculated.\n");
  scanf( "%d", &request);

  answer = triang_rec( (long int) request );
  printf("The triangular answer is %l\n", answer);
}
```

Note this version of function triang_rec

```
#include <stdio.h>
long int triang_rec( long int );

long int triang_rec( long int number )
{
   return((number == 0l) ? 0l : number*triang_rec( number-1));
}
```

© Copyright B Brown. 1984-2000. All rights reserved.

C Programming

Practise Exercise 8: Functions

JavaScript compatible inter-active version of this test.

1. Write a function called menu which prints the text string "Menu choices". The function does not pass any data back, and does not accept any data as parameters.

2. Write a function prototype for the above function.

3. Write a function called print which prints a text string passed to it as a parameter (ie, a character based array).

4. Write a function prototype for the above function print.

5. Write a function called total, which totals the sum of an integer array passed to it (as the first parameter) and returns the total of all the elements as an integer. Let the second parameter to the function be an integer which contains the number of elements of the array.

6. Write a function prototype for the above function.

Answers

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

Practise Exercise 8: Functions

1. Write a function called menu which prints the text string "Menu choices". The function does not pass any data back, and does not accept any data as parameters.

```
void menu( void )
{
        printf("Menu choices");
}
```

2. Write a function prototype for the above function.

```
void menu( void );
```

3. Write a function called print which prints a text string passed to it as a parameter (ie, a character based array).

```
void print( char message[] )
{
        printf("%s, message );
}
```

4. Write a function prototype for the above function print.

```
void print( char [] );
```

5. Write a function called total, which totals the sum of an integer array passed to it (as the first parameter) and returns the total of all the elements as an integer. Let the second parameter to the function be an integer which contains the number of elements of the array.

```
int total( int array[], int elements )
{
        int loop, sum;

        for( loop = 0, sum = 0; loop < elements; loop++ )
```

```
                        sum += array[loop];
                return sum;
        }
```

6. Write a function prototype for the above function.

```
        int total( int [], int );
```

©Copyright B Brown. 1984-2000. All rights reserved.

Previous

C Programming

Practise Exercise 8: Functions

To run this test requires a JavaScript enabled browser

---

1. The function called *menu* which prints the text string "Menu choices", and does not pass any data back, and does not accept any data as parameters, looks like

○ **function 1**

```
void menu( void ) {
        printf("Menu choices");
}
```

○ **function 2**

```
int menu( void ) {
        printf("Menu choices");
}
```

○ **function 3**

```
int menu( char string[] ) {
        printf("%s", string);
}
```

---

2. A function prototype for the above function looks like

○ int menu(char []);
○ void menu(char []);
○ void menu(void);
○ int menu(void);

---

3. A function called *print* which prints a text string passed to it as a parameter (ie, a character based array), looks like

○ **function 1**

```
int print( char string[] ) {
        printf("%s", string);
```

```
        }
```

○ **function 2**

```
        void print( char string[] ) {
                printf("Menu choices");
        }
```

○ **function 3**

```
        void print( char string[] ) {
                printf("%s", string);
        }
```

---

4. A function prototype for the above function *print* looks like

○ int print(char []);
○ void print(char []);
○ void print(void);
○ int print(void);

---

5. A function called *total*, totals the sum of an integer array passed to it (as the first parameter) and returns the total of all the elements as an integer. Let the second parameter to the function be an integer which contains the number of elements of the array.

○ **function 1**

```
        int total( int numbers[], int elements ) {
                int total = 0, loop;
                for( loop = 0; loop < elements; loop++ )
                        total = total + numbers[loop];
                return total;
        }
```

○ **function 2**

```
        int total( int numbers[], int elements ) {
                int total = 0, loop;
                for( loop = 0; loop <= elements; loop++ )
                        total = total + numbers[loop];
                return total;
        }
```

○ **function 3**

```
        int total( int numbers[], int elements ) {
                int total, loop;
```

```
        for( loop = 0; loop > elements; loop++ )
                total = total + numbers[loop];
        return total;
}
```

---

6. A function prototype for the above function looks like

- ○ int total(char []);
- ○ int total(int [], int);
- ○ void total(char [], int);
- ○ int total(void);

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

INDEX | Previous | Next

Handling User Input In C

scanf() has problems, in that if a user is expected to type an integer, and types a string instead, often the program bombs. This can be overcome by reading all input as a string (use *getchar()*), and then converting the string to the correct data type.

```
/* example one, to read a word at a time */
#include <stdio.h>
#include <ctype.h>
#define MAXBUFFERSIZE   80

void cleartoendofline( void );  /* ANSI function prototype */

void cleartoendofline( void )
{
	char ch;
	ch = getchar();
	while( ch != '\n' )
		ch = getchar();
}

main()
{
	char    ch;                     /* handles user input */
	char    buffer[MAXBUFFERSIZE];  /* sufficient to handle one line */
	int     char_count;             /* number of characters read for this line */
	int     exit_flag = 0;
	int     valid_choice;

	while( exit_flag  == 0 ) {
		printf("Enter a line of text (<80 chars)\n");
		ch = getchar();
		char_count = 0;
		while( (ch != '\n')  &&  (char_count < MAXBUFFERSIZE)) {
			buffer[char_count++] = ch;
			ch = getchar();
		}
		buffer[char_count] = 0x00;      /* null terminate buffer */
		printf("\nThe line you entered was:\n");
		printf("%s\n", buffer);

		valid_choice = 0;
		while( valid_choice == 0 ) {
			printf("Continue (Y/N)?\n");
			scanf(" %c", &ch );
			ch = toupper( ch );
			if((ch == 'Y') || (ch == 'N') )
				valid_choice = 1;
			else
				printf("\007Error: Invalid choice\n");
			cleartoendofline();
```

```
                }
                if( ch == 'N' ) exit_flag = 1;
        }
}
```

Another Example, read a number as a string

```
/* example two, reading a number as a string */
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#define MAXBUFFERSIZE   80

void cleartoendofline( void );          /* ANSI function prototype */

void cleartoendofline( void )
{
        char ch;
        ch = getchar();
        while( ch != '\n' )
                ch = getchar();
}

main()
{
        char    ch;                     /* handles user input */
        char    buffer[MAXBUFFERSIZE];  /* sufficient to handle one line */
        int     char_count;             /* number of characters read for this line */
        int     exit_flag = 0, number, valid_choice;

        while( exit_flag  == 0 ) {
                valid_choice = 0;
                while( valid_choice == 0 ) {
                        printf("Enter a number between 1 and 1000\n");
                        ch = getchar();
                        char_count = 0;
                        while( (ch != '\n')  &&  (char_count < MAXBUFFERSIZE)) {
                                buffer[char_count++] = ch;
                                ch = getchar();
                        }
                        buffer[char_count] = 0x00;      /* null terminate buffer */
                        number = atoi( buffer );
                        if( (number < 1) || (number > 1000) )
                                printf("\007Error. Number outside range 1-1000\n");
                        else
                                valid_choice = 1;
                }
                printf("\nThe number you entered was:\n");
                printf("%d\n", number);

                valid_choice = 0;
                while( valid_choice == 0 ) {
                        printf("Continue (Y/N)?\n");
                        scanf(" %c", &ch );
```

```
                        ch = toupper( ch );
                        if((ch == 'Y') || (ch == 'N') )
                                valid_choice = 1;
                        else
                                printf("\007Error: Invalid choice\n");
                        cleartoendofline();
                }
                if( ch == 'N' ) exit_flag = 1;
        }
}
```

[Other validation examples]

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

More Data Validation

Consider the following program

```
#include <stdio.h>

main() {
        int number;

        printf("Please enter a number\n");
        scanf("%d", &number );
        printf("The number you entered was %d\n", number );
}
```

The above program has several problems

- the input is not validated to see if its the correct data type
- it is not clear if there are explicit number ranges expected
- the program might crash if an incorrect data type was entered

Perhaps the best way of handling input in C programs is to treat all input as a sequence of characters, and then perform the necessary data conversion.

At this point we shall want to explore some other aspects also, like the concepts of

- trapping data at the source
- the domino/ripple effect

Trapping Data At The Source

This means that the validation of data as to its correct range/limit and data type is best done at the point of entry. The benefits of doing this at the time of data entry are

- less cost later in the program maintenance phase (because data is already validated)
- programs are easier to maintain and modify
- reduces the chances of incorrect data crashing the program later on

The Ripple Through Effect

This refers to the problem of incorrect data which is allowed to propagate through the program. An example of this is sending invalid data to a function to process.

By trapping data at the source, and ensuring that it is correct as to its data type and range, we ensure that bad data cannot be passed onwards. This makes the code which works on processing the data simpler to write and thus reduces errors.

An example

Lets look at the case of wanting to handle user input. Now, we know that users of programs out there in user-land are a bunch of annoying people who spend most of their time inventing new and more wonderful ways of making our programs crash.

Lets try to implement a sort of general purpose way of handling data input, as a replacement to *scanf()*. To do this, we will implement a function which reads the input as a sequence of characters.

The function is *readinput()*, which, in order to make it more versatile, accepts several parameters,

- a character array to store the inputted data
- an integer which specifies the data type to read, STRING, INTEGER, ALPHA
- an integer which specifies the amount of digits/characters to read

We have used some of the functions covered in ctype.h to check the data type of the inputted data.

```
/* version 1.0 */
#include <stdio.h>
#include <ctype.h>

#define MAX      80          /* maximum length of buffer         */
#define DIGIT     1          /* data will be read as digits 0-9   */
#define ALPHA     2          /* data will be read as alphabet A-Z */
#define STRING    3          /* data is read as ASCII            */

void readinput( char buff[], int mode, int limit ) {
        int ch, index = 0;

        ch = getchar();
        while( (ch != '\n') && (index < limit) ) {
                switch( mode ) {
                        case DIGIT:
                                if( isdigit( ch ) ) {
                                        buff[index] = ch;
                                        index++;
                                }
                                break;
                        case ALPHA:
                                if( isalpha( ch ) ) {
                                        buff[index] = ch;
                                        index++;
                                }
                                break;
                        case STRING:
                                if( isascii( ch ) ) {
                                        buff[index] = ch;
                                        index++;
                                }
                                break;
                        default:
```

```
                                /* this should not occur */
                                break;
                }
                ch = getchar();
        }
        buff[index] = 0x00;  /* null terminate input */
}

main() {
        char buffer[MAX];
        int number;

        printf("Please enter an integer\n");
        readinput( buffer, DIGIT, MAX );
        number = atoi( buffer );
        printf("The number you entered was %d\n", number );
}
```

Of course, there are improvements to be made. We can change *readinput* to return an integer value which represents the number of characters read. This would help in determining if data was actually entered. In the above program, it is not clear if the user actually entered any data (we could have checked to see if buffer was an empty array).

So lets now make the changes and see what the modified program looks like

```
/* version 1.1 */
#include <stdio.h>
#include <ctype.h>

#define MAX      80         /* maximum length of buffer          */
#define DIGIT     1         /* data will be read as digits 0-9   */
#define ALPHA     2         /* data will be read as alphabet A-Z */
#define STRING    3         /* data is read as ASCII             */

int readinput( char buff[], int mode, int limit ) {
        int ch, index = 0;

        ch = getchar();
        while( (ch != '\n') && (index < limit) ) {
                switch( mode ) {
                        case DIGIT:
                                if( isdigit( ch ) ) {
                                        buff[index] = ch;
                                        index++;
                                }
                                break;
                        case ALPHA:
                                if( isalpha( ch ) ) {
```

```
                                        buff[index] = ch;
                                        index++;
                                }
                                break;
                        case STRING:
                                if( isascii( ch ) ) {
                                        buff[index] = ch;
                                        index++;
                                }
                                break;
                        default:
                                /* this should not occur */
                                break;
                }
                ch = getchar();
        }
        buff[index] = 0x00;  /* null terminate input */
        return index;
}

main() {
        char buffer[MAX];
        int number, digits = 0;

        while( digits == 0 ) {
                printf("Please enter an integer\n");
                digits = readinput( buffer, DIGIT, MAX );
                if( digits != 0 ) {
                        number = atoi( buffer );
                        printf("The number you entered was %d\n", number );
                }
        }
}
```

The second version is a much better implementation.

Other validation examples

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

INDEX Previous Next

Controlling the cursor position

The following characters, placed after the \ character in a *printf()* statement, have the following effect.

Modifier	Meaning
\b	*backspace*
\f	*form feed*
\n	*new line*
\r	*carriage return*
\t	*horizontal tab*
\v	*vertical tab*
**	*backslash*
\"	*double quote*
\'	*single quote*
\<enter>	*line continuation*
\nnn	*nnn = octal character value*
\0xnn	*nn = hexadecimal value (some compilers only)*

printf("\007Attention, that was a beep!\n");

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

FORMATTERS FOR scanf()

The following characters, after the % character, in a scanf argument, have the following effect.

Modifer	Meaning
d	read a decimal integer
o	read an octal value
x	read a hexadecimal value
h	read a short integer
l	read a long integer
f	read a float value
e	read a double value
c	read a single character
s	read a sequence of characters, stop reading when an enter key or whitespace character [tab or space]
[...]	Read a character string. The characters inside the brackets indicate the allow-able characters that are to be contained in the string. If any other character is typed, the string is terminated. If the first characteris a ^, the remaining characters inside the brackets indicate that typing them will terminate the string.
*	this is used to skip input fields

Example of scanf() modifiers

```
int number;
char text1[30], text2[30];

scanf("%s %d %*f %s", text1, &number, text2);
```

If the user response is,

```
Hello 14 736.55 uncle sam
```

then

```
text1 = hello, number = 14, text2 = uncle
```

and the next call to the scanf function will continue from where the last one left off, so if

```
scanf("%s ", text2);
```

was the next call, then

```
text2 = sam
```

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

INDEX Previous Next

PRINTING OUT THE ASCII VALUES OF CHARACTERS
Enclosing the character to be printed within single quotes will instruct the compiler to print out the Ascii value of the enclosed character.

```
        printf("The character A has a value of %d\n", 'A');
```

The program will print out the integer value of the character A.

EXERCISE C20
What would the result of the following operation be?

```
        int c;
        c = 'a' + 1;
        printf("%c\n", c);
```

Answer

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

Previous

PRINTING OUT THE ASCII VALUES OF CHARACTERS

EXERCISE C20
What would the result of the following operation be?

```
int c;
c = 'a' + 1;
printf("%c\n", c);
```

The program adds one to the value 'a', resulting in the value 'b' as the value which is assigned to the variable c.

©Copyright B Brown. 1984-2000. All rights reserved.
Previous

C Programming

INDEX | Previous | Next

BIT OPERATIONS
C has the advantage of direct bit manipulation and the operations available are,

Operation	Operator	Comment	Value of Sum before	Value of sum after
AND	&	sum = sum & 2;	4	0
OR	|	sum = sum | 2;	4	6
Exclusive OR	^	sum = sum ^ 2;	4	6
1's Complement	~	sum = ~sum;	4	-5
Left Shift	<<	sum = sum << 2;	4	16
Right Shift	>>	sum = sum >> 2;	4	1

```
/* Example program illustrating << and >> */
#include <stdio.h>

main()
{
        int  n1 = 10, n2 = 20, i = 0;

        i = n2 << 4;  /* n2 shifted left four times */
        printf("%d\n", i);
        i = n1 >> 5;  /* n1 shifted right five times */
        printf("%d\n", i);
}
```

Sample Program Output
```
320
0
```

```
/* Example program using EOR operator  */
#include <stdio.h>

main()
{
        int  value1 = 2, value2 = 4;

        value1 ^= value2;
        value2 ^= value1;
        value1 ^= value2;
        printf("Value1 = %d, Value2 = %d\n", value1, value2);
}
```

Sample Program Output

```
Value1 = 4, Value2 = 2
```

```
/* Example program using AND operator  */
#include <stdio.h>

main()
{
        int  loop;

        for( loop = 'a'; loop <= 'f'; loop++ )
                printf("Loop = %c, AND 0xdf = %c\n", loop, loop & 0xdf);
}
```

Sample Program Output

```
Loop = a, AND 0xdf = A
Loop = b, AND 0xdf = B
Loop = c, AND 0xdf = C
Loop = d, AND 0xdf = D
Loop = e, AND 0xdf = E
Loop = f, AND 0xdf = F
```

© Copyright B Brown. 1984-2000. All rights reserved.

C Programming

INDEX | Previous | Next

STRUCTURES

A Structure is a data type suitable for grouping data elements together. Lets create a new data structure suitable for storing the date. The elements or fields which make up the structure use the four basic data types. As the storage requirements for a structure cannot be known by the compiler, a definition for the structure is first required. This allows the compiler to determine the storage allocation needed, and also identifies the various sub-fields of the structure.

```
struct   date {
        int  month;
        int  day;
        int  year;
};
```

This declares a NEW data type called *date*. This date structure consists of three basic data elements, all of type integer. **This is a definition to the compiler.** It does not create any storage space and cannot be used as a variable. In essence, its a new data type keyword, like *int* and *char*, and can now be used to create variables. Other data structures may be defined as consisting of the same composition as the *date* structure,

```
struct  date   todays_date;
```

defines a variable called *todays_date* to be of the same data type as that of the newly defined data type struct *date*.

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX | Previous | Next

C Programming

ASSIGNING VALUES TO STRUCTURE ELEMENTS

To assign todays date to the individual elements of the structure *todays_date*, the statement

```
todays_date.day = 21;
todays_date.month = 07;
todays_date.year = 1985;
```

is used. NOTE the use of the .element to reference the individual elements within *todays_date*.

```
/* Program to illustrate a structure */
#include <stdio.h>

struct date {                   /* global definition of type date */
        int month;
        int day;
        int year;
};

main()
{

        struct date  today;

        today.month = 10;
        today.day = 14;
        today.year = 1995;

        printf("Todays date is %d/%d/%d.\n", \
                today.month, today.day, today.year );
}
```

CLASS EXERCISE C21

Write a program in C that prompts the user for todays date, calculates tomorrows date, and displays the result. Use structures for todays date, tomorrows date, and an array to hold the days for each month of the year. Remember to change the month or year as necessary.

Answer

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

Previous

CLASS EXERCISE C21

Write a program in C that prompts the user for todays date, calculates tomorrows date, and displays the result. Use structures for todays date, tomorrows date, and an array to hold the days for each month of the year. Remember to change the month or year as necessary.

```
        #include <stdio.h>

        struct date {
                int day, month, year;
        };

        int days[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
        struct date today, tommorrow;

        void gettodaysdate( void );

        void gettodaysdate( void )
        {
                int valid = 0;

                while( valid == 0 ) {
                        printf("Enter in the current year (1990-2000)-->");
                        scanf("&d", &today.year);
                        if( (today.year < 1990) || (today.year > 1999) )
                                printf("\007Invalid year\n");
                        else
                                valid = 1;
                }
                valid = 0;
                while( valid == 0 ) {
                        printf("Enter in the current month (1-12)-->");
                        scanf("&d", &today.month);
                        if( (today.month < 1) || (today.month > 12) )
                                printf("\007Invalid month\n");
                        else
                                valid = 1;
                }
                valid = 0;
                while( valid == 0 ) {
                        printf("Enter in the current day (1-%d)-->",
days[today.month-1]);
                        scanf("&d", &today.day);
                        if( (today.day < 1) || (today.day > days[today.month-1]) )
                                printf("\007Invalid day\n");
                        else
                                valid = 1;
                }
        }
```

```
main()
{

        gettodaysdate();
        tommorrow = today;
        tommorrow.day++;
        if( tommorrow.day > days[tommorrow.month-1] ) {
                tommorrow.day = 1;
                tommorrow.month++;
                if( tommorrow.month > 12 ) {
                        tommorrow.year++;
                        tommorrow.month = 1;
                }
        }
        printf("Tommorrows date is %02d:%02d:%02d\n", \
                tommorrow.day, tommorrow.month, tommorrow.year );
}
```

©Copyright B Brown. 1984-2000. All rights reserved.

Previous

C Programming


```
/* TIME.C  Program updates time by 1 second using functions */
#include <stdio.h>

struct time {
    int hour, minutes, seconds;
};

void time_update( struct time );        /* ANSI function prototype */

/* function to update time by one second */
void time_update( struct time new_time )
{
        ++new_time.seconds;
        if( new_time.seconds == 60) {
                new_time.seconds = 0;
                ++new_time.minutes;
                if(new_time.minutes == 60) {
                        new_time.minutes = 0;
                        ++new_time.hour;
                        if(new_time.hour == 24)
                                new_time.hour = 0;
                }
        }
}

main()
{
        struct time current_time;

        printf("Enter the time (hh:mm:ss):\n");
        scanf("%d:%d:%d", \

&current_time.hour,&current_time.minutes,&current_time.seconds);
        time_update ( current_time);
        printf("The new time is %02d:%02d:%02d\n",current_time.hour, \
                current_time.minutes, current_time.seconds);
}
```

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

INITIALIZING STRUCTURES

This is similar to the initialization of arrays; the elements are simply listed inside a pair of braces, with each element separated by a comma. The structure declaration is preceded by the keyword *static*

```
static struct date today = { 4,23,1998 };
```

ARRAYS OF STRUCTURES

Consider the following,

```
struct  date  {
     int month, day, year;
};
```

Lets now create an array called *birthdays* of the same data type as the structure *date*

```
struct date birthdays[5];
```

This creates an array of 5 elements which have the structure of *date*.

```
birthdays[1].month = 12;
birthdays[1].day   = 04;
birthdays[1].year  = 1998;
--birthdays[1].year;
```

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

INDEX Previous Next

STRUCTURES AND ARRAYS
Structures can also contain arrays.

```
struct month {
        int  number_of_days;
        char name[4];
};

static struct month this_month = { 31, "Jan" };

this_month.number_of_days = 31;
strcpy( this_month.name, "Jan" );
printf("The month is %s\n", this_month.name );
```

Note that the array *name* has an extra element to hold the end of string nul character.

VARIATIONS IN DECLARING STRUCTURES
Consider the following,

```
struct date {
     int month, day, year;
} todays_date, purchase_date;
```

or another way is,

```
struct date {
     int month, day, year;
} todays_date = { 9,25,1985 };
```

or, how about an array of structures similar to date,

```
struct date {
     int month, day, year;
} dates[100];
```

Declaring structures in this way, however, prevents you from using the structure definition later in the program. The structure definition is thus bound to the variable name which follows the right brace of the structures definition.

CLASS EXERCISE C22

Write a program to enter in five dates, Store this information in an array of structures.

Answer

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

Previous

CLASS EXERCISE C22

```
#include <stdio.h>

struct  date  {        /* Global definition of date */
        int day, month, year;
};

main()
{
        struct date dates[5];
        int i;

        for( i = 0; i < 5; ++i ) {
                printf("Please enter the date (dd:mm:yy)" );
                scanf("%d:%d:%d", &dates[i].day, &dates[i].month,
                        &dates[i].year );
        }
}
```

©Copyright B Brown. 1984-2000. All rights reserved.

Previous

C Programming

INDEX Previous Next

STRUCTURES WHICH CONTAIN STRUCTURES
Structures can also contain structures. Consider where both a date and time structure are combined into a single structure called *date_time*, eg,

```
struct date {
        int  month, day, year;
};

struct time {
        int  hours, mins, secs;
};

struct date_time {
        struct date sdate;
        struct time stime;
};
```

This declares a structure whose elements consist of two other previously declared structures. Initialization could be done as follows,

```
static struct date_time today = { { 2, 11, 1985 }, { 3, 3,33 } };
```

which sets the *sdate* element of the structure *today* to the eleventh of February, 1985. The *stime* element of the structure is initialized to three hours, three minutes, thirty-three seconds. Each item within the structure can be referenced if desired, eg,

```
++today.stime.secs;
if( today.stime.secs == 60 ) ++today.stime.mins;
```

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

INDEX Previous Next

BIT FIELDS

Consider the following data elements defined for a PABX telephone system.

```
flag = 1 bit
off_hook = 1 bit
status =  2 bits
```

In C, these can be defined as a structure, and the number of bits each occupy can be specified.

```
struct packed_struct {
        unsigned int flag:1;
        unsigned int off_hook:1;
        unsigned int status:2;
} packed_struct1;
```

The :1 following the variable *flag* indicates that flag occupies a single bit. The C compiler will assign all the above fields into a single word.

Assignment is as follows,

```
packed_struct1.flag = 0;
packed_struct1.status = 3;
if( packed_struct1.flag )
        .............
```

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

INDEX Previous Next

Practise Exercise 9: Structures

JavaScript compatible inter-active version of this test.

---

1. Define a structure called *record* which holds an integer called *loop*, a character array of 5 elements called *word*, and a float called *sum*.

2. Declare a structure variable called *sample*, defined from a structure of type *record*.

3. Assign the value 10 to the field *loop* of the *sample* structure of type record.

4. Print out (using printf) the value of the word array of the *sample* structure.

5. Define a new structure called *birthdays*, whose fields are a structure of type *time* called *btime*, and a structure of type *date*, called *bdate*.

Answers

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

Previous

Practise Exercise 9: Structures

1. Define a structure called *record* which holds an integer called *loop*, a character array of 5 elements called *word*, and a float called *sum*.

```
struct record {
        int loop;
        char word[5];
        float sum;
};
```

2. Declare a structure variable called *sample*, defined from a structure of type *struct record*.

```
struct record sample;
```

3. Assign the value 10 to the field *loop* of the *sample* structure of type *struct record*.

```
sample.loop = 10;
```

4. Print out (using printf) the value of the word array of the *sample* structure.

```
printf("%s", sample.word );
```

5. Define a new structure called *birthdays*, whose fields are a structure of type *struct time* called *btime*, and a structure of type *struct date*, called *bdate*.

```
struct birthdays {
        struct time btime;
        struct date bdate;
};
```

©Copyright B Brown. 1984-2000. All rights reserved.

Previous

C Programming

Previous Next

Practise Exercise 9: Structures

To run this test requires a JavaScript enabled browser

---

1. A structure called *record* which holds an integer called *loop*, a character array of 5 elements called *word*, and a float called *sum*, looks like

○ **Structure 1**

```
struct record {
        int loop;
        char word[5];
        float sum;
};
```

○ **Structure 2**

```
type structure record {
        loop : integer;
        word : array[0..4] of char;
        sum : real;
};
```

○ **Structure 3**

```
type record {
        integer loop;
        char word[4];
        float sum;
}
```

---

2. The statement which declares a structure variable called *sample*, defined from a structure of type *struct record*, is

○ type sample : record;
○ struct sample;
○ struct record sample;
○ declare sample as type record;

---

3. The statment that assigns the value 10 to the field *loop* of the *sample* structure (which is of type *struct record*), is

- loop = 10;
- sample.loop = 10;
- record.sample.loop = 10;
- record.loop = 10;

---

4. The statement that prints out (using printf) the value of the word array of the *sample* structure is

- printf("%d", sample);
- printf("%s", word);
- printf("%c", sample-word);
- printf("%s", sample.word);

---

5. The correct definition for a structure called *birthdays*, whose fields are a structure of type *struct time* called *btime*, and a structure of type *struct date*, called *bdate*, is

- **Structure 1**

```
        birthdays {
                time btime;
                date bdate;
        };
```

- **Structure 2**

```
        struct birthdays {
                struct time btime;
                struct date bdate;
        };
```

- **Structure 3**

```
        struct birthdays {
                struct bdate date;
                struct btime time;
        };
```

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

INDEX Previous Next

DATA CONVERSION

The following functions convert between data types.

```
        atof()          converts an ascii character array to a float
        atoi()          converts an ascii character array to an integer
        itoa()          converts an integer to a character array
```

Example

```
        /* convert a string to an integer */
        #include <stdio.h>
        #include <stdlib.h>

        char string[] = "1234";

        main()
        {
                int sum;
                sum = atoi( string );
                printf("Sum = %d\n", sum );
        }
```

```
/* convert an integer to a string */
        #include <stdio.h>
        #include <stdlib.h>

        main()
        {
                int sum;
                char buff[20];

                printf("Enter in an integer ");
                scanf(" %d", &sum );
                printf( "As a string it is %s\n", itoa( sum, buff, 10 ) );
        }
```

Note that itoa() takes three parameters,

- the integer to be converted
- a character buffer into which the resultant string is stored
- a radix value (10=decimal,16=hexadecimal)

In addition, itoa() returns a pointer to the resultant string.

© Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

INDEX Previous Next

FILE INPUT/OUTPUT
To work with files, the library routines must be included into your programs. This is done by the statement,

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

as the first statement of your program.

USING FILES

- **Declare a variable of type FILE**
  To use files in C programs, you must declare a file variable to use. This variable must be of type **FILE**, and be declared as a pointer type.

  FILE is a predefined type. You declare a variable of this type as

  ```
                  FILE  *in_file;
  ```

  This declares *infile* to be a pointer to a file.
- **Associate the variable with a file using fopen()**
  Before using the variable, it is associated with a specific file by using the *fopen()* function, which accepts the pathname for the file and the access mode (like reading or writing).

  ```
                  in_file = fopen( "myfile.dat", "r" );
  ```

  In this example, the file **myfile.dat** in the current directory is opened for **read** access.
- **Process the data in the file**
  Use the appropriate file routines to process the data
- **When finished processing the file, close it**
  Use the *fclose()* function to close the file.

  ```
                  fclose( in_file );
  ```

The following illustrates the *fopen* function, and adds testing to see if the file was opened successfully.

```
        #include <stdio.h>
        /* declares pointers to an input file, and the fopen function */
        FILE   *input_file, *fopen ();

        /* the pointer of the input file is assigned the value returned from the
fopen call. */
        /* fopen tries to open a file called datain for read only. Note that */
        /* "w" = write, and "a" = append.   */
        input_file = fopen("datain", "r");

        /* The pointer is now checked. If the file was opened, it will point to the
first */
        /* character of the file. If not, it will contain a NULL or 0. */
```

```
        if( input_file == NULL ) {
                printf("*** datain could not be opened.\n");
                printf("returning to dos.\n");
                exit(1);
        }
```

NOTE: Consider the following statement, which combines the opening of the file and its test to see if it was successfully opened into a single statement.

```
        if(( input_file = fopen ("datain", "r" )) == NULL ) {
                printf("*** datain could not be opened.\n");
                printf("returning to dos.\n");
                exit(1);
        }
```

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

INPUTTING/OUTPUTTING SINGLE CHARACTERS

Single characters may be read/written with files by use of the two functions, *getc()*, and *putc()*.

```
int ch;

ch = getc( input_file );   /*  assigns character to ch  */
```

The *getc()* also returns the value EOF (end of file), so

```
while( (ch = getc( input_file )) != EOF )
        ......................
```

NOTE that the *putc/getc* are similar to *getchar/putchar* except that arguments are supplied specifying the I/O device.

```
putc('\n', output_file ); /* writes a newline to output file */
```

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

INDEX | Previous | Next

CLOSING FILES

When the operations on a file are completed, it is closed before the program terminates. This allows the operating system to cleanup any resources or buffers associated with the file. The *fclose()* function is used to close the file and flush any buffers associated with the file.

```
fclose( input_file );
fclose( output_file );
```

COPYING A FILE

The following demonstrates copying one file to another using the functions we have just covered.

```
#include <stdio.h>

main()   /* FCOPY.C    */
{
        char in_name[25], out_name[25];
        FILE *in_file, *out_file, *fopen ();
        int c;

        printf("File to be copied:\n");
        scanf("%24s", in_name);
        printf("Output filename:\n");
        scanf("%24s", out_name);

        in_file = fopen ( in_name, "r");

        if( in_file == NULL )
                printf("Cannot open %s for reading.\n", in_name);
        else {
                out_file = fopen (out_name, "w");
                if( out_file == NULL )
                        printf("Can't open %s for writing.\n",out_name);
                else {
                        while( (c = getc( in_file)) != EOF )
                                putc (c, out_file);
                        putc (c, out_file);   /* copy EOF */
                        printf("File has been copied.\n");
                        fclose (out_file);
                }
                fclose (in_file);
        }
}
```

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

TESTING FOR THE End Of File TERMINATOR (feof)
This is a built in function incorporated with the *stdio.h* routines. It returns 1 if the file pointer is at the end of the file.

```
	if( feof ( input_file ))
		printf("Ran out of data.\n");
```

THE fprintf AND fscanf STATEMENTS
These perform the same function as *printf* and *scanf*, but work on files. Consider,

```
	fprintf(output_file, "Now is the time for all..\n");
	fscanf(input_file, "%f", &float_value);
```

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

THE fgets AND fputs STATEMENTS

These are useful for reading and writing entire lines of data to/from a file. If *buffer* is a pointer to a character array and *n* is the maximum number of characters to be stored, then

```
fgets (buffer, n, input_file);
```

will read an entire line of text (max chars = n) into *buffer* until the newline character or n=max, whichever occurs first. The function places a NULL character after the last character in the buffer. The function will be equal to a NULL if no more data exists.

```
fputs (buffer, output_file);
```

writes the characters in *buffer* until a NULL is found. The NULL character is not written to the *output_file*.

NOTE: fgets does not store the newline into the buffer, fputs will append a newline to the line written to the output file.

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

INDEX Previous Next

Practise Exercise 9A: File Handling

JavaScript compatible inter-active version of this test.

1. Define an input file handle called *input_file*, which is a pointer to a type FILE.

2. Using *input_file*, open the file *results.dat* for read mode as a text file.

3. Write C statements which tests to see if *input_file* has opened the data file successfully. If not, print an error message and exit the program.

4. Write C code which will read a line of characters (terminated by a \n) from *input_file* into a character array called *buffer*. NULL terminate the buffer upon reading a \n.

5. Close the file associated with *input_file*.

Answers

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

Previous

Practise Exercise 9A: File Handling

1. Define an input file handle called *input_file*, which is a pointer to a type FILE.

```
FILE *input_file;
```

2. Using *input_file*, open the file *results.dat* for read mode.

```
input_file = fopen( "results.dat", "r" );
```

3. Write C statements which tests to see if *input_file* has opened the data file successfully. If not, print an error message and exit the program.

```
if( input_file == NULL ) {
        printf("Unable to open file.\n");\
        exit(1);
}
```

4. Write C code which will read a line of characters (terminated by a \n) from *input_file* into a character array called *buffer*. NULL terminate the buffer upon reading a \n.

```
int ch, loop = 0;

ch = fgetc( input_file );
while( (ch != '\n') && (ch != EOF) ) {
        buffer[loop] = ch;
        loop++;
        ch = fgetc( input_file );
}
buffer[loop] = NULL;
```

5. Close the file associated with *input_file*.

```
fclose( input_file );
```

©Copyright B Brown. 1984-2000. All rights reserved.

Previous

C Programming

Previous Next

Practise Exercise 9A: File Handling

To run this test requires a JavaScript enabled browser

---

1. The statement that defines an input file handle called *input_file*, which is a pointer to type FILE, is

- type input_file as FILE;
- FILE *input_file;
- input_file FILE;
- *FILE input_file;

---

2. Using *input_file*, open the file *results.dat* for read mode.

- input_file = "results.dat" opened as "r";
- open input_file as "results.dat" for "r";
- fopen(input_file, "results.dat", "r");
- input_file = fopen("results.dat", "r");

---

3. Write C statements which tests to see if *input_file* has opened the data file successfully. If not, print an error message and exit the program.

Test 1

```
if( input_file == NULL ) {
        printf("Unable to open file.\n");\
        exit(1);
}
```

Test 2

```
if( input_file != NULL ) {
        printf("Unable to open file.\n");\
        exit(1);
}
```

Test 3

```
while( input_file = NULL ) {
        printf("Unable to open file.\n");\
```

```
        exit(1);
}
```

4. Write C code which will read a line of characters (terminated by a \n) from *input_file* into a character array called *buffer*. NULL terminate the buffer upon reading a \n.

○ **Example 1**

```
int ch, loop = 0;

ch = fgetc( input_file );
while( (ch != '\n') && (ch != EOF) ) {
        buffer[loop] = ch;
        loop++;
        ch = fgetc( input_file );
}
buffer[loop] = NULL;
```

○ **Example 2**

```
int ch, loop = 0;

ch = fgetc( input_file );
while( (ch = '\n') && (ch = EOF) ) {
        buffer[loop] = ch;
        loop--;
        ch = fgetc( input_file );
}
buffer[loop] = NULL;
```

○ **Example 3**

```
int ch, loop = 0;

ch = fgetc( input_file );
while( (ch <> '\n') && (ch != EOF) ) {
        buffer[loop] = ch;
        loop++;
        ch = fgetc( input_file );
}
buffer[loop] = -1;
```

5. Close the file associated with *input_file*.

○ close input_file;

○ fclose(input_file);

- fcloseall();
- input_file(fclose);

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

File handling using open(), read(), write() and close()
The previous examples of file handling deal with File Control Blocks (FCB). Under MSDOS v3.x (or greater) and UNIX systems, file handling is often done using handles, rather than file control blocks.

Writing programs using handles ensures portability of source code between different operating systems. Using handles allows the programmer to treat the file as a stream of characters.

---

open()

```
#include <fcntl.h>
int  open(  char  *filename,  int  access,  int  permission  );
```

The available access modes are

```
O_RDONLY                O_WRONLY                O_RDWR
O_APPEND                O_BINARY                O_TEXT
```

The permissions are

```
S_IWRITE        S_IREAD S_IWRITE | S_IREAD
```

The *open()* function returns an integer value, which is used to refer to the file. If un- successful, it returns -1, and sets the global variable *errno* to indicate the error type.

---

read()

```
#include  <fcntl.h>
int  read(  int  handle,  void  *buffer,  int  nbyte );
```

The *read()* function attempts to read nbytes from the file associated with handle, and places the characters read into *buffer*. If the file is opened using O_TEXT, it removes carriage returns and detects the end of the file.

The function returns the number of bytes read. On end-of-file, 0 is returned, on error it returns -1, setting errno to indicate the type of error that occurred.

---

write()

```
#include  <fcntl.h>
int  write(  int  handle,  void  *buffer,  int  nbyte  );
```

The *write()* function attempts to write nbytes from *buffer* to the file associated with handle. On text files, it expands each LF to a CR/LF.

The function returns the number of bytes written to the file. A return value of -1 indicates an error, with errno set appropriately.

---

close()

```
#include  <fcntl.h>
int  close(  int  handle  );
```

The *close()* function closes the file associated with handle. The function returns 0 if successful, -1 to indicate an error, with errno set appropriately.

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

INDEX Previous Next

File handling example of a goods re-ordering program
The following program handles an ASCII text file which describes a number of products, and reads each product into a structure with the program.

```
/* File handling example for PR101      */
/* processing an ASCII file of records */
/* Written by B. Brown, April 1994      */
/*                                      */

/* process a goods file, and print out */
/* all goods where the quantity on      */
/* hand is less than or equal to the    */
/* re-order level.                      */


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

/* definition of a record of type goods */
struct goods {
   char  name[20];     /* name of product      */
   float price;        /* price of product     */
   int   quantity;     /* quantity on hand     */
   int   reorder;      /* re-order level       */
};

/* function prototypes */
void myexit( int );
void processfile( void );
void printrecord( struct goods );
int getrecord( struct goods * );

/* global data variables */
FILE *fopen(), *input_file;  /* input file pointer */

/* provides a tidy means to exit program gracefully */
void myexit( int exitcode )
{
   if( input_file != NULL )
      fclose( input_file );
```

```
   exit( exitcode );
}

/* prints a record */
void printrecord( struct goods record )
{
   printf("\nProduct name\t%s\n", record.name );
   printf("Product price\t%.2f\n", record.price );
   printf("Product quantity\t%d\n", record.quantity );
   printf("Product reorder level\t%d\n", record.reorder );
}

/* reads one record from inputfile into 'record', returns 1 for success */
int getrecord( struct goods *record )
{
   int loop = 0, ch;
   char buffer[40];

   ch = fgetc( input_file );
   /* skip to start of record */
   while( (ch == '\n') || (ch == ' ') && (ch != EOF) )
      ch = fgetc( input_file );
   if( ch == EOF ) return 0;

   /* read product name */
   while( (ch != '\n') && (ch != EOF)) {
      buffer[loop++] = ch;
      ch = fgetc( input_file );
   }
   buffer[loop] = 0;
   strcpy( record->name, buffer );
   if( ch == EOF ) return 0;

   /* skip to start of next field */
   while( (ch == '\n') || (ch == ' ') && (ch != EOF) )
      ch = fgetc( input_file );
   if( ch == EOF ) return 0;

   /* read product price */
   loop = 0;
   while( (ch != '\n') && (ch != EOF)) {
      buffer[loop++] = ch;
      ch = fgetc( input_file );
   }
   buffer[loop] = 0;
   record->price = atof( buffer );
   if( ch == EOF ) return 0;
```

```
   /* skip to start of next field */
   while( (ch == '\n') || (ch == ' ') && (ch != EOF) )
      ch = fgetc( input_file );
   if( ch == EOF ) return 0;

   /* read product quantity */
   loop = 0;
   while( (ch != '\n') && (ch != EOF)) {
      buffer[loop++] = ch;
      ch = fgetc( input_file );
   }
   buffer[loop] = 0;
   record->quantity = atoi( buffer );
   if( ch == EOF ) return 0;

   /* skip to start of next field */
   while( (ch == '\n') || (ch == ' ') && (ch != EOF) )
      ch = fgetc( input_file );
   if( ch == EOF ) return 0;

   /* read product reorder level */
   loop = 0;
   while( (ch != '\n') && (ch != EOF)) {
      buffer[loop++] = ch;
      ch = fgetc( input_file );
   }
   buffer[loop] = 0;
   record->reorder = atoi( buffer );
   if( ch == EOF ) return 0;

   return 1;  /* signify record has been read successfully */
}

/* processes file for records */
void processfile( void )
{
   struct goods record;   /* holds a record read from inputfile */

   while( ! feof( input_file )) {
      if( getrecord( &record ) == 1 ) {
         if( record.quantity <= record.reorder )
            printrecord( record );
      }
      else myexit( 1 );  /* error getting record */
   }
}
```

```
main()
{
   char filename[40];     /* name of database file */

   printf("Example Goods Re-Order File Program\n");
   printf("Enter database file ");
   scanf(" %s", filename );
   input_file = fopen( filename, "rt" );
   if( input_file == NULL ) {
      printf("Unable to open datafile %s\n", filename );
      myexit( 1 );
   }
   processfile();
   myexit( 0 );
}
```

The datafile (a standard ASCII text file) used for this example looks like

```
baked beans
1.20
10
5

greggs coffee
2.76
5
10

walls ice-cream
3.47
5
5

cadburys chocs
4.58
12
10
```

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

INDEX | Previous | Next

POINTERS
Pointers enable us to effectively represent complex data structures, to change values as arguments to functions, to work with memory which has been dynamically allocated, and to more concisely and efficiently deal with arrays. A pointer provides an indirect means of accessing the value of a particular data item. Lets see how pointers actually work with a simple example,

```
	int  count = 10, *int_pointer;
```

declares an integer *count* with a value of 10, and also an integer pointer called *int_pointer*. Note that the prefix * defines the variable to be of type pointer. To set up an indirect reference between *int_pointer* and *count*, the & prefix is used, ie,

```
	int_pointer = &count
```

This assigns the memory address of *count* to int_pointer, not the actual value of *count* stored at that address.

POINTERS CONTAIN MEMORY ADDRESSES, NOT VALUES!

To reference the value of *count* using *int_pointer*, the * is used in an assignment, eg,

```
	x = *int_pointer;
```

Since *int_pointer* is set to the memory address of *count*, this operation has the effect of assigning the contents of the memory address pointed to by *int_pointer* to the variable *x*, so that after the operation variable *x* has a value of 10.

```
	#include <stdio.h>

	main()
	{
		int count  = 10, x, *int_pointer;

		/* this assigns the memory address of count to int_pointer  */
		int_pointer = &count;

		/* assigns the value stored at the address specified by int_pointer
to x */
		x = *int_pointer;

		printf("count = %d, x = %d\n", count, x);
	}
```

This however, does not illustrate a good use for pointers.

The following program illustrates another way to use pointers, this time with characters,

```
	#include <stdio.h>

	main()
	{
```

```
                char c = 'Q';
                char *char_pointer = &c;

                printf("%c %c\n", c, *char_pointer);

                c = 'Z';
                printf("%c %c\n", c, *char_pointer);
                *char_pointer = 'Y';
                /* assigns Y as the contents of the memory address specified by
char_pointer  */

                printf("%c %c\n", c, *char_pointer);
        }
```

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

INDEX Previous Next

CLASS EXERCISE C23

Determine the output of the pointer programs P1, P2, and P3.

```
        /* P1.C  illustrating pointers */
        #include <stdio.h>

        main()
        {
                int count  = 10, x, *int_pointer;

                /* this assigns the memory address of count to int_pointer  */
                int_pointer = &count;

                /* assigns the value stored at the address specified by int_pointer
to x */
                x = *int_pointer;

                printf("count = %d, x = %d\n", count, x);
        }
```

```
/* P2.C  Further examples of pointers */
        #include <stdio.h>

        main()
        {
                char c = 'Q';
                char *char_pointer = &c;

                printf("%c %c\n", c, *char_pointer);

                c = '/';
                printf("%c %c\n", c, *char_pointer);
                *char_pointer = '(';
        /* assigns ( as the contents of the memory address specified by char_pointer
*/

                printf("%c %c\n", c, *char_pointer);
        }
```

Answers

CLASS EXERCISE C24

```
        /* P3.C  Another program with pointers */
        #include <stdio.h>

        main()
        {
                int i1, i2, *p1, *p2;
```

```
        i1 = 5;
        p1 = &i1;
        i2 = *p1 / 2 + 10;
        p2 = p1;

        printf("i1 = %d, i2 = %d, *p1 = %d, *p2 = %d\n", i1, i2, *p1, *p2);
}
```

[Answers]

© Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

INDEX Previous Next

Practise Exercise 10: Pointers

JavaScript compatible inter-active version of this test.

---

1. Declare a pointer to an integer called *address*.

2. Assign the address of a float variable *balance* to the float pointer *temp*.

3. Assign the character value 'W' to the variable pointed to by the char pointer *letter*.

4. What is the output of the following program segment?

```
int  count = 10, *temp, sum = 0;

temp = &count;
*temp = 20;
temp = &sum;
*temp = count;
printf("count = %d, *temp = %d, sum = %d\n", count, *temp, sum );
```

5. Declare a pointer to the text string "Hello" called *message*.

Answers

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

Previous

Practise Exercise 10: Pointers

1. Declare a pointer to an integer called *address*.

```
int *address;
```

2. Assign the address of a float variable *balance* to the float pointer *temp*.

```
temp = &balance;
```

3. Assign the character value 'W' to the variable pointed to by the char pointer *letter*.

```
*letter = 'W';
```

4. What is the output of the following program segment?

```
int  count = 10, *temp, sum = 0;

temp = &count;
*temp = 20;
temp = &sum;
*temp = count;
printf("count = %d, *temp = %d, sum = %d\n", count, *temp, sum );

count = 20, *temp = 20, sum = 20
```

5. Declare a pointer to the text string "Hello" called *message*.

```
char *message = "Hello";
```

©Copyright B Brown. 1984-2000. All rights reserved.

Previous

C Programming

Previous

CLASS EXERCISE C23
Determine the output of the pointer programs P1, P2, and P3.

```
        /* P1.C  illustrating pointers */
        #include <stdio.h>

        main()
        {
                int count  = 10, x, *int_pointer;

                /* this assigns the memory address of count to int_pointer  */
                int_pointer = &count;

                /* assigns the value stored at the address specified by int_pointer
to x */
                x = *int_pointer;

                printf("count = %d, x = %d\n", count, x);
        }


        count = 10, x = 10;
```

```
/* P2.C  Further examples of pointers */
        #include <stdio.h>

        main()
        {
                char c = 'Q';
                char *char_pointer = &c;

                printf("%c %c\n", c, *char_pointer);

                c = '/';
                printf("%c %c\n", c, *char_pointer);
                *char_pointer = '(';
        /* assigns ( as the contents of the memory address specified by char_pointer
*/

                printf("%c %c\n", c, *char_pointer);
        }


        Q Q
        / /
        ( (
```

```
/* P3.C  Another program with pointers */
        #include <stdio.h>

        main()
        {
                int i1, i2, *p1, *p2;

                i1 = 5;
                p1 = &i1;
                i2 = *p1 / 2 + 10;
                p2 = p1;

                printf("i1 = %d, i2 = %d, *p1 = %d, *p2 = %d\n", i1, i2, *p1, *p2);
        }


        i1 = 5, i2 = 12, *p1 = 5, *p2 = 5
```

© Copyright B Brown. 1984-2000. All rights reserved. Previous

C Programming

Previous Next

Practise Exercise 10: Pointers

To run this test requires a JavaScript enabled browser

---

1. Declare a pointer to an integer called *address*.

- int address;
- address *int;
- int *address;
- *int address;

---

2. Assign the address of a float variable *balance* to the float pointer *temp*.

- temp = &balance;
- balance = float temp;
- float temp *balance;
- &temp = balance;

---

3. Assign the character value 'W' to the variable pointed to by the char pointer *letter*.

- 'W' = *letter;
- letter = "W";
- letter = *W;
- *letter = 'W';

---

4. What is the output of the following program segment?

```
int  count = 10, *temp, sum = 0;

temp = &count;
*temp = 20;
temp = &sum;
*temp = count;
printf("count = %d, *temp = %d, sum = %d\n", count, *temp, sum );
```

- count = 2, *temp = 10, sum = 10
- count = 20, *temp = 20, sum = 20
- count = 10, *temp = 2, sum = 10
- count = 200, *temp = 0.2, sum = 1

5. Declare a pointer to the text string "Hello" called *message*.

- char message = "Hello";
- *message = "Hello";
- char *message = "Hello";
- char message = 'Hello';

©Copyright B Brown. 1984-2000. All rights reserved.

Previous Next

C Programming

INDEX | Previous | Next

POINTERS AND STRUCTURES

Consider the following,

```
struct date {
        int month, day, year;
};

struct date  todays_date, *date_pointer;

date_pointer = &todays_date;

(*date_pointer).day = 21;
(*date_pointer).year = 1985;
(*date_pointer).month = 07;

++(*date_pointer).month;
if((*date_pointer).month == 08 )
        ......
```

Pointers to structures are so often used in C that a special operator exists. The structure pointer operator, the ->, permits expressions that would otherwise be written as,

```
(*x).y
```

to be more clearly expressed as

```
x->y
```

making the if statement from above program

```
if( date_pointer->month == 08 )
        .....
```

```
/* Program to illustrate structure pointers */
#include <stdio.h>

main()
{
```

```
            struct date { int month, day, year; };
            struct date today, *date_ptr;

            date_ptr = &today;
            date_ptr->month = 9;
            date_ptr->day = 25;
            date_ptr->year = 1983;

            printf("Todays date is %d/%d/%d.\n", date_ptr->month, \
                    date_ptr->day, date_ptr->year % 100);
    }
```

So far, all that has been done could've been done without the use of pointers. Shortly, the real value of pointers will become apparent.

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

INDEX Previous Next

STRUCTURES CONTAINING POINTERS

Naturally, a pointer can also be a member of a structure.

```
struct  int_pointers {
        int   *ptr1;
        int   *ptr2;
};
```

In the above, the structure *int_pointers* is defined as containing two integer pointers, *ptr1* and *ptr2*. A variable of type struct int_pointers can be defined in the normal way, eg,

```
struct  int_pointers  ptrs;
```

The variable *ptrs* can be used normally, eg, consider the following program,

```
#include <stdio.h>
main()   /* Illustrating structures containing pointers */
{
        struct  int_pointers {  int   *ptr1, *ptr2;  };
        struct int_pointers  ptrs;
        int  i1 = 154, i2;

        ptrs.ptr1 =  &i1;
        ptrs.ptr2 =  &i2;
        *ptrs.ptr2 =  -97;
        printf("i1 = %d, *ptrs.ptr1 = %d\n", i1, *ptrs.ptr1);
        printf("i2 = %d, *ptrs.ptr2 = %d\n", i2, *ptrs.ptr2);
}
```

The following diagram may help to illustrate the connection,

```
|------------|
|  i1        |<---------------
|------------|               |
|  i2        |<-------       |
|------------|       |       |
|            |       |       |
|------------|       |       |
|  ptr1      |----------------
|------------|       |           ptrs
```

```
        |  ptr2      |--------
        |------------|
```

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

INDEX Previous Next

POINTERS AND CHARACTER STRINGS

A pointer may be defined as pointing to a character string.

```
	#include <stdio.h>

	main()
	{
		char *text_pointer = "Good morning!";

		for( ; *text_pointer != '\0'; ++text_pointer)
			printf("%c", *text_pointer);
	}
```

---

or another program illustrating pointers to text strings,

```
	#include <stdio.h>

	main()
	{
		static char *days[] = {"Sunday", "Monday", "Tuesday", "Wednesday", \
		                        "Thursday", "Friday",
"Saturday"};
		int i;

		for( i = 0; i < 6; ++i )
			printf( "%s\n", days[i]);
	}
```

Remember that if the declaration is,

```
   char *pointer = "Sunday";
```

then the null character { '\0' } is automatically appended to the end of the text string. This means that %s may be used in a *printf* statement, rather than using a *for* loop and %c to print out the contents of the pointer. The %s will print out all characters till it finds the null terminator.

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

INDEX Previous Next

Practise Exercise 11: Pointers & Structures

JavaScript compatible inter-active version of this test.

---

1. Declare a pointer to a structure of type *date* called *dates*.

2. If the above structure of type date comprises three integer fields, day, month, year, assign the value 10 to the field *day* using the *dates* pointer.

3. A structure of type *machine* contains two fields, an integer called *name*, and a char pointer called *memory*. Show what the definition of the structure looks like.

4. A pointer called *mpu641* of type machine is declared. What is the command to assign the value NULL to the field *memory*.

5. Assign the address of the character array *CPUtype* to the field *memory* using the pointer *mpu641*.

6. Assign the value 10 to the field *name* using the pointer *mpu641*.

7. A structure pointer *times* of type *time* (which has three fields, all pointers to integers, day, month and year respectively) is declared. Using the pointer *times*, update the field *day* to 10.

8. An array of pointers (10 elements) of type *time* (as detailed above in ' 7.), called *sample* is declared. Update the field *month* of the third array element to 12.

Answers

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

Previous

Practise Exercise 11: Pointers & Structures

1. Declare a pointer to a structure of type *date* called *dates*.

```
struct date *dates;
```

2. If the above structure of type date comprises three integer fields, day, month, year, assign the value 10 to the field *day* using the *dates* pointer.

```
dates->day = 10;
```

3. A structure of type *machine* contains two fields, an integer called *name*, and a char pointer called *memory*. Show what the definition of the structure looks like.

```
|-----------| <---------
|           | name       |
|-----------|            | machine
|           | memory     |
|-----------| <---------
```

4. A pointer called *mpu641* of type machine is declared. What is the command to assign the value NULL to the field *memory*.

```
mpu641->memory = (char *) NULL;
```

5. Assign the address of the character array *CPUtype* to the field *memory* using the pointer *mpu641*.

```
mpu641->memory = CPUtype;
```

6. Assign the value 10 to the field *name* using the pointer *mpu641*.

```
mpu641->name = 10;
```

7. A structure pointer *times* of type *time* (which has three fields, all pointers to integers, day, month and year respectively) is declared. Using the pointer *times*, update the field *day* to 10.

```
        *(times->day) = 10;
```

8. An array of pointers (10 elements) of type *time* (as detailed above in 7.), called *sample* is declared. Update the field *month* of the third array element to 12.

```
        *(sample[2]->month) = 12;
```

```
#include <stdio.h>

struct machine {
   int name;
   char *memory;
};

struct machine p1, *mpu641;

main()
{
   p1.name = 3;
   p1.memory = "hello";
   mpu641 = &p1;
   printf("name = %d\n", mpu641->name );
   printf("memory = %s\n", mpu641->memory );

   mpu641->name = 10;
   mpu641->memory = (char *) NULL;
   printf("name = %d\n", mpu641->name );
   printf("memory = %s\n", mpu641->memory );
}
```

```
#include <stdio.h>

struct time {
   int *day;
   int *month;
   int *year;
};

struct time t1, *times;
```

```
main()
{
   int d=5, m=12, y=1995;

   t1.day = &d;
   t1.month = &m;
   t1.year = &y;

   printf("day:month:year = %d:%d:%d\n", *t1.day, *t1.month, *t1.year );

   times = &t1;

   *(times->day) = 10;
   printf("day:month:year = %d:%d:%d\n", *t1.day, *t1.month, *t1.year );
}
```

© Copyright B Brown. 1984-2000. All rights reserved. Previous

C Programming

Previous

Practise Exercise 11: Pointers & Structures

To run this test requires a JavaScript enabled browser

---

1. Declare a pointer to a structure of type *date* called *dates*.

- struct dates dates;
- struct *date *dates;
- struct dates date;
- struct date *dates;

---

2. If the above structure of type date comprises three integer fields, day, month, year, assign the value 10 to the field *day* using the *dates* pointer.

- dates.day = 10;
- dates->day = 10;
- dates = 10.day;
- day.dates = 10;

---

3. A structure of type *machine* contains two fields, an integer called *name*, and a char pointer called *memory*. Show what the definition of the structure looks like.

Choice 1

```
struct machine {
        int name;
        char memory;
}
```

Choice 2

```
machine {
        name : integer;
        memory : char^;
};
```

Choice 3

```
struct machine {
```

```
                int name;
                char *memory;
        };
```

---

4. A char pointer called *mpu641* is declared. What is the command to assign the value NULL to the field *memory*.

- mpu641->memory = (char *) NULL;
- mpu641.memory = 0;
- mpu641-memory = 0;
- strcpy(mpu641.memory, NULL);

---

5. Assign the address of the character array *CPUtype* to the field *memory* using the pointer *mpu641*.

- mpu641.memory = &CPUtype;
- mpu641->memory = CPUtype;
- strcpy(mpu641.memory, CPUtype);
- mpu641.memory = CPUtype;

---

6. Assign the value 10 to the field *name* using the pointer *mpu641*.

- mpu641.name = 10;
- mpu641->name = 10;
- mpu641.name = *10;
- *mpu641.name = 10;

---

7. A structure pointer *times* of type *time* (which has three fields, all pointers to integers, day, month and year respectively) is declared. Using the pointer *times*, update the field *day* to 10.

- times.day = 10;
- *(times->day) = 10;
- *times.day = 10;
- times.day = *10;

---

8. An array of pointers (10 elements) of type *time* (as detailed above in 7.), called *sample* is declared. Update the field *month* of the third array element to 12.

- *(sample[2]->month) = 12;
- sample[3].month = 12;
- *sample[2]->month = 12;
- *(sample[3]->month) = 12;

©Copyright B Brown. 1984-2000. All rights reserved.

Previous

C Programming

INDEX Previous Next

Practise Exercise 11a: Pointers & Structures
This program introduces a structure which is passed to a function *editrecord()* as a reference and accessed via a pointer *goods*.

Determine the output of the following program.

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

struct  record {
        char name[20];
        int id;
        float price;
};

void editrecord( struct record * );

void editrecord( struct record *goods )
{
        strcpy( goods->name, "Baked Beans" );
        goods->id = 220;
        (*goods).price = 2.20;
        printf("Name = %s\n", goods->name );
        printf("ID = %d\n", goods->id);
        printf("Price = %.2f\n", goods->price );
}

main()
{
        struct record item;

        strcpy( item.name, "Red Plum Jam");
        editrecord( &item );
        item.price = 2.75;
        printf("Name = %s\n", item.name );
        printf("ID = %d\n", item.id);
        printf("Price = %.2f\n", item.price );
}
```

1. Before call to editrecord()

2. After return from editrecord()

3. The final values of values, item.name, item.id, item.price

Answer

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

Previous

Practise Exercise 11a: Pointers & Structures

Determine the output of the following program.

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

struct  record {
        char name[20];
        int id;
        float price;
};

void editrecord( struct record * );

void editrecord( struct record *goods )
{
        strcpy( goods->name, "Baked Beans" );
        goods->id = 220;
        (*goods).price = 2.20;
        printf("Name = %s\n", goods->name );
        printf("ID = %d\n", goods->id);
        printf("Price = %.2f\n", goods->price );
}

main()
{
        struct record item;

        strcpy( item.name, "Red Plum Jam");
        editrecord( &item );
        item.price = 2.75;
        printf("Name = %s\n", item.name );
        printf("ID = %d\n", item.id);
        printf("Price = %.2f\n", item.price );
}
```

1. Before call to editrecord()

```
        item.name = "Red Plum Jam"
        item.id = 0
        item.price = 0.0
```

2. After return from editrecord()

```
        item.name = "Baked Beans"
        item.id = 220
        item.price = 2.20
```

3. The final values of values, item.name, item.id, item.price

```
        item.name = "Baked Beans"
        item.id = 220
        item.price = 2.75
```

©Copyright B Brown. 1984-2000. All rights reserved.

Previous

C Programming

C25: Examples on Pointer Usage
This program introduces a structure which contains pointers as some of its fields.

Determine the output of the following program.

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

struct  sample {
        char *name;
        int *id;
        float price;
};

static char product[]="Red Plum Jam";

main()
{
        int code = 312, number;
        char name[] = "Baked beans";
        struct sample item;

        item.name = product;
        item.id = &code;
        item.price = 2.75;
        item.name = name;
        number = *item.id;
        printf("Name = %s\n", item.name );
        printf("ID = %d\n", *item.id);
        printf("Price = %.2f\n", item.price );
}
```

Answer

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

Previous

C25: Examples on Pointer Usage
Determine the output of the following program.

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

struct  sample {
        char *name;
        int *id;
        float price;
};

static char product[]="Red Plum Jam";

main()
{
        int code = 312, number;
        char name[] = "Baked beans";
        struct sample item;

        item.name = product;
        item.id = &code;
        item.price = 2.75;
        item.name = name;
        number = *item.id;
        printf("Name = %s\n", item.name );
        printf("ID = %d\n", *item.id);
        printf("Price = %.2f\n", item.price );
}
```

```
Name = Baked Beans
ID = 312
Price = 2.75
```

©Copyright B Brown. 1984-2000. All rights reserved.

Previous

C Programming

INDEX | Previous | Next

C26: Examples on Pointer Usage
This program introduces a structure which has pointers as some of its fields. The structure is passed to a function *printrecord()* as a reference and accessed via a pointer *goods*. This function also updates some of the fields.

Determine the output of the following program.

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

struct  sample {
        char *name;
        int *id;
        float price;
};

static char  product[] = "Greggs Coffee";
static float price1 = 3.20;
static int   id = 773;

void printrecord( struct sample * );

void printrecord( struct sample *goods )
{
        printf("Name = %s\n", goods->name );
        printf("ID = %d\n", *goods->id);
        printf("Price = %.2f\n", goods->price );
        goods->name = &product[0];
        goods->id = &id;
        goods->price = price1;
}

main()
{
        int code = 123, number;
        char name[] = "Apple Pie";
        struct sample item;

        item.id = &code;
        item.price = 1.65;
```

```
        item.name = name;
        number = *item.id;
        printrecord( &item );
        printf("Name = %s\n", item.name );
        printf("ID = %d\n", *item.id);
        printf("Price = %.2f\n", item.price );
}
```

Answer

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

Previous

C26: Examples on Pointer Usage
Determine the output of the following program.

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

struct  sample {
        char *name;
        int *id;
        float price;
};

static char  product[] = "Greggs Coffee";
static float price1 = 3.20;
static int   id = 773;

void printrecord( struct sample * );

void printrecord( struct sample *goods )
{
        printf("Name = %s\n", goods->name );
        printf("ID = %d\n", *goods->id);
        printf("Price = %.2f\n", goods->price );
        goods->name = &product[0];
        goods->id = &id;
        goods->price = price1;
}

main()
{
        int code = 123, number;
        char name[] = "Apple Pie";
        struct sample item;

        item.id = &code;
        item.price = 1.65;
        item.name = name;
        number = *item.id;
        printrecord( &item );
        printf("Name = %s\n", item.name );
```

```
        printf("ID = %d\n", *item.id);
        printf("Price = %.2f\n", item.price );
}
```

What are we trying to print out?

What does it evaluate to?

eg,

```
        printf("ID = %d\n", *goods->id);
        %d is an integer
                we want the value to be a variable integer type
        goods->id,
                what is id, its a pointer, so we mean contents of,
                        therefor we use *goods->id
                which evaluates to an integer type
```

```
Name = Apple Pie
ID = 123
Price = 1.65

Name = Greggs Coffee
ID = 773
Price = 3.20
```

© Copyright B Brown. 1984-2000. All rights reserved. Previous

C Programming

INDEX Previous Next

File Handling Example

```
/* File handling example for PR101     */
/* processing an ASCII file of records */
/* Written by B. Brown, April 1994     */

/* process a goods file, and print out */
/* all goods where the quantity on     */
/* hand is less than or equal to the   */
/* re-order level.                     */

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

/* definition of a record of type goods */
struct goods {
   char  name[20];      /* name of product     */
   float price;         /* price of product    */
   int   quantity;      /* quantity on hand    */
   int   reorder;       /* re-order level      */
};

/* function prototypes */
void myexit( int );
void processfile( void );
void printrecord( struct goods );
int getrecord( struct goods * );

/* global data variables */
FILE *fopen(), *input_file;  /* input file pointer */

/* provides a tidy means to exit program gracefully */
void myexit( int exitcode )
{
   if( input_file != NULL )
      fclose( input_file );
   exit( exitcode );
}

/* prints a record */
```

```
void printrecord( struct goods record )
{
   printf("\nProduct name\t%s\n", record.name );
   printf("Product price\t%.2f\n", record.price );
   printf("Product quantity\t%d\n", record.quantity );
   printf("Product reorder level\t%d\n", record.reorder );
}

/* reads one record from inputfile into 'record', returns 1 for success */
int getrecord( struct goods *record )
{
   int loop = 0, ch;
   char buffer[40];

   ch = fgetc( input_file );
   /* skip to start of record */
   while( (ch == '\n') || (ch == ' ') && (ch != EOF) )
      ch = fgetc( input_file );
   if( ch == EOF ) return 0;

   /* read product name */
   while( (ch != '\n') && (ch != EOF)) {
      buffer[loop++] = ch;
      ch = fgetc( input_file );
   }
   buffer[loop] = 0;
   strcpy( record->name, buffer );
   if( ch == EOF ) return 0;

   /* skip to start of next field */
   while( (ch == '\n') || (ch == ' ') && (ch != EOF) )
      ch = fgetc( input_file );
   if( ch == EOF ) return 0;

   /* read product price */
   loop = 0;
   while( (ch != '\n') && (ch != EOF)) {
      buffer[loop++] = ch;
      ch = fgetc( input_file );
   }
   buffer[loop] = 0;
   record->price = atof( buffer );
   if( ch == EOF ) return 0;

   /* skip to start of next field */
   while( (ch == '\n') || (ch == ' ') && (ch != EOF) )
      ch = fgetc( input_file );
   if( ch == EOF ) return 0;
```

```
   /* read product quantity */
   loop = 0;
   while( (ch != '\n') && (ch != EOF)) {
      buffer[loop++] = ch;
      ch = fgetc( input_file );
   }
   buffer[loop] = 0;
   record->quantity = atoi( buffer );
   if( ch == EOF ) return 0;

   /* skip to start of next field */
   while( (ch == '\n') || (ch == ' ') && (ch != EOF) )
      ch = fgetc( input_file );
   if( ch == EOF ) return 0;

   /* read product reorder level */
   loop = 0;
   while( (ch != '\n') && (ch != EOF)) {
      buffer[loop++] = ch;
      ch = fgetc( input_file );
   }
   buffer[loop] = 0;
   record->reorder = atoi( buffer );
   if( ch == EOF ) return 0;

   return 1;  /* signify record has been read successfully */
}

/* processes file for records */
void processfile( void )
{
   struct goods record;   /* holds a record read from inputfile */

   while( ! feof( input_file )) {
      if( getrecord( &record ) == 1 ) {
         if( record.quantity <= record.reorder )
            printrecord( record );
      }
      else myexit( 1 );  /* error getting record */
   }
}

main()
{
   char filename[40];     /* name of database file */

   printf("Example Goods Re-Order File Program\n");
```

```
   printf("Enter database file ");
   scanf(" %s", filename );
   input_file = fopen( filename, "rt" );
   if( input_file == NULL ) {
      printf("Unable to open datafile %s\n", filename );
      myexit( 1 );
   }
   processfile();
   myexit( 0 );
}
```

Please obtain the data file for this example from your tutor, or via **ftp**.

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

Previous

File Handling Example

The data file for this exercise looks like,

```
baked beans
1.20
10
5

greggs coffee
2.76
5
10

walls ice-cream
3.47
5
5

cadburys chocs
4.58
12
10
```

©Copyright B Brown. 1984-2000. All rights reserved.

Previous

C Programming

INDEX Previous Next

LINKED LISTS

A linked list is a complex data structure, especially useful in systems or applications programming. A linked list is comprised of a series of nodes, each node containing a data element, and a pointer to the next node, eg,

```
         --------            --------
        |  data  |      --->|  data  |
        |--------|     |    |--------|
        | pointer|----      | pointer| ---> NULL
         --------            --------
```

A structure which contains a data element and a pointer to the next node is created by,

```
        struct list {
                int   value;
                struct list  *next;
        };
```

This defines a new data structure called *list* (actually the definition of a node), which contains two members. The first is an integer called *value*. The second is called *next*, which is a pointer to another list structure (or node). Suppose that we declare two structures to be of the same type as list, eg,

```
        struct list  n1, n2;
```

The next pointer of structure *n1* may be set to point to the *n2* structure by

```
        /* assign address of first element in n2 to the pointer next of the n1 
structure  */
        n1.next = &n2;
```

which creates a link between the two structures.

```
        /* LLIST.C    Program to illustrate linked lists */
        #include <stdio.h>
        struct list {
                int         value;
                struct list *next;
        };

        main()
        {
                struct list n1, n2, n3;
                int          i;
```

```
        n1.value = 100;
        n2.value = 200;
        n3.value = 300;
        n1.next = &n2;
        n2.next = &n3;
        i = n1.next->value;
        printf("%d\n", n2.next->value);
}
```

Not only this, but consider the following

```
n1.next = n2.next;       /* removes n2 from the list */
n2_3.next = n2.next;     /* adds struct n2_3 */
n2.next = &n2_3;
```

In using linked list structures, it is common to assign the value of 0 to the last pointer in the list, to indicate that there are no more nodes in the list, eg,

```
n3.next = 0;
```

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

Traversing a linked list

```
/* Program to illustrate traversing a list */
#include <stdio.h>
struct list {
        int          value;
        struct list *next;
};

main()
{
        struct list n1, n2, n3, n4;
        struct list *list_pointer = &n1;

        n1.value = 100;
        n1.next = &n2;
        n2.value = 200;
        n2.next = &n3;
        n3.value = 300;
        n3.next = &n4;
        n4.value = 400;
        n4.next = 0;


        while( list_pointer != 0 )  {
                printf("%d\n", list_pointer->value);
                list_pointer = list_pointer->next;
        }
}
```

This program uses a pointer called *list_pointer* to cycle through the linked list.

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

Practise Exercise 12: Lists

1. Define a structure called *node*, which contains an integer element called *data*, and a pointer to a structure of type *node* called *next_node*.

2. Declare three structures called *node1*, *node2*, *node3*, of type *node*.

3. Write C statements which will link the three nodes together, with node1 at the head of the list, node2 second, and node3 at the tail of the list. Assign the value NULL to node3.next to signify the end of the list.

4. Using a pointer *list*, of type *node*, which has been initialised to the address of *node1*, write C statements which will cycle through the list and print out the value of each nodes data field.

5. Assuming that pointer *list* points to *node2*, what does the following statement do?

```
list->next_node = (struct node *) NULL;
```

6. Assuming the state of the list is that as in 3., write C statements which will insert a new node *node1a* between node1 and node2, using the pointer *list* (which is currently pointing to node1). Assume that a pointer *new_node* points to node node1a.

7. Write a function called *delete_node*, which accepts a pointer to a list, and a pointer to the node to be deleted from the list, eg

```
void delete_node( struct node *head, struct node *delnode );
```

8. Write a function called *insert_node*, which accepts a pointer to a list, a pointer to a new node to be inserted, and a pointer to the node after which the insertion takes place, eg

```
void insert_node( struct node *head, struct node *newnode, struct node *prevnode );
```

Answers

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

Previous

Practise Exercise 12: Lists

1. Define a structure called *node*, which contains an integer element called *data*, and a pointer to a structure of type *node* called *next_node*.

```
	struct node {
		int data;
		struct node *next_node;
	};
```

2. Declare three structures called *node1*, *node2*, *node3*, of type *node*.

```
	struct node node1, node3, node3;
```

3. Write C statements which will link the three nodes together, with node1 at the head of the list, node2 second, and node3 at the tail of the list. Assign the value NULL to node3.next to signify the end of the list.

```
	node1.next_node = &node2;
	node2.next_node = &node3;
	node3.next_node = (struct node *) NULL;
```

4. Using a pointer *list*, of type *node*, which has been initialised to the address of *node1*, write C statements which will cycle through the list and print out the value of each nodes data field.

```
	while( list != NULL ) {
		printf("%d\n", list->data );
		list = list->next_node;
	}
```

5. Assuming that pointer *list* points to *node2*, what does the following statement do?

```
	list->next_node = (struct node *) NULL;

	The statement writes a NULL into the next_node pointer, making node2 the end 
of
	the list, thereby erasing node3 from the list.
```

6. Assuming the state of the list is that as in 3., write C statements which will insert a new node *node1a* between node1 and node2, using the pointer *list* (which is currently pointing to node1). Assume that a pointer *new_node* points to node node1a.

```
	new_node.next_node = list.next_node;
	list.next_node = new_node;
```

7. Write a function called *delete_node*, which accepts a pointer to a list, and a pointer to the node to be deleted from the list,

eg

```
        void  delete_node(  struct  node  *head,  struct  node  *delnode );


        void  delete_node( struct node *head, struct node *delnode )
        {
                struct node *list;

                list = head;
                while( list->next != delnode )
                        list = list->next;

                if( list ) {    /* will be valid if the node was found */
                        list->next = delnode->next;     /* bypass it */
                        free(delnode);  /* deallocate node space */
                }
                else {
                        /* node to delete was not found */
                }
        }
```

8. Write a function called *insert_node*, which accepts a pointer to a list, a pointer to a new node to be inserted, and a pointer to the node after which the insertion takes place, eg

```
        void insert_node( struct node *head, struct node *newnode, struct node
*prevnode );


        void insert_node( struct node *head, struct node *newnode, struct node
*prevnode )
        {
                struct node *list;

                list = head;
                while( list != prevnode )
                        list = list->next;

                newnode->next = list->next;
                list->next = newnode;
        }
```

©Copyright B Brown. 1984-2000. All rights reserved.

Previous

C Programming

INDEX Previous Next

DYNAMIC MEMORY ALLOCATION (CALLOC, SIZEOF, FREE)
It is desirable to dynamically allocate space for variables at runtime. It is wasteful when dealing with array type structures to allocate so much space when declared, eg,

```
struct client clients[100];
```

This practice may lead to memory contention or programs crashing. A far better way is to allocate space to clients when needed.

The C programming language allows users to dynamically allocate and deallocate memory when required. The functions that accomplish this are *calloc()*, which allocates memory to a variable, *sizeof*, which determines how much memory a specified variable occupies, and *free()*, which deallocates the memory assigned to a variable back to the system.

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

SIZEOF

The sizeof() function returns the memory size of the requested variable. This call should be used in conjunction with the *calloc()* function call, so that only the necessary memory is allocated, rather than a fixed size. Consider the following,

```
struct date {
        int hour, minute, second;
};

int x;

x = sizeof( struct date );
```

x now contains the information required by *calloc()* so that it can allocate enough memory to contain another structure of type *date*.

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

CALLOC

This function is used to allocate storage to a variable whilst the program is running. The function takes two arguments that specify the number of elements to be reserved, and the size of each element (obtained from *sizeof*) in bytes. The function returns a character pointer (void in ANSI C) to the allocated storage, which is initialized to zero's.

```
struct date *date_pointer;

date_pointer = (struct date *)  calloc( 10, sizeof(struct date) );
```

The (struct date *) is a type cast operator which converts the pointer returned from *calloc* to a character pointer to a structure of type *date*. The above function call will allocate size for ten such structures, and *date_pointer* will point to the first in the chain.

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

INDEX Previous Next

FREE
When the variables are no longer required, the space which was allocated to them by *calloc* should be returned to the system. This is done by,

```
free( date_pointer );
```

Other C calls associated with memory are,

```
alloc           allocate a block of memory from the heap
malloc          allocate a block of memory, do not zero out
zero            zero a section of memory
blockmove       move bytes from one location to another
```

Other routines may be included in the particular version of the compiler you may have, ie, for MS-DOS v3.0,

```
memccpy         copies characters from one buffer to another
memchr          returns a pointer to the 1st occurrence of a
                designated character searched for
memcmp          compares a specified number of characters
memcpy          copies a specified number of characters
memset          initialise a specified number of bytes with a given character
movedata        copies characters
```

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

EXAMPLE OF DYNAMIC ALLOCATION

```
/* linked list example, pr101, 1994 */
#include <string.h>
#include <alloc.h>
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <conio.h>

/* definition of a node */
struct node {
    char data[20];
    struct node *next;
};


struct node * initialise( void );
void freenodes( struct node * );
int insert( struct node * );
void delete( struct node *, struct node * );
void list( struct node * );
void menu( struct node *, struct node * );
void readline( char [] );



void readline( char buff[] )
{
    int ch, loop = 0;

    ch = getche();
    while( ch != '\r' ) {
       buff[loop] = ch;
       loop++;
       ch = getche();
    }
    buff[loop] = 0;
}
```

```
struct node * initialise( void )
{
    return( (struct node *) calloc(1, sizeof( struct node *) ));
}

/* free memory allocated for node */
void freenodes( struct node *headptr )
{
    struct node *temp;
    while( headptr ) {
       temp = headptr->next;
       free( headptr );
       headptr = temp;
    }
}

/* insert a new node after nodeptr, return 1 = success */
int insert( struct node *nodeptr )
{
    char buffer[20];
    struct node *newptr;

    newptr = initialise(); /* allocate a new node */
    if( newptr == NULL ) {
        return 0;
    }
    else {                /* fill in its data and add to the list */
        newptr->next = nodeptr->next;
        nodeptr->next = newptr;
        nodeptr = newptr;
        printf("\nEnter data --->");
        readline( buffer );
        strcpy( nodeptr->data, buffer );
    }
    return 1;
}

/* delete a node from list */
void delete( struct node *headptr, struct node *nodeptr )
{
    struct node *deletepointer, *previouspointer;
    char buffer[20];

    deletepointer = headptr->next;
    previouspointer = headptr;
```

```
    /* find the entry */
    printf("\nEnter name to be deleted --->");
    readline( buffer );
    while( deletepointer ) {
        if( strcmp( buffer, deletepointer->data ) == 0 ) {
            /* delete node pointed to by delete pointer */
            previouspointer->next = deletepointer->next;
            break;
        }
        else {
            /* goto next node in list */
            deletepointer = deletepointer->next;
            previouspointer = previouspointer->next;
        }
    }
    /* did we find it? */
    if( deletepointer == NULL )
        printf("\n\007Error, %s not found or list empty\n", buffer);
    else {
        free( deletepointer );
        /* adjust nodeptr to the last node in list */
        nodeptr = headptr;
        while( nodeptr->next != NULL )
            nodeptr = nodeptr->next;
    }
}

/* print out the list */
void list( struct node *headptr )
{
    struct node *listpointer;

    listpointer = headptr->next;
    if( listpointer == NULL )
        printf("\nThe list is empty.\n");
    else {
        while( listpointer ) {
            printf("Name : %20s\n", listpointer->data );
            listpointer = listpointer->next;
        }
    }
}

/* main menu system */
void menu( struct node *headp, struct node *nodep )
{
```

```
    int menuchoice = 1;
    char buffer[20];

    while( menuchoice != 4 ) {
       printf("1  insert a node\n");
       printf("2  delete a node\n");
       printf("3  list nodes\n");
       printf("4  quit\n");
       printf("Enter choice -->");
       readline( buffer );
       menuchoice = atoi( buffer );
       switch( menuchoice ) {
           case 1 : if( insert( nodep ) == 0 )
                          printf("\n\007Insert failed.\n");
                       break;
           case 2 : delete( headp, nodep );  break;
           case 3 : list( headp );    break;
           case 4 : break;
           default : printf("\n\007Invalid option\n"); break;
       }
    }
}

main()
{
    struct node *headptr, *nodeptr;
    headptr = initialise();
    nodeptr = headptr;
    headptr->next = NULL;
    menu( headptr, nodeptr );
    freenodes( headptr );
}
```

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

INDEX Previous Next

Another Linked List Example

```
/* linked list example */
#include <stdio.h>
#include <alloc.h>
#include <stdlib.h>
#include <conio.h>
#include <ctype.h>
#include <string.h>

/* function prototypes */
struct node * initnode( char *, int );
void printnode( struct node * );
void printlist( struct node * );
void add( struct node * );
struct node * searchname( struct node *, char * );
void deletenode( struct node * );
void insertnode( struct node * );
void deletelist( struct node * );

/* definition of a data node for holding student information */
struct node {
   char name[20];
   int  id;
   struct node *next;
};

/* head points to first node in list, end points to last node in list */
/* initialise both to NULL, meaning no nodes in list yet */
struct node *head = (struct node *) NULL;
struct node *end = (struct node *) NULL;

/* this initialises a node, allocates memory for the node, and returns   */
/* a pointer to the new node. Must pass it the node details, name and id */
struct node * initnode( char *name, int id )
{
   struct node *ptr;
   ptr = (struct node *) calloc( 1, sizeof(struct node ) );
   if( ptr == NULL )                       /* error allocating node?      */
       return (struct node *) NULL;        /* then return NULL, else      */
   else {                                  /* allocated node successfully */
       strcpy( ptr->name, name );          /* fill in name details        */
       ptr->id = id;                       /* copy id details             */
```

```
        return ptr;                          /* return pointer to new node  */
    }
}

/* this prints the details of a node, eg, the name and id                   */
/* must pass it the address of the node you want to print out               */
void printnode( struct node *ptr )
{
   printf("Name ->%s\n", ptr->name );
   printf("ID   ->%d\n", ptr->id );
}

/* this prints all nodes from the current address passed to it. If you    */
/* pass it 'head', then it prints out the entire list, by cycling through */
/* each node and calling 'printnode' to print each node found             */
void printlist( struct node *ptr )
{
   while( ptr != NULL )           /* continue whilst there are nodes left */
   {
      printnode( ptr );           /* print out the current node           */
      ptr = ptr->next;            /* goto the next node in the list       */
   }
}

/* this adds a node to the end of the list. You must allocate a node and  */
/* then pass its address to this function                                 */
void add( struct node *new )  /* adding to end of list */
{
   if( head == NULL )      /* if there are no nodes in list, then         */
       head = new;         /* set head to this new node                   */
   end->next = new;        /* link in the new node to the end of the list */
   new->next = NULL;       /* set next field to signify the end of list   */
   end = new;              /* adjust end to point to the last node        */
}

/* search the list for a name, and return a pointer to the found node     */
/* accepts a name to search for, and a pointer from which to start. If    */
/* you pass the pointer as 'head', it searches from the start of the list */
struct node * searchname( struct node *ptr, char *name )
{
    while( strcmp( name, ptr->name ) != 0 ) {    /* whilst name not found */
       ptr = ptr->next;                          /* goto the next node    */
       if( ptr == NULL )                         /* stop if we are at the */
          break;                                 /* of the list           */
    }
    return ptr;                                  /* return a pointer to   */
}                                                /* found node or NULL    */

/* deletes the specified node pointed to by 'ptr' from the list           */
```

```
void deletenode( struct node *ptr )
{
   struct node *temp, *prev;
   temp = ptr;    /* node to be deleted */
   prev = head;   /* start of the list, will cycle to node before temp    */

   if( temp == prev ) {                        /* are we deleting first node  */
       head = head->next;                      /* moves head to next node     */
       if( end == temp )                       /* is it end, only one node?   */
          end = end->next;                     /* adjust end as well          */
       free( temp );                           /* free space occupied by node */
   }
   else {                                      /* if not the first node, then */
       while( prev->next != temp ) {           /* move prev to the node before*/
           prev = prev->next;                  /* the one to be deleted       */
       }
       prev->next = temp->next;                /* link previous node to next  */
       if( end == temp )                       /* if this was the end node,   */
           end = prev;                         /* then reset the end pointer  */
       free( temp );                           /* free space occupied by node */
   }
}

/* inserts a new node, uses name field to align node as alphabetical list */
/* pass it the address of the new node to be inserted, with details all   */
/* filled in                                                              */
void insertnode( struct node *new )
{
   struct node *temp, *prev;                    /* similar to deletenode      */

   if( head == NULL ) {                         /* if an empty list,          */
       head = new;                              /* set 'head' to it           */
       end = new;
       head->next = NULL;                       /* set end of list to NULL    */
       return;                                  /* and finish                 */
   }

   temp = head;                                 /* start at beginning of list */
                     /* whilst currentname < newname to be inserted then */
   while( strcmp( temp->name, new->name) < 0 ) {
          temp = temp->next;                    /* goto the next node in list */
          if( temp == NULL )                    /* dont go past end of list   */
              break;
   }

   /* we are the point to insert, we need previous node before we insert  */
   /* first check to see if its inserting before the first node!          */
   if( temp == head ) {
      new->next = head;             /* link next field to original list   */
```

```
      head = new;                     /* head adjusted to new node         */
   }
   else {     /* okay, so its not the first node, a different approach    */
      prev = head;   /* start of the list, will cycle to node before temp */
      while( prev->next != temp ) {
          prev = prev->next;
      }
      prev->next = new;               /* insert node between prev and next  */
      new->next = temp;
      if( end == prev )               /* if the new node is inserted at the */
         end = new;                   /* end of the list the adjust 'end'   */
   }
}

/* this deletes all nodes from the place specified by ptr                 */
/* if you pass it head, it will free up entire list                       */
void deletelist( struct node *ptr )
{
   struct node *temp;

   if( head == NULL ) return;   /* dont try to delete an empty list       */

   if( ptr == head ) {      /* if we are deleting the entire list         */
       head = NULL;         /* then reset head and end to signify empty   */
       end = NULL;          /* list                                       */
   }
   else {
       temp = head;          /* if its not the entire list, readjust end  */
       while( temp->next != ptr )          /* locate previous node to ptr  */
           temp = temp->next;
       end = temp;                         /* set end to node before ptr   */
   }

   while( ptr != NULL ) {   /* whilst there are still nodes to delete     */
      temp = ptr->next;     /* record address of next node                */
      free( ptr );          /* free this node                             */
      ptr = temp;           /* point to next node to be deleted           */
   }
}

/* this is the main routine where all the glue logic fits                 */
main()
{
   char name[20];
   int id, ch = 1;
   struct node *ptr;

   clrscr();
   while( ch != 0 ) {
```

```
    printf("1 add a name \n");
    printf("2 delete a name \n");
    printf("3 list all names \n");
    printf("4 search for name \n");
    printf("5 insert a name \n");
    printf("0 quit\n");
    scanf("%d", &ch );
    switch( ch )
    {
        case 1:  /* add a name to end of list */
                 printf("Enter in name -- ");
                 scanf("%s", name );
                 printf("Enter in id -- ");
                 scanf("%d", &id );
                 ptr = initnode( name, id );
                 add( ptr );
                 break;
        case 2:  /* delete a name */
                 printf("Enter in name -- ");
                 scanf("%s", name );
                 ptr = searchname( head, name );
                 if( ptr ==NULL ) {
                     printf("Name %s not found\n", name );
                 }
                 else
                    deletenode( ptr );
                 break;

        case 3:  /* list all nodes */
                 printlist( head );
                 break;

        case 4:  /* search and print name */
                 printf("Enter in name -- ");
                 scanf("%s", name );
                 ptr = searchname( head, name );
                 if( ptr ==NULL ) {
                     printf("Name %s not found\n", name );
                 }
                 else
                    printnode( ptr );
                 break;
        case 5:  /* insert a name in list */
                 printf("Enter in name -- ");
                 scanf("%s", name );
                 printf("Enter in id -- ");
                 scanf("%d", &id );
                 ptr = initnode( name, id );
                 insertnode( ptr );
```

```
                        break;

        }
    }
    deletelist( head );
}
```

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

INDEX Previous Next

PREPROCESSOR STATEMENTS
The *define* statement is used to make programs more readable, and allow the inclusion of macros. Consider the following examples,

```
        #define TRUE    1    /* Do not use a semi-colon , # must be first character
on line */
        #define FALSE   0
        #define NULL    0
        #define AND     &
        #define OR      |
        #define EQUALS  ==

                        game_over = TRUE;
                        while( list_pointer != NULL )
                                ................
```

Macros

Macros are inline code which are substituted at compile time. The definition of a macro, which accepts an argument when referenced,

```
        #define  SQUARE(x)  (x)*(x)

        y = SQUARE(v);
```

In this case, *v* is equated with *x* in the macro definition of *square*, so the variable *y* is assigned the square of *v*. The brackets in the macro definition of *square* are necessary for correct evaluation. The expansion of the macro becomes

```
        y = (v) * (v);
```

Naturally, macro definitions can also contain other macro definitions,

```
        #define IS_LOWERCASE(x)  (( (x)>='a') && ( (x) <='z') )
        #define TO_UPPERCASE(x)  (IS_LOWERCASE (x)?(x)-'a'+'A':(x))

        while(*string) {
                *string = TO_UPPERCASE(*string);
                ++string;
        }
```

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

CONDITIONAL COMPILATIONS
These are used to direct the compiler to compile/or not compile the lines that follow

```
#ifdef  NULL
#define NL 10
#define SP 32
#endif
```

In the preceding case, the definition of NL and SP will only occur if NULL has been defined prior to the compiler encountering the #ifdef NULL statement. The scope of a definition may be limited by

```
#undef NULL
```

This renders the identification of NULL invalid from that point onwards in the source file.

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

typedef

This statement is used to classify existing C data types, eg,

```
typedef  int counter;  /* redefines counter as an integer */
counter j, n;          /* counter now used to define j and n as integers */

typedef struct {
        int month, day, year;
} DATE;

DATE  todays_date;      /* same as struct date todays_date */
```

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

INDEX Previous Next

ENUMERATED DATA TYPES
Enumerated data type variables can only assume values which have been previously declared.

```
        enum month { jan = 1, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec
};
        enum month this_month;

        this_month = feb;
```

In the above declaration, *month* is declared as an enumerated data type. It consists of a set of values, jan to dec. Numerically, jan is given the value 1, feb the value 2, and so on. The variable *this_month* is declared to be of the same type as month, then is assigned the value associated with feb. This_month cannot be assigned any values outside those specified in the initialization list for the declaration of month.

```
        #include <stdio.h>

        main()
        {
                char *pwest = "west",*pnorth = "north", *peast="east", *psouth =
"south";
                enum location { east=1, west=2, south=3, north=4};
                enum location direction;

                direction = east;

                if( direction == east )
                        printf("Cannot go %s\n", peast);
        }
```

The variables defined in the enumerated variable *location* should be assigned initial values.

©Copyright B Brown. 1984-2000. All rights reserved.
INDEX Previous Next

C Programming

INDEX | Previous | Next

UNIONS

This is a special data type which looks similar to a structure, but is very different. The declaration is,

```
union  mixed {
        char   letter;
        float  radian;
        int    number;
};

union mixed all;
```

The first declaration consists of a union of type mixed, which consists of a char, float, or int variable. NOTE that it can be ONLY ONE of the variable types, they cannot coexist.

This is due to the provision of a single memory address which is used to store the largest variable, unlike the arrangement used for structures.

Thus the variable *all* can only be a character, a float or an integer at any one time. The C language keeps track of what *all* actually is at any given moment, but does not provide a check to prevent the programmer accessing it incorrectly.

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX | Previous | Next

C Programming

DECLARING VARIABLES TO BE REGISTER BASED
Some routines may be time or space critical. Variables can be defined as being register based by the following declaration,

```
register int index;
```

DECLARING VARIABLES TO BE EXTERNAL
Here variables may exist in separately compiled modules, and to declare that the variable is external,

```
extern  int  move_number;
```

This means that the data storage for the variable *move_number* resides in another source module, which will be linked with this module to form an executable program. In using a variable across a number of independently compiled modules, space should be allocated in only one module, whilst all other modules use the extern directive to access the variable.

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

NULL OR EMPTY STATEMENTS
These are statements which do not have any body associated with them.

```
        /* sums all integers in array a containing n elements and initializes */
        /* two variables at the start of the for loop */
        for( sum = 0, i = 0; i < n; sum += a[i++] )
                ;
```

```
/* Copies characters from standard input to standard output until EOF is reached  */
        for( ; (c = getchar ()) != EOF; putchar (c))
                ;
```

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

INDEX Previous Next

STRINGS
Consider the following,

```
char  *text_pointer = "Hello said the man.";
```

This defines a character pointer called *text_pointer* which points to the start of the text string 'Hello said the man'. This message could be printed out by

```
printf("%s", text_pointer);
```

text_pointer holds the memory address of where the message is located in memory.

Lets append two strings together by using arrays.

```
#include <stdio.h>

main()
{
        static char string1[]={'H','e','l','l','o',' ' };
        static char string2[]={'s','a','i','d',' ','t','h','e',' ','m','a','n','.' };
        char  string3[25];
        int string_length1 = 6, string_length2 = 13, n;

        for( n = 0; n < string_length1; ++n )
                string3[n] = string1[n];

        for( n = 0; n < string_length2; ++n )
                string3[n + string_length1] = string2[n];

        for(n = 0; n < (stringlength1+string_length2); ++n)
                printf("%c", string3[n]);
}
```

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

Strings continued

There are times that the length of a string may not be known. Consider the following improvements by terminating each string with a null character.

```
#include <stdio.h>

main()
{
        static char  string1[] = "Bye Bye ";
        static char  string2[] = "love.";
        char  string3[25];
        int  n = 0, n2;

        for( ; string1[n] != '\0'; ++n )
                string3[n] = string1[n];

        n2 = n;  n = 0;

        for( ; string2[n] != '\0'; ++n )
                string3[n2 + n] = string2[n];

        n2 += n;

        for(n = 0; n < n2 ; ++n)
                printf("%c", string3[n]);
}
```

Minor modification to above program is,

```
string3[n2 + n] = '\0';
printf("%s", string3);
```

© Copyright B Brown. 1984-2000. All rights reserved.

C Programming

INDEX | Previous | Next

FURTHER IMPROVEMENTS by using POINTERS

The previous program still required the use of variables to keep track of string lengths. Implementing concatenation by the use of pointers eliminates this, eg,

```
#include <stdio.h>

void concat( char *, char *, char * );

/* this functions copies the strings a and b to the destination string c */
void concat( char *a, char *b, char *c)
{
        while( *a )  {           /* while( *c++ = *a++ );  */
                *c = *a; ++a; ++c;
        }
        while( *b )  {
                *c = *b; ++b; ++c;
        }
        *c = '\0';
}

main()
{
        static char string1[] = "Bye Bye ";
        static char string2[] = "love.";
        char string3[20];

        concat( string1, string2, string3);
        printf("%s\n", string3);
}
```

USING strcat IN THE LIBRARY ROUTINE string.h

The following program illustrates using the supplied function resident in the appropriate library file. *strcat()* concatenates one string onto another and returns a pointer to the concatenated string.

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

main()
{
        static char string1[] = "Bye Bye ";
        static char string2[] = "love.";
        char *string3;

        string3 = strcat ( string1, string2 );
        printf("%s\n", string3);
}
```

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

COMMAND LINE ARGUMENTS

It is possible to pass arguments to C programs when they are executed. The brackets which follow main are used for this purpose. *argc* refers to the number of arguments passed, and *argv[]* is a pointer array which points to each argument which is passed to main. A simple example follows, which checks to see if a single argument is supplied on the command line when the program is invoked.

```
#include <stdio.h>

main( int argc, char *argv[] )
{
        if( argc == 2 )
                printf("The argument supplied is %s\n", argv[1]);
        else if( argc > 2 )
                printf("Too many arguments supplied.\n");
        else
                printf("One argument expected.\n");
}
```

Note that **argv[0]* is the name of the program invoked, which means that **argv[1]* is a pointer to the first argument supplied, and **argv[n]* is the last argument. If no arguments are supplied, *argc* will be one. Thus for n arguments, *argc* will be equal to n + 1. The program is called by the command line,

```
 myprog  argument1
```

EXERCISE C27

Rewrite the program which copies files, ie, FCOPY.C to accept the source and destination filenames from the command line. Include a check on the number of arguments passed.

Answer

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

Previous

EXERCISE C27
Rewrite the program which copies files, ie, FCOPY.C to accept the source and destination filenames from the command line. Include a check on the number of arguments passed.

```
        #include <stdio.h>

        main( int argc, char *argv[])
        {
                FILE *in_file, *out_file, *fopen();
                int c;

                if( argc != 3 )
                {
                        printf("Incorrect, format is FCOPY source dest\n");
                        exit(2);
                }
                in_file = fopen( argv[1], "r");
                if( in_file == NULL )  printf("Cannot open %s for reading\n", 
argv[1]);
                else {
                        out_file = fopen( argv[2], "w");
                        if ( out_file == NULL )
                                printf("Cannot open %s for writing\n", argv[2]);
                        else {
                                printf("File copy program, copying %s to %s\n", 
argv[1],  argv[2]);
                                while ( (c=getc( in_file) ) != EOF )
                                        putc( c, out_file );
                                putc( c, out_file);                  /* copy EOF */
                                printf("File has been copied.\n");
                                fclose( out_file);
                        }
                        fclose( in_file);
                }
        }
```

©Copyright B Brown. 1984-2000. All rights reserved.

Previous

C Programming

POINTERS TO FUNCTIONS

A pointer can also be declared as pointing to a function. The declaration of such a pointer is done by,

```
int  (*func_pointer)();
```

The parentheses around **func_pointer* are necessary, else the compiler will treat the declaration as a declaration of a function. To assign the address of a function to the pointer, the statement,

```
func_pointer = lookup;
```

where *lookup* is the function name, is sufficient. In the case where no arguments are passed to *lookup*, the call is

```
(*func_pointer)();
```

The parentheses are needed to avoid an error. If the function *lookup* returned a value, the function call then becomes,

```
i = (*func_pointer)();
```

If the function accepted arguments, the call then becomes,

```
i = (*func_pointer)( argument1, argument2, argumentn);
```

©Copyright B Brown. 1984-2000. All rights reserved.

C Programming

INDEX Previous Next

SAMPLE CODE FOR POINTERS TO FUNCTIONS

Pointers to functions allow the creation of jump tables and dynamic routine selection. A pointer is assigned the start address of a function, thus, by typing the pointer name, program execution jumps to the routine pointed to.

By using a single pointer, many different routines could be executed, simply by re-directing the pointer to point to another function. Thus, programs could use this to send information to a printer, console device, tape unit etc, simply by pointing the pointer associated with output to the appropriate output function!

The following program illustrates the use of pointers to functions, in creating a simple shell program which can be used to specify the screen mode on a CGA system.

```
#include <stdio.h>      /* Funcptr.c */
#include <dos.h>

#define dim(x) (sizeof(x) / sizeof(x[0]) )
#define GETMODE        15
#define SETMODE         0
#define VIDCALL         0X10
#define SCREEN40        1
#define SCREEN80        3
#define SCREEN320       4
#define SCREEN640       6
#define VID_BIOS_CALL(x)  int86( VIDCALL, &x, &x )

int cls(), scr40(), scr80(), scr320(), scr640(), help(), shellquit();
union REGS regs;

struct command_table
{
  char *cmd_name;
  int (*cmd_ptr) ();
}
cmds[]={"40",scr40,"80",scr80,"320",scr320,"640",scr640,"HELP",help,"CLS",cls,"EXIT",\
               shellquit};

cls()
{
  regs.h.ah = GETMODE;     VID_BIOS_CALL( regs );
  regs.h.ah = SETMODE;     VID_BIOS_CALL( regs );
}

scr40()
{
  regs.h.ah = SETMODE;
  regs.h.al = SCREEN40;
  VID_BIOS_CALL( regs );
}

scr80()
{
```

```
  regs.h.ah = SETMODE;
  regs.h.al = SCREEN80;
  VID_BIOS_CALL( regs );
}

scr320()
{
  regs.h.ah = SETMODE;
  regs.h.al = SCREEN320;
  VID_BIOS_CALL( regs );
}

scr640()
{
  regs.h.ah = SETMODE;
  regs.h.al = SCREEN640;
  VID_BIOS_CALL( regs );
}

shellquit()
{
   exit( 0 );
}

help()
{
   cls();
   printf("The available commands are; \n");
   printf("    40    Sets 40 column mode\n");
   printf("    80    Sets 80 column mode\n");
   printf("   320    Sets medium res graphics mode\n");
   printf("   640    Sets high res graphics mode\n");
   printf("   CLS    Clears the display screen\n");
   printf("  HELP    These messages\n");
   printf("  EXIT    Return to DOS\n");
}

get_command( buffer )
char *buffer;
{
  printf("\nShell: ");
  gets( buffer );
  strupr( buffer );
}

execute_command( cmd_string )
char *cmd_string;
{
  int i, j;
  for( i = 0; i < dim( cmds); i++ )
  {
    j = strcmp( cmds[i].cmd_name, cmd_string );
    if( j == 0 )
    {
      (*cmds[i].cmd_ptr) ();
      return 1;
```

```
    }
  }
  return 0;
}

main()
{
  char input_buffer[81];
  while( 1 )
  {
    get_command( input_buffer );
    if( execute_command( input_buffer ) == 0 )
      help();
  }
}
```

©Copyright B Brown. 1984-2000. All rights reserved.

INDEX Previous Next

C Programming

FORMATTERS FOR STRINGS/CHARACTERS
Consider the following program.

```
        #include <stdio.h>

        main()    /* FORMATS.C   */
        {
                char          c = '#';
                static char s[] = "helloandwelcometoclanguage";

                printf("Characters:\n");
                printf("%c\n", c);
                printf("%3c%3c\n", c, c);
                printf("%-3c%-3c\n", c, c);
                printf("Strings:\n");
                printf("%s\n", s);
                printf("%.5s\n", s);
                printf("%30s\n", s);
                printf("%20.5s\n", s);
                printf("%-20.5s\n", s);
        }
```

The output of the above program will be,

```
Characters:
#
  #  #
#  #
Strings:
helloandwelcometoclanguage
hello
    helloandwelcometoclanguage
               hello
hello
```

The statement printf("%.5s\n",s) means print the first five characters of the array s. The statement printf("%30s\n", s) means that the array s is printed right justified, with leading spaces, to a field width of thirty characters.

The statement printf("%20.5s\n", s) means that the first five characters are printed in a field size of twenty which is right justified and filled with leading spaces.

The final printf statement uses a left justified field of twenty characters, trailing spaces, and the .5 indicating to print the first five characters of the array s.

©Copyright B Brown. 1984-2000. All rights reserved.

Advanced C: Part 1 of 3

Copyright Brian Brown, 1986-2000. All rights reserved.

INDEX Next

Comprehensive listing of interrupts and hardware details.

CONTENTS OF PART ONE

- Rom Bios Calls
- Video Bios Calls
- Input/Output Port Access
- Extended Video Bios Calls
- Making Libraries
- Pointers to Functions

ROM BIOS CALLS

The ROM BIOS (Basic Input Output System) provides device control for the PC's major devices (disk, video, keyboard, serial port, printer), allowing a programmer to communicate with these devices without needing detailed knowledge of their operation. The ROM routines are accessed via the Intel 8088/86 software generated interrupts. The interrupts 10H through to 1AH each access a different routine.

Parameters are passed to and from the BIOS routines using the 8088/86 CPU registers. The routines normally preserve all registers except AX and the flags. Some registers are altered if they return values to the calling process.

```
ROM BIOS INTERRUPT ROUTINES
10      Video routines
11      Equipment Check
12      Memory Size Determination
13      Diskette routines
14      Communications routines
15      Cassette
16      Keyboard routines
17      Printer
18      Cassette BASIC
19      Bootstrap loader
1A      Time of Day
```

The interrupts which handle devices are like a gateway which provide access to more than one routine. The routine executed will depend upon the contents of a particular CPU register. Each of the software interrupt calls use the 8088/86 register contents to determine the desired function call. It is necessary to use a C definition of the CPU programming model, this allows the registers to be initialised with the correct values before the interrupt is generated. The definition also provides a convienent place to store the returned register values. Luckily, the definition has already been created, and resides in the header file dos.h. It is a union of type REGS, which has two parts, each structures.

One structure contains the eight bit registers (accessed by .h.), whilst the other structure contains the 16 bit registers (accessed by .x.) To generate the desired interrupt, a special function call has been provided. This function accepts the interrupt number, and pointers to the programming model union for the entry and return register values. The following program demonstrates the use of these concepts to set the display mode to 40x25 color.

```
#include <dos.h>
union REGS regs;
main()
{
        regs.h.ah = 0;
        regs.h.al = 1;
        int86( 0x10, &regs, &regs );
        printf("Fourty by Twenty-Five color mode.");
```

```
}
```

VIDEO ROM BIOS CALLS

The ROM BIOS supports many routines for accessing the video display. The following table illustrates the use of software interrupt 0x10.

```
SCREEN DISPLAY MODE FUNCTION CALL (regs.h.ah = 0)
regs.h.al       Screen Mode
0               40.25 BW
1               40.25 CO
2               80.25 BW
3               80.25 CO
4               320.200 CO
5               320.200 BW
6               640.200 BW
7               Mono-chrome
8               160.200 16col PCjr
9               320.200 16col PCjr
A               640.200 4col PCjr
D               320.200 16col EGA
E               640.200 16col EGA
F               640.350 mono EGA
10              640.350 16col EGA
```

Note: The change screen mode function call also has the effect of clearing the video screen! The other routines associated with the video software interrupt 0x10 are, (Bold represents return values)

```
        Function Call           regs.h.ah               Entry/Exit Values
        Set display mode                0               Video mode in al
        Set cursor type         1               Start line=ch, end line=cl
                                                (Block cursor, cx = 020C)
        Set cursor position     2               Row,column in dh,dl, page number in
bh
        Read cursor position    3               Page number in bh, dh,dl on exit has
row,column
        Read light pen pos      4               ah=0 light pen not active
                                                ah=1 light pen activated
                                                dh,dl has row,column
                                                ch has raster line (0-199)
                                                bx has pixel column (0-139,639)
        Select active page      5               New page number in al
        Scroll active page up   6               Lines to scroll in al(0=all)
                                                upper left row,column in ch,cl
                                                lower right row,col in dh,dl
                                                attribute for blank line in bh
        Scroll active page dn   7               Same as for scroll up
        Read char and attr      8               Active page in bh
                                                al has character
                                                ah has attribute
        Write char and attr     9               Active page in bh
                                                number of characters in cx
                                                character in al
                                                attribute in bl
        Write character         0a              Active page in bh
                                                number of characters in cx
                                                character in al
        Set color palette       0b              Palette color set in bh
                                                (graphics) color value in bl
        Write dot               0c              Row,column in dx,cx
                                                color of pixel in al
```

```
Read dot               0d             Row,column in dx,cx color in al
Write teletype         0e             Character in al, active page in bh
                                      foregrnd color(graphics) in bl
Return video state     0f             Current video mode in al
                                      columns in ah,active page in bh
```

PORT ACCESS

The C language can be used to transfer data to and from the contents of the various registers and controllers associated with the IBM-PC. These registers and control devices are port mapped, and are accessed using special **IN** and **OUT** instructions. Most C language support library's include functions to do this. The following is a brief description of how this may be done.

```
/* #include <conio.h> */
outp( Port_Address, value); /* turboC uses outportb() */
value = inp( Port_address); /* and inportb() */
```

The various devices, and their port values, are shown below,

Port Range	Device
00 - 0f	DMA Chip 8737
20 - 21	8259 PIC
40 - 43	Timer Chip 8253
60 - 63	PPI 8255 (cassette, sound)
80 - 83	DMA Page registers
200 - 20f	Game I/O Adapter
278 - 27f	Reserved
2f8 - 2ff	COM2
378 - 37f	Parallel Printer
3b0 - 3bf	Monochrome Display
3d0 - 3df	Color Display
3f0 - 3f7	Diskette
3f8 - 3ff	COM1

PROGRAMMING THE 6845 VIDEO CONTROLLER CHIP

The various registers of the 6845 video controller chip, resident on the CGA card, are

Port Value	Register Description
3d0	6845 registers
3d1	6845 registers
3d8	DO Register (Mode control)
3d9	DO Register (Color Select)
3da	DI Register (Status)
3db	Clear light pen latch
3dc	Preset light pen latch

PROGRAMMING EXAMPLE FOR THE BORDER COLOR

The register which controls the border color is the Color Select Register located at port 3d9. Bits 0 - 2 determine the border color. The available colors and values to use are,

Value	Color	Value	Color
0	Black	8	Dark Grey
1	Blue	9	Light Blue
2	Green	0a	Light Green
3	Cyan	0b	Light Cyan

```
        4       Red             0c        Light Red
        5       Magenta         0d        Light Magenta
        6       Brown           0e        Yellow
        7       Light Grey      0f        White
```

The following program will set the border color to blue.

```
#include <conio.h> /* needed for outp() */
#include <stdio.h> /* needed for getchar() */
#include <dos.h> /* for REGS definition */
#define CSReg 0x3d9
#define BLUE 1
void cls()
{
        union REGS regs;
        regs.h.ah = 15; int86( 0x10, &regs, &regs );
        regs.h.ah = 0; int86( 0x10, &regs, &regs );
}
main()
{
        cls();
        printf("Press any key to set border color to blue.\n");
        getchar();
        outp( CSReg, BLUE );
}
```

PROGRAMMING EXAMPLE FOR 40x25 COLOR MODE

The 6845 registers may be programmed to select the appropiate video mode. This may be done via a ROM BIOS call or directly. The values for each of the registers to program 40.25 color mode are,

```
38,28,2d,0a,1f,6,19,1c,2,7,6,7,0,0,0,0
```

The default settings for the registers for the various screen modes can be found in ROM BIOS listing's and technical reference manuals.

To program the various registers, first write to the address register at port 3d4, telling it which register you are programming, then write the register value to port 3d5. The mode select register must also be programmed. This is located at port 3d8. The sequence of events is,

1: Disable the video output signal

2: Program each register

3: Enable video output, setting mode register

The following program illustrates how to do this,

```
#include <conio.h> /* needed for outp() */
#include <stdio.h> /* needed for getchar() */
#include <process.h> /* needed for system calls */
#define MODE_REG 0x3d8
#define VID_DISABLE 0
#define FOURTY_25 0X28
#define ADDRESS_REG 0x3d4
#define REGISTER_PORT 0x3d5
static int mode40x25[] = {0x38,0x28,0x2d,0x0a,0x1f,6,0x19,0x1c,
2,7,6,7,0,0,0,0 };

void cls()
{
        union REGS regs;
        regs.h.ah = 15; int86( 0x10, &regs, &regs );
```

```
                regs.h.ah = 0; int86( 0x10, &regs, &regs );
        }

        main()
        {
                int loop_count = 0;
                cls();
                printf("Press a key to set 40x25 color mode.\n");
                getchar();
                outp(MODE_REG,VID_DISABLE); /*disable video signal */
                for( ; loop_count < 0x0e; ++loop_count) {
                        outp( ADDRESS_REG, loop_count ); /* set up CRT register */
                        outp( REGISTER_PORT, mode40x25[loop_count]); /* write to reg
*/
                }
                outp( MODE_REG, FOURTY_25); /* switch in mode now */;
                printf("Press key to exit.\n"); getchar();
        }
```

The program should update the video_mode byte stored at 0040:0049 to indicate the change in video state. This type of low level programming is an example of code required for embedded applications (ie, ROM CODE running without DOS or the ROM BIOS chip present).

USING A ROM BIOS INTERRUPT CALL TO SET THE VIDEO MODE

The video chip may be also manipulated by using the ROM BIOS calls. A BIOS interrupt call follows the following syntax,

```
        int86( interrupt_number, &regs1, &regs2);
        where int86() is the function call,
        interrupt_number is the interrupt to generate
        &regs1 defines the input register values
        &regs2 defines the returned register values
```

Int 10h is the interrupt to use for video calls. We will be using the set mode routine, so the values are,

```
        regs.h.ah = 0; /* set mode function call */
        regs.h.al = 1; /* set mode to 40x25 color */
```

The following program illustrates how this all fits together,

```
        #include <dos.h>
        #include <stdio.h>
        #include <process.h>
        union REGS regs;
        void cls()
        {
                regs.h.ah = 15; int86( 0x10, &regs, &regs );
                regs.h.ah = 0; int86( 0x10, &regs, &regs );
        }

        main()
        {
                cls();
                printf("Setting mode to 40x25 color.\n");
                printf("Press any key....\n");
                getchar();
                regs.h.ah = 0;
                regs.h.al = 1;
                int86( 0x10, &regs, &regs);
                printf("Mode has been set.\n");
        }
```

Calls to the ROM BIOS Int 10h Set Video Mode routine also update the video_mode flag stored at 0040:0049.

EXTENDED VIDEO BIOS CALLS

The following video calls are not supported on all machines. Some calls are only present if an adapter card is installed (ie, EGA or VGA card).

```
        AH = 10h Select Colors in EGA/VGA
                AL = 1 BL = color register (0 - 15), BH = color to set
                AL = 2 ES:DX ptr to change all 16 colors and overscan number
                AL = 3 BL = color intensity bit. 0 = intensity, 1 = blinking
                (VGA systems only)
                AL = 7 BL = color register to get into BH
                AL = 8 BH = returned overscan value
                AL = 9 ES:DX = address to store all 16 colors + overscan number
                AL = 10h BX = color register to set; ch/cl/dl = green/blue/red
                AL = 12h ES:DX = pointer to change color registers,
                        BX = 1st register to set,
                        CX = number of registers involved
                AL = 13h BL = 0, set color page mode in BH
                        BL = 1, set page number specified by BH
                AL = 15h BX = color register to read; ch/cl/dl = grn/blue/red
                AL = 17h ES:DX = pointer where to load color registers
                        BX = 1st register to set,
                        CX = number of registers involved
                AL = 1Ah get color page information; BL = mode, BH = page number
        AH = 11h Reset Mode with New Character Set
                AL = 0 Load new character set; ES:BP pointer to new table
                        BL/BH = how many blocks/bytes per character
                        CX/DX = number of characters/where to start in block
                AL = 1 BL = block to load the monochrome character set
                AL = 2 BL = block to load the double width character set
                AL = 3 BL = block to select related to attribute
                AL = 4 BL = block to load the 8x16 set (VGA)
                AL = 10h - 14h Same as above, but must be called after a set mode
                AL = 20h ES:BP pointer to a character table, using int 1Fh pointer
                        BL = 0, DL = number of rows, 1=14rows, 2=25rows, 3=43rows
                        CX = number of bytes per character in table
                AL = 22h use 8x14 character set, BL = rows
                AL = 23h use double width character set, BL = rows
                AL = 24h use 8x16 character set, BL = rows
                Get table pointers and other information
                AL = 30h ES:BP = returned pointer; CX = bytes/character; DL = rows
                BH = 0, get int 1Fh pointer, BH = 1, get int 43h pointer
                BH = 2, get 8x14 BH = 3, get double width
                BH = 4, get double width BH = 5, get mono 9x14
                BH = 6, get 8x16 (VGA) BH = 7, get 9x16 (VGA)
        AH = 12h Miscellaneous functions, BL specifies sub-function number
                BL = 10h Get info, BH = 0, now color mode, 1 = now monochrome
                CH/CL = information bits/switches
                BL = 20h Set print screen to work with EGA/VGA
                Functions for VGA only (BL = 30-34h, AL returns 12h)
                BL = 30h Set number of scan lines, 0=200, 1=350, 2=400
                BL = 31h AX = 0/1 Allow/Prevent palette load with new mode
                BL = 32h AL = 0/1 Video OFF/ON
                BL = 33h AL = 0/1 Grey scale summing OFF/ON
                BL = 34h AL = 0/1 Scale cursor size to font size OFF/ON
                BL = 35h Switch between adapter and motherboard video
                AL = 0, adapter OFF, ES:DX = save state area
                AL = 1, motherboard on
                AL = 2, active video off, ES:DX = save area
```

```
        AL = 3, inactive video on, ES:DX = save area
        BL = 36h, AL = 0/1 Screen OFF/ON
AH = 13h Write Character string(cr,lf,bell and bs as operators)
        AL = 0/1 cursor NOT/IS moved, BL = attribute for all chars
        AL = 2/3 cursor NOT/IS moved, string has char/attr/char/attr
                BH = page number, 0 = 1st page, CX = number of characters
                DH/DL = row/column to start, ES:BP = pointer to string
AH = 14h LCD Support
        AL = 0h ES:DI = pointer to font table to load
                BL/BH = which blocks/bytes per character
                CX/DX = number of characters/where to start in block
        AL = 1h BL = block number of ROM font to load
        AL = 2h BL = enable high intensity
AH = 15h Return LCD information table Pointer in ES:DI, AX has screen mode
AH = 16h GET/SET display type (VGA ONLY)
        AL = 0h Get display type BX = displays used, AL = 1Ah
        AL = 1h Set display type BX = displays to use, returns AL = 1Ah
AH = 1Bh Get Video System Information (VGA ONLY)
        Call with BX = 0; ES:DI ptr to buffer area
AH = 1Ch Video System Save & Restore Functions (VGA ONLY)
        AL = 0 Get buffer size
        AL = 1 Save system, buffer at ES:BX
        AL = 2 Restore system, buffer at ES:BX
                CX = 1 For hardware registers
                CX = 2 For software states
                CX = 4 For colors and DAC registers
```

SYSTEM VARIABLES IN LOW MEMORY

```
0040:0000        Address of RS232 card COM1
0040:0002        Address of RS232 card COM2
0040:0004        Address of RS232 card COM3
0040:0006        Address of RS232 card COM4
0040:0008        Address of Printer port LPT1
0040:000A        Address of Printer port LPT2
0040:000C        Address of Printer port LPT3
0040:0010        Equipment bits: Bits 13,14,15 = number of printers
                        12 = game port attached
                        9,10,11 = number of rs232 cards
                        6,7 = number of disk drives
                        4,5 = Initial video mode
                                (00=EGA, 01=CGA40,10=CGA80, 11=MONO)
                        3,2 = System RAM size
                        1 = Maths co-processor
                        0 = Boot from drives?
0040:0013        Main RAM Size
0040:0015        Channel IO size
0040:0017        Keyboard flag bits (byte) 7=ins, 6=caps, 5=num, 4=scrll,
                        3=ALT, 2=CTRL, 1=LSHFT, 0=RSHFT (toggle states)
0040:0018        Keyboard flag bits (byte) (depressed states)
0040:0019        Keyboard ALT-Numeric pad number buffer area
0040:001A        Pointer to head of keyboard queue
0040:001C        Pointer to tail of keyboard queue
0040:001E        15 key queue (head=tail, queue empty)
0040:003E        Recalibrate floppy drive, 1=drive0, 2=drv1, 4=drv2, 8=drv3
0040:003F        Disk motor on status, 1=drive0, 2=drv1, 4=drv2, 8=drv3
                        80h = disk write in progress
0040:0040        Disk motor timer 0=turn off motor
0040:0041        Disk controller return code
                        1=bad cmd, 2=no address mark, 3=cant write, 4=sector not
```

```
found
                        8=DMA overrun,9=DMA over 64k
                        10h=CRC error,20h=controller fail, 40h=seek fail,80h=timeout
0040:0042        Disk status bytes (seven)
0040:0049        Current Video Mode (byte)
0040:004A        Number of video columns
0040:004C        Video buffer size in bytes
0040:004E        Segment address of current video memory
0040:0050        Video cursor position page 0, bits8-15=row,bits0-7=column
0040:0052        Video cursor position page 1, bits8-15=row,bits0-7=column
0040:0054        Video cursor position page 2, bits8-15=row,bits0-7=column
0040:0056        Video cursor position page 3, bits8-15=row,bits0-7=column
0040:0058        Video cursor position page 4, bits8-15=row,bits0-7=column
0040:005A        Video cursor position page 5, bits8-15=row,bits0-7=column
0040:005C        Video cursor position page 6, bits8-15=row,bits0-7=column
0040:005E        Video cursor position page 7, bits8-15=row,bits0-7=column
0040:0060        Cursor mode, bits 8-12=start line, 0-4=end line
0040:0062        Current video page number
0040:0063        Video controller base I/O port address
0040:0065        Hardware mode register bits
0040:0066        Color set in CGA mode
0040:0067        ROM initialisation pointer
0040:0069        ROM I/O segment address
0040:006B        Unused interrupt occurrences
0040:006C        Timer low count (every 55milliseconds)
0040:006E        Timer high count
0040:0070        Timer rollover (byte)
0040:0071        Key-break, bit 7=1 if break key is pressed
0040:0072        Warm boot flag, set to 1234h for warm boot
0040:0074        Hard disk status byte
0040:0075        Number of hard disk drives
0040:0076        Head control byte for hard drives
0040:0077        Hard disk control port (byte)
0040:0078 - 7B   Countdown timers for printer timeouts LPT1 - LPT4
0040:007C - 7F   Countdown timers for RS232 timeouts, COM1 - COM4
0040:0080        Pointer to beginning of keyboard queue
0040:0082        Pointer to end of keyboard queue
```

Advanced Video Data, EGA/VGA

```
0040:0084        Number of rows - 1
0040:0085        Number of pixels per character * 8
0040:0087        Display adapter options (bit3=0 if EGA card is active)
0040:0088        Switch settings from adapter card
0040:0089 - 8A   Reserved
0040:008B        Last data rate for diskette
0040:008C        Hard disk status byte
0040:008D        Hard disk error byte
0040:008E        Set for hard disk interrupt flag
0040:008F        Hard disk options byte, bit0=1 when using a single
                 controller for both hard disk and floppy
0040:0090        Media state for drive 0 Bits 6,7=data transfer rate
                        (00=500k,01=300k,10=250k)
                        5=two steps?(80tk as 40k) 4=media type 3=unused
                        2,1,0=media/drive state (000=360k in 360k drive)
                        (001=360k in 1.2m drive) (010=1.2m in 1.2m drive)
                        (011=360k in 360k drive) (100=360k in 1.2m drive)
                        (101=1.2m in 1.2m drive) (111=undefined )
0040:0091        Media state for drive 1
0040:0092        Start state for drive 0
0040:0093        Start state for drive 1
0040:0094        Track number for drive 0
0040:0095        Track number for drive 1
0040:0096 - 97   Advanced keyboard data
0040:0098 - A7   Real time clock and LAN data
```

```
0040:00A8 - FF   Advanced Video data
0050:0000        Print screen status 00=ready,01=in progress,FFh=error
```

LIBRARIES

A library is a collection of useful routines or modules which perform various functions. They are grouped together as a single unit for ease of use. If the programmer wishes to use a module contained in a library, they only need to specify the module name, observing the correct call/return conditions. C programmers use librarys all the time, they are just unaware of it. The vast majority of routines such as printf, scanf, etc, are located in a C library that the linker joins to the code generated by the compiler. In this case, we will generate a small library which contains the modules *rdot()* and *wdot()*.

These modules read and write a dot to the video screen respectively. The programmer could retain the object code for these seperately, instead of placing them in a library, but the use of a library makes life simpler in that a library contains as many routines as you incorporate into it, thus simplifying the linking process. In other words, a library just groups object modules together under a common name.

The following programs describe the source code for the modules *rdot()* and *wdot()*.

```
#include <dos.h>
union REGS regs;
wdot( int row, int column, unsigned int color )
{
        regs.x.dx = row; regs.x.cx = column;
        regs.h.al = color; regs.h.ah = 12;
        int86( 0x10, &regs, &regs);
}

unsigned int rdot( int row, int column )
{
        regs.x.dx = row; regs.x.cx = column;
        regs.h.ah = 13; int86( 0x10, &regs, &regs);
        return( regs.h.al );
}
```

The modules are compiled into object code. If we called the source code VIDCALLS.C then the object code will be VIDCALLS.OBJ. To create a library requires the use of the LIB.EXE program. This is invoked from the command line by typing **LIB**

Enter in the name of the library you wish to create, in this case **VIDCALLS**. The program will check to see if it already exists, which it doesn't, so answer **Y** to the request to create it. The program requests that you now enter in the name of the object modules. The various operations to be performed are,

```
to add an object module +module_name
to remove an object module -module_name
```

Enter in **+vidcalls**. It is not necessary to include the extension. The program then requests the list file generated by the compiler when the original source was compiled. If a list file was not generated, just press enter, otherwise specify the list file. That completes the generation of the VIDCALLS.LIB library.

The routines in VIDCALLS.LIB are incorporated into user programs as follows,

```
/* source code for program illustrating use of VIDCALLS.LIB */
#include <dos.h>
union REGS regs;
extern void wdot();
extern int rdot();
main()
{
        int color;
        regs.h.ah = 0; /* set up 320 x 200 color */
        regs.h.al = 4;
        int86( 0x10, &regs, &regs );
        wdot( 20, 20, 2 );
```

```
        color = rdot( 20, 20 );
        printf("Color value of dot @20.20 is %d.\n", color);
}
```

The program is compiled then linked. When the linker requests the library name, enter **+VIDCALLS**. This includes the rdot() and wdot() modules at linking time. If this is not done, the linker will come back with unresolved externals error on the rdot() and wdot() functions.

```
; Microsoft C
; msc source;
; link source,,+vidcalls;
;
; TurboC
; tcc -c -ml -f- source.c
; tlink c0l source,source,source,cl vidcalls
```

ACCESSING MEMORY

The following program illustrates how to access memory. A **far pointer** called *scrn* is declared to point to the video RAM. This is actually accessed by use of the ES segment register. The program fills the video screen with the character A

```
#include <stdio.h> /* HACCESS1.C */
main()
{
        char far *scrn = (char far *) 0xB8000000;
        short int attribute = 164; /* Red on green blinking */
        int full_screen = 80 * 25 * 2, loop = 0;
        for( ; loop < full_screen; loop += 2 ) {
                scrn[ loop ] = 'A';
                scrn[ loop + 1 ] = attribute;
        }
        getchar();
}
```

The declaration of a far pointer specifies a 32 bit address. However the IBM-PC uses a 20 bit address bus. This 20 bit physical address is generated by adding the contents of a 16 bit offset to a SEGMENT register.

The Segment register contains a 16 bit value which is left shifted four times, then the 16 bit offset is added to this generating the 20 bit physical address.

```
Segment register = 0xB000 = 0xB0000
Offset = 0x0010 = 0x 0010
Actual 20 bit address = 0xB0010
```

When specifying the pointer using C, the full 32 bit combination of segment:offset is used, eg,

```
char far *memory_pointer = (char far *) 0xB0000000;
```

Lets consider some practical applications of this now. Located at segment 0x40, offset 0x4A is the number of columns used on the current video screen. The following program shows how to access this,

```
main()
{
        char far *video_parameters = (char far *) 0x00400000;
        int columns, offset = 0x4A;
        columns = video_parameters[offset];
        printf("The current column setting is %d\n", columns);
}
```

A practical program illustrating this follows. It directly accesses the video_parameter section of low memory using a far pointer, then prints out the actual mode using a message.

```
        main() /* HACCESS2.C */
        {
                static char *modes[] = { "40x25 BW", "40x25 CO", "80X25 BW",
                "80X25 CO", "320X200 CO", 320X200 BW",
                "640X200 BW", "80X25 Monitor" };
                char far *video_parameters = (char far *) 0x00400000;
                int crt_mode = 0x49, crt_columns = 0x4A;
                printf("The current video mode is %d\n",
modes[video_parameters[crt_mode]]);
                printf("The current column width is %d\n",
video_parameters[crt_columns]);
        }
```

An adaptation of this technique is the following two functions. **Poke** stores a byte value at the specified segment:offset, whilst **Peek** returns the byte value at the specified segment:offset pair.

```
        void poke( unsigned int seg, unsigned int off, char value) {
                char far *memptr;
                memptr = (char far *) MK_FP( seg, off);
                *memptr = value;
        }

        char peek( unsigned int seg, unsigned int off ) {
                char far *memptr;
                memptr = (char far *) MK_FP( seg, off);
                return( *memptr );
        }
```

The program **COLOR** illustrates the use of direct access to update the foreground color of a CGA card in text mode. It accepts the color on the command line, eg, COLOR RED will set the foreground color to RED.

```
        #include <string.h> /* Color.c */
        #include <dos.h>
        #include <stdio.h>
        #include <conio.h>
        #define size 80*25*2

        struct table {
                char *string;
                int value;
        } colors[] = { {"BLACK" , 0, "BLUE" , 1 },
                {"GREEN" , 2, "CYAN" , 3 },
                {"RED" , 4, "MAGENTA" , 5 },
                {"BROWN" , 6, "LIGHT_GRAY" , 7 },
                {"DARK_GRAY" , 8, "LIGHT_BLUE" , 9 },
                {"LIGHT_GREEN" ,10, "LIGHT_CYAN" ,11 },
                {"LIGHT_RED" ,12, "LIGHT_MAGENTA" ,13 },
                {"YELLOW" ,14, "WHITE" ,15 } };

        int getcolor( char *color) {
                char *temp;
                int loop, csize;
                temp = color;
                while( *temp )
                        *temp++ = toupper( *temp );
                csize = sizeof(colors) / sizeof(colors[0]);
                for( loop = 0; loop < csize; loop++ )
                        if( strcmp(color, colors[loop].string) == 0)
                                return(colors[loop].value;
```

```
        return( 16 );
}

void setcolor( int attr ) {
        int loop;
        char far *scrn = (char far *) 0xb8000000;
        for( loop = 1; loop < size; loop += 2 )
                scrn[loop] = attr;
}

main( int argc, char *argv[] ) {
        int ncolor;
        if( argc == 2 ) {
                ncolor = getcolor( argv[1] );
                if( ncolor == 16 )
                        printf("The color %s is not available.\n", argv[1]);
                else
                        setcolor( ncolor );
        }
}
```

POINTERS TO FUNCTIONS

Pointers to functions allow the creation of jump tables and dynamic routine selection. A pointer is assigned the start address of a function, thus, by typing the pointer name, program execution jumps to the routine pointed to. By using a single pointer, many different routines could be executed, simply by re-directing the pointer to point to another function. Thus, programs could use this to send information to a printer, console device, tape unit etc, simply by pointing the pointer associated with output to the appropiate output function!

The following program illustrates the use of pointers to functions, in creating a simple shell program which can be used to specify the screen mode on a CGA system.

```
#include <stdio.h> /* Funcptr.c */
#include <dos.h>
#define dim(x) (sizeof(x) / sizeof(x[0]) )
#define GETMODE 15
#define SETMODE 0
#define VIDCALL 0X10
#define SCREEN40 1
#define SCREEN80 3
#define SCREEN320 4
#define SCREEN640 6
#define VID_BIOS_CALL(x) int86( VIDCALL, &x, &x )
int cls(), scr40(), scr80(), scr320(), scr640(), help(), shellquit();
union REGS regs;
struct command_table {
        char *cmd_name;
        int (*cmd_ptr) ();

}cmds[]={"40",scr40,"80",scr80,"320",scr320,"640",scr640,"HELP",help,"CLS",cls,"EXIT",shellquit};

cls() {
        regs.h.ah = GETMODE; VID_BIOS_CALL( regs );
        regs.h.ah = SETMODE; VID_BIOS_CALL( regs );
}

scr40() {
        regs.h.ah = SETMODE;
        regs.h.al = SCREEN40;
        VID_BIOS_CALL( regs );
}
```

```
scr80() {
        regs.h.ah = SETMODE;
        regs.h.al = SCREEN80;
        VID_BIOS_CALL( regs );
}

scr320() {
        regs.h.ah = SETMODE;
        regs.h.al = SCREEN320;
        VID_BIOS_CALL( regs );
}

scr640() {
        regs.h.ah = SETMODE;
        regs.h.al = SCREEN640;
        VID_BIOS_CALL( regs );
}

shellquit() {
        exit( 0 );
}

help() {
        cls();
        printf("The available commands are; \n");
        printf(" 40 Sets 40 column mode\n");
        printf(" 80 Sets 80 column mode\n");
        printf(" 320 Sets medium res graphics mode\n");
        printf(" 640 Sets high res graphics mode\n");
        printf(" CLS Clears the display screen\n");
        printf(" HELP These messages\n");
        printf(" EXIT Return to DOS\n");
}

get_command( char *buffer ) {
        printf("\nShell: ");
        gets( buffer );
        strupr( buffer );
}

execute_command( char *cmd_string ) {
        int i, j;
        for( i = 0; i < dim( cmds); i++ ) {
                j = strcmp( cmds[i].cmd_name, cmd_string );
                if( j == 0 ) {
                        (*cmds[i].cmd_ptr) ();
                        return 1;
                }
        }
        return 0;
}

main() {
        char input_buffer[81];
        while( 1 ) {
                get_command( input_buffer );
                if( execute_command( input_buffer ) == 0 )
                        help();
        }
}
```

Copyright Brian Brown, 1986-2000. All rights reserved.

INDEX Next

C Programming

Suggested Model Answers

Exercise C1 The program output is,

Prog1

```
Programming in C is easy.
And so is Pascal.
```

Prog2

```
The black dog was big. The cow jumped over the moon.
```

Prog3

```
Hello...
..oh my
...when do i stop?
```

Exercise C2 Typical program output is,

```
The sum of 35 and 18 is 53
```

Exercise C3 Invalid variable names,

```
value$sum       - must be an underscore, $ sign is illegal
exit flag       - no spaces allowed
3lotsofmoney    - must start with a-z or an underscore
char            - reserved keyword
```

When %X\n is used, the hex digits a to f become A to F

Exercise C4 Constants

```
#define smallvalue  0.312
#define letter      'W'
#define smallint    37
```

Exercise C5

```
The % of 50 by 10 is 0.00
```

Exercise C6

```
#include <stdio.h>

main ()
{
	int  n = 1, t_number = 0;

	for ( ; n <= 200; n++ )
		t_number = t_number + n;

	printf("The 200th triangular number is %d\n", t_number);
}
```

Exercise C7

```
a == 2  this is an equality test
a  = 2  this is an assignment

/* program which illustrates relational assignments */
#include <stdio.h>

main()
{
	int val1 = 50, val2 = 20, sum = 0;

	printf("50 + 20 is %d\n", val1 + val2 );
	printf("50 - 20 is %d\n", val1 - val2 );
	printf("50 * 20 is %d\n", val1 * val2 );
	printf("50 / 20 is %d\n", val1 / val2 );
}
```

Exercise C8

Prints result with two leading places

Exercise C9

```
main()
{
	int  n = 1, t_number = 0, input;

	printf("Enter a number\n");
	scanf("%d", &input);
	for( ; n <= input; n++ )
		t_number = t_number + n;

	printf("The triangular_number of %d is %d\n", input, t_number);
}
```

Exercise C10

```
#include <stdio.h>

main()
{
	int grade;	/* to hold the entered grade */
	float average;	/* the average mark */
	int loop;	/* loop count */
	int sum;	/* running total of all entered grades */
	int valid_entry;	/* for validation of entered grade */
	int failures;	/* number of people with less than 65 */

	sum = 0;	/* initialise running total to 0 */
	failures = 0;

	for( loop = 0; loop < 5; loop = loop + 1 )
	{
		valid_entry = 0;
		while( valid_entry == 0 )
		{
			printf("Enter mark (1-100):");
			scanf(" %d", &grade );
			if ((grade > 1 ) && (grade < 100 ))
			{
				valid_entry = 1;
			}
		}
		if( grade < 65 )
			failures++;
		sum = sum + grade;
	}
	average = (float) sum / loop;
	printf("The average mark was %.2f\n", average );
	printf("The number less than 65 was %d\n", failures );
}
```

Exercise C11

```
#include <stdio.h>

main ()
{
  int   invalid_operator = 0;
  char  operator;
  float number1, number2, result;

  printf("Enter two numbers and an operator in the format\n");
  printf(" number1  operator  number2\n");
```

```
          scanf( "%f %c %f", &number1, &operator, &number2);

          switch( operator )
          {
            case '*' : result = number1 * number2; break;
            case '-' : result = number1 - number2; break;
            case '/' : result = number1 / number2; break;
            case '+' : result = number1 + number2; break;
            default  : invalid_operator = 1;
          }

          switch ( invalid_operator )
          {
            case 1:  printf("Invalid operator.\n");     break;
            default: printf("%2.2f %c %2.2f is %2.2f\n",
                     number1,operator,number2,result);  break;
          }
        }
```

Exercise C12

```
        least_value = 4
```

Exercise C13

```
        #include <stdio.h>

        main()
        {
                static int m[][] = { {10,5,-3}, {9, 0, 0}, {32,20,1}, {0,0,8} };
                int row, column, sum;

                sum = 0;
                for( row = 0; row < 4; row++ )
                        for( column = 0; column < 3; column++ )
                                sum = sum + m[row][column];
                printf("The total is %d\n", sum );
        }
```

Exercise C14

Variables declared type static are initialised to zero. They are created and initialised only once, in their own data segment. As such, they are permanent, and still remain once the function terminates (but disappear when the program terminates).

Variables which are not declared as type static are type automatic by default. C creates these on the stack, thus they can assume non zero values when created, and also disappear once the function that creates them terminates.

Exercise C15

```
#include <stdio.h>
int calc_result( int, int, int );

int calc_result( int var1, int var2, int var3 )
{
  int sum;

  sum = var1 + var2 + var3;
  return( sum );          /* return( var1 + var2 + var3 ); */
}

main()
{
  int numb1 = 2, numb2 = 3, numb3=4, answer=0;

  answer = calc_result( numb1, numb2, numb3 );
  printf("%d + %d + %d = %d\n", numb1, numb2, numb3, answer);
}
```

Exercise C16

```
#include <stdio.h>

int add2darray( int [][5], int );       /* function prototype */

int add2darray( int array[][5], int rows )
{
        int total = 0, columns, row;

        for( row = 0; row < rows; row++ )
                for( columns = 0; columns < 5; columns++ )
                        total = total + array[row][columns];
        return total;
}

main()
{
        int numbers[][] = { {1, 2, 35, 7, 10}, {6, 7, 4, 1, 0} };
        int sum;

        sum = add2darray( numbers, 2 );
        printf("the sum of numbers is %d\n", sum );
}
```

Exercise C17

```
time = time - 5;
a = a * (b + c);
```

Exercise C18

```
#include <stdio.h>
void sort_array( int [], int );

void sort_array( values, number_of_elements )
int values[], number_of_elements;
{
  int index_pointer, base_pointer = 0, temp;

  while ( base_pointer < (number_of_elements - 1) )
  {
    index_pointer = base_pointer + 1;
    while ( index_pointer < number_of_elements )
    {
      if( values[base_pointer] > values[index_pointer] )
      {
        temp = values[base_pointer];
        values[base_pointer]  = values[index_pointer];
        values[index_pointer] = temp;
      }
      ++index_pointer;
    }
    ++base_pointer;
  }
}

main ()
{
  static int array[] = { 4, 0, 8, 3, 2, 9, 6, 1, 7, 5 };
  int number_of_elements = 10, loop_count = 0;

  printf("Before the sort, the contents are\n");
  for ( ; loop_count < number_of_elements; ++loop_count )
    printf("Array[%d] is %d\n", loop_count,array[loop_count]);

  sort_array( array, number_of_elements );

  printf("After the sort, the contents are\n");
  loop_count = 0;
  for( ; loop_count < number_of_elements; ++loop_count )
    printf("Array[%d] is %d\n", loop_count,array[loop_count]);
}
```

Exercise C19

```
#include <stdio.h>
long int triang_rec( long int );

long int triang_rec( long int number )
{
```

```
        long int result;

        if( number == 0l )
          result = 0l;
        else
          result = number + triang_rec( number - 1 );
        return( result );
    }

    main ()
    {
      int request;
      long int triang_rec(), answer;

      printf("Enter number to be calculated.\n");
      scanf( "%d", &request);

      answer = triang_rec( (long int) request );
      printf("The triangular answer is %l\n", answer);
    }


    Note this version of function triang_rec

    #include <stdio.h>
    long int triang_rec( long int );

    long int triang_rec( long int number )
    {
       return((number == 0l) ? 0l : number*triang_rec( number-1));
    }
```

Exercise C20

```
    b
```

Exercise C21

```
    #include <stdio.h>

    struct date {
            int day, month, year;
    };

    int days[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
    struct date today, tommorrow;

    void gettodaysdate( void );

    void gettodaysdate( void )
    {
```

```
            int valid = 0;

            while( valid == 0 ) {
                    printf("Enter in the current year (1990-2000)-->");
                    scanf("&d", &today.year);
                    if( (today.year < 1990) || (today.year > 1999) )
                            printf("\007Invalid year\n");
                    else
                            valid = 1;
            }
            valid = 0;
            while( valid == 0 ) {
                    printf("Enter in the current month (1-12)-->");
                    scanf("&d", &today.month);
                    if( (today.month < 1) || (today.month > 12) )
                            printf("\007Invalid month\n");
                    else
                            valid = 1;
            }
            valid = 0;
            while( valid == 0 ) {
                    printf("Enter in the current day (1-%d)-->",
days[today.month-1]);
                    scanf("&d", &today.day);
                    if( (today.day < 1) || (today.day > days[today.month-1]) )
                            printf("\007Invalid day\n");
                    else
                            valid = 1;
            }
    }

    main()
    {

            gettodaysdate();
            tommorrow = today;
            tommorrow.day++;
            if( tommorrow.day > days[tommorrow.month-1] ) {
                    tommorrow.day = 1;
                    tommorrow.month++;
                    if( tommorrow.month > 12 )
                            tommorrow.year++;
            }
            printf("Tommorrows date is %02d:%02d:%02d\n", \
                    tommorrow.day, tommorrow.month, tommorrow.year );
    }
```

Exercise C22

```
    #include <stdio.h>

    struct  date  {         /* Global definition of date */
            int day, month, year;
    };
```

```
main()
{
        struct date dates[5];
        int i;

        for( i = 0; i < 5; ++i ) {
                printf("Please enter the date (dd:mm:yy)" );
                scanf("%d:%d:%d", &dates[i].day, &dates[i].month,
                        &dates[i].year );
        }
}
```

Exercise C23

```
count = 10, x = 10;

Q Q
/ /
( (
```

Exercise C24

```
i1 = 5, i2 = 12, *p1 = 5; *p2 = 5
```

Exercise C25

```
Name = Baked Beans
ID = 312
Price = 2.75
```

Exercise C26

```
Name = Apple Pie
ID = 123
Price = 1.65

Name = Greggs Coffee
ID = 773
Price = 3.20
```

Exercise C27

```
#include <stdio.h>

main( int argc, char *argv[])
{
  FILE *in_file, *out_file, *fopen();
```

```
        int c;

        if( argc != 3 )
        {
          printf("Incorrect, format is FCOPY source dest\n");
          exit(2);
        }
        in_file = fopen( argv[1], "r");
        if( in_file == NULL )  printf("Cannot open %s for reading\n", argv[1]);
        else
        {
          out_file = fopen( argv[2], "w");
          if ( out_file == NULL )   printf("Cannot open %s for writing\n",
argv[2]);
          else
          {
             printf("File copy program, copying %s to %s\n", argv[1],  argv[2]);
             while ( (c=getc( in_file) ) != EOF )  putc( c, out_file );
             putc( c, out_file);                  /* copy EOF */
             printf("File has been copied.\n");
             fclose( out_file);
          }
        fclose( in_file);
        }
      }
```

Practise Exercise 1: Answers

```
1.      int  sum;

2.      char  letter;

3.      #define  TRUE  1

4.      float  money;

5.      double  arctan;

6.      int  total = 0;

7.      int loop;

8.      #define GST  0.125
```

Practise Exercise 2: Answers

```
1.      total = number1;

2.      sum = loop_count  +  petrol_cost;

3.      discount = total  /  10;
```

```
4.       letter = 'W';

5.       costing = (float) sum  / 0.25;
```

Practise Exercise 3: Answers

```
1.       printf("%d", sum );

2.       printf("Welcome\n");

3.       printf("%c", letter );

4.       printf("%f", discount );

5.       printf("%.2f", dump );

6.       scanf("%d", &sum );

7.       scanf("%f", &discount_rate );

8.       scanf("  %c", &operator );
```

Practise Exercise 4: Answers

```
1.       for(  loop = 1;  loop <= 10; loop++ )
             printf("%d\n", loop );

2.       for(  loop = 1; loop <= 5; loop++ ) {
            for( count = 1; count <= loop; count++ )
               printf("%d", loop );
            printf("\n");
         }

3.       total = 0;
         for(  loop = 10; loop <= 100; loop++ )
            total = total + loop;

         or

         for(  loop = 10, total = 0; loop <= 100; loop++ )
            total = total + loop;

5.       for( loop = 'A';  loop <= 'Z';  loop++ )
            printf("%c", loop );
```

Practise Exercise 5: Answers

```
1.       loop = 1;
         while( loop <= 10 ) {
                 printf("%d", loop );
```

```
                loop++;
        }
```

```
2.      loop = 1;
        while ( loop <= 5 ) {
                count = 1;
                while( count <= loop )
                        printf("%d", loop);
                printf("\n");
        }
```

```
3.      if( sum < 65 )
                printf("Sorry. Try again");
```

```
4.      if( total == good_guess )
                printf("%d", total );
        else
                printf("%d", good_guess );
```

Practise Exercise 6: Answers

```
1.      if( (sum == 10)  && (total < 20) )
                printf("incorrect.");
```

```
2.      if(  (flag == 1)  ||  (letter != 'X') )
                exit_flag = 0;
        else
                exit_flag = 1;
```

```
3.      switch( letter ) {
                case 'X' : sum = 0; break;
                case 'Z' : valid_flag = 1; break;
                case 'A' : sum = 1; break;
                default: printf("Unknown letter -->%c\n", letter ); break;
        }
```

Practise Exercise 7: Answers

```
1.      char  letters[10];
```

```
2.      letters[3] = 'Z';
```

```
3.      total = 0;
        for( loop = 0; loop < 5; loop++ )
                total = total + numbers[loop];
```

```
4.      float  balances[3][5];
```

```
5.      total = 0.0;
        for( row = 0; row < 3; row++ )
                for( column = 0; column < 5; column++ )
```

```
                        total = total + balances[row][column];

6.      char  words[] = "Hello";

7.      strcpy(  stuff, "Welcome" );

8.      printf("%d", totals[2] );

9.      printf("%s", words );

10.     scanf(" %s", &words[0] );

        or

        scanf("  %s", words );

11.     for( loop = 0; loop < 5; loop++ )
           scanf(" %c", &words[loop] );
```

Practise Exercise 8: Answers

```
1.      void  menu( void )
        {
                printf("Menu choices");
        }

2.      void  menu( void );

3.      void  print(  char  message[] )
        {
                printf("%s", message );
        }

4.      void  print(  char  [] );

5.      int  total( int  array[], int elements )
        {
                int  count, total = 0;
                for( count = 0; count < elements; count++ )
                        total = total + array[count];
                return total;
        }

6.      int  total(  int  [],  int );
```

Practise Exercise 9: Answers

```
1.      struct  client {
                int     count;
                char  text[10];
                float  balance;
```

```
        };

2.      struct  date  today;

3.      struct  client  clients[10];

4.      clients[2].count = 10;

5.      printf("%s", clients[0].text );

6.      struct birthdays
        {
                struct time  btime;
                struct date  bdate;
        };
```

Practise Exercise 9A: Answers

```
1.      FILE *input_file;

2.      input_file = fopen( "results.dat", "rt" );

3.      if( input_file == NULL ) {
                printf("Unable to open file.\n");\
                exit(1);
        }

4.      int ch, loop = 0;

        ch = fgetc( input_file );
        while( ch != '\n' ) {
                buffer[loop] = ch;
                loop++;
                ch = fgetc( input_file );
        }
        buffer[loop] = NULL;

5.      fclose( input_file );
```

Practise Exercise 10: Answers

```
1.      int  *address;

2.      temp = &balance;

3.      *letter = 'W';

4.      count = 20, *temp = 20, sum = 20

5.      char  *message = "Hello";
```

```
6.      array = (char *) getmem( 200 );
```

Practise Exercise 11: Answers

```
1.      struct  date  *dates;

2.      (*dates).day = 10;
        or
        dates->day = 10;

3.      struct  machine  {
                int  name;
                char  *memory;
        };

4.      mpu641->memory = (char *) NULL;

5.      mpu641->memory = CPUtype;

        [       -> means mpu641 is a pointer to a structure                    ]
        [       memory is a pointer, so is assigned an address (note &)        ]
        [       the name of an array is equivalent to address of first element  ]

6.      mpu641->name = 10;

        [       -> means mpu641 is a pointer to a structure            ]
        [       name is a variable, so normal assignment is possible    ]


7.      *(times->day) = 10;

        [       -> means times is a pointer to a structure             ]
        [       day is a pointer, so to assign a value requires * operator      ]
        [       *times->day is not quite correct                                ]
        [       using the pointer times, goto the day field            ]
times->day
        [       this is an address                                     ]       x
        [       let the contents of this address be equal to 10        ]       *(x)
= 10


8.      *(times[2]->month) = 12;
```

Practise Exercise 11a: Answers

```
1. Before call to editrecord()
        item.name = "Red Plum Jam"
        item.id = 0
        item.price = 0.0

2. After return from editrecord()
        item.name = "Baked Beans"
```

```
        item.id = 220
        item.price = 2.20

3. The final values of values, item.name, item.id, item.price
        item.name = "Baked Beans"
        item.id = 220
        item.price = 2.75
```

[Practise Exercise 12: Answers]

```
1.      struct  node  {
                int  data;
                struct  node  *next_node;
        };

2.      struct  node  node1, node2, node3;

3.      node1.next = &node2;
        node2.next = &node3;
        node3.next = (struct node *) NULL;

4.      while( list != (struct node *) NULL ) {
                printf("data = %d\n", list->data );
                list = list->next_node;
        }

5.      terminates the list at node2, effectively deleting node3 from the list.

6.      new_node->next = list->next;
        list->next = new_node;

7.      void  delete_node( struct node *head, struct node *delnode )
        {
                struct node *list;

                list = head;
                while( list->next != delnode ) {
                        list = list->node;

                list->next = delnode->next;
        }

8.      void insert_node( struct node *head, struct node *newnode, struct node
*prevnode )
        {
                struct node *list;

                list = head;
                while( list != prevnode )
                        list = list->next;

                newnode->next = list->next;
```

```
            list->next = newnode;
    }
```

© Copyright B Brown. 1984-2000. All rights reserved.

Index of /courseware/cprogram

Name	Last modified	Size	Description
Parent Directory	04-Jul-2000 08:54	-	
advcw1.htm	04-Jul-2000 08:15	31k	
advcw2.htm	04-Jul-2000 08:15	35k	
advcw3.htm	04-Jul-2000 08:15	43k	
c_000.htm	04-Jul-2000 08:15	4k	
c_000a.htm	04-Jul-2000 08:15	2k	
c_001.htm	04-Jul-2000 08:15	2k	
c_001a.htm	04-Jul-2000 08:15	1k	
c_002.htm	04-Jul-2000 08:15	4k	
c_003.htm	04-Jul-2000 08:15	2k	
c_003a.htm	04-Jul-2000 08:15	1k	
c_004.htm	04-Jul-2000 08:15	3k	
c_005.htm	04-Jul-2000 08:15	3k	
c_005a.htm	04-Jul-2000 08:15	1k	
c_006.htm	04-Jul-2000 08:15	4k	
c_007.htm	04-Jul-2000 08:15	4k	
c_008.htm	04-Jul-2000 08:15	3k	
c_009.htm	04-Jul-2000 08:15	3k	
c_010.htm	04-Jul-2000 08:15	6k	
c_010a.htm	04-Jul-2000 08:15	1k	
c_011.htm	04-Jul-2000 08:15	2k	

c_012.htm	04-Jul-2000 08:15	2k
c_012a.htm	04-Jul-2000 08:15	2k
c_012s.htm	04-Jul-2000 08:15	8k
c_013.htm	04-Jul-2000 08:15	4k
c_013a.htm	04-Jul-2000 08:15	1k
c_014.htm	04-Jul-2000 08:15	2k
c_014a.htm	04-Jul-2000 08:15	1k
c_014s.htm	04-Jul-2000 08:15	7k
c_015.htm	04-Jul-2000 08:15	3k
c_015a.htm	04-Jul-2000 08:15	3k
c_016.htm	04-Jul-2000 08:15	4k
c_017.htm	04-Jul-2000 08:15	2k
c_017a.htm	04-Jul-2000 08:15	2k
c_017s.htm	04-Jul-2000 08:15	10k
c_018.htm	04-Jul-2000 08:15	2k
c_019.htm	04-Jul-2000 08:15	5k
c_019a.htm	04-Jul-2000 08:15	4k
c_019b.htm	04-Jul-2000 08:15	1k
c_020.htm	04-Jul-2000 08:15	2k
c_020a.htm	04-Jul-2000 08:15	2k
c_021.htm	04-Jul-2000 08:15	2k
c_021a.htm	04-Jul-2000 08:15	2k
c_021s.htm	04-Jul-2000 08:15	7k
c_022.htm	04-Jul-2000 08:15	2k
c_023.htm	04-Jul-2000 08:15	4k

c_024.htm	04-Jul-2000 08:15	4k
c_024a.htm	04-Jul-2000 08:15	2k
c_025.htm	04-Jul-2000 08:15	3k
c_026.htm	04-Jul-2000 08:15	2k
c_026a.htm	04-Jul-2000 08:15	2k
c_026s.htm	04-Jul-2000 08:15	7k
c_027.htm	04-Jul-2000 08:15	3k
c_027a.htm	04-Jul-2000 08:15	2k
c_027b.htm	04-Jul-2000 08:15	3k
c_028.htm	04-Jul-2000 08:15	3k
c_028a.htm	04-Jul-2000 08:15	2k
c_029.htm	04-Jul-2000 08:15	2k
c_029a.htm	04-Jul-2000 08:15	2k
c_029s.htm	04-Jul-2000 08:15	5k
c_030.htm	04-Jul-2000 08:15	2k
c_031.htm	04-Jul-2000 08:15	4k
c_032.htm	04-Jul-2000 08:16	3k
c_033.htm	04-Jul-2000 08:16	3k
c_033a.htm	04-Jul-2000 08:16	1k
c_034.htm	04-Jul-2000 08:16	3k
c_035.htm	04-Jul-2000 08:16	2k
c_036.htm	04-Jul-2000 08:16	2k
c_037.htm	04-Jul-2000 08:16	2k
c_037a.htm	04-Jul-2000 08:16	1k
c_038.htm	04-Jul-2000 08:16	2k

c_039.htm	04-Jul-2000 08:16	1k
c_040.htm	04-Jul-2000 08:16	2k
c_041.htm	04-Jul-2000 08:16	2k
c_041a.htm	04-Jul-2000 08:16	3k
c_041s.htm	04-Jul-2000 08:16	13k
c_042.htm	04-Jul-2000 08:16	3k
c_043.htm	04-Jul-2000 08:16	2k
c_044.htm	04-Jul-2000 08:16	3k
c_045.htm	04-Jul-2000 08:16	3k
c_045a.htm	04-Jul-2000 08:16	1k
c_046.htm	04-Jul-2000 08:16	5k
c_046a.htm	04-Jul-2000 08:16	1k
c_047.htm	04-Jul-2000 08:16	3k
c_048.htm	04-Jul-2000 08:16	2k
c_049.htm	04-Jul-2000 08:16	2k
c_049a.htm	04-Jul-2000 08:16	1k
c_050.htm	04-Jul-2000 08:16	3k
c_051.htm	04-Jul-2000 08:16	1k
c_051a.htm	04-Jul-2000 08:16	1k
c_052.htm	04-Jul-2000 08:16	2k
c_052a.htm	04-Jul-2000 08:16	2k
c_053.htm	04-Jul-2000 08:16	2k
c_053a.htm	04-Jul-2000 08:16	1k
c_054.htm	04-Jul-2000 08:16	2k
c_054a.htm	04-Jul-2000 08:16	2k

c_054s.htm	04-Jul-2000 08:16	7k
c_055.htm	04-Jul-2000 08:16	4k
c_055a.htm	04-Jul-2000 08:16	7k
c_056.htm	04-Jul-2000 08:16	2k
c_057.htm	04-Jul-2000 08:16	3k
c_058.htm	04-Jul-2000 08:16	1k
c_058a.htm	04-Jul-2000 08:16	1k
c_059.htm	04-Jul-2000 08:16	3k
c_060.htm	04-Jul-2000 08:16	2k
c_061.htm	04-Jul-2000 08:16	2k
c_061a.htm	04-Jul-2000 08:16	2k
c_062.htm	04-Jul-2000 08:16	2k
c_063.htm	04-Jul-2000 08:16	2k
c_064.htm	04-Jul-2000 08:16	2k
c_064a.htm	04-Jul-2000 08:16	1k
c_065.htm	04-Jul-2000 08:16	2k
c_066.htm	04-Jul-2000 08:16	2k
c_067.htm	04-Jul-2000 08:16	2k
c_067a.htm	04-Jul-2000 08:16	2k
c_067s.htm	04-Jul-2000 08:16	6k
c_068.htm	04-Jul-2000 08:16	2k
c_069.htm	04-Jul-2000 08:16	4k
c_070.htm	04-Jul-2000 08:16	2k
c_071.htm	04-Jul-2000 08:16	2k
c_072.htm	04-Jul-2000 08:16	2k

[c_073.htm]()	04-Jul-2000 08:16	2k
[c_074.htm]()	04-Jul-2000 08:16	2k
[c_074a.htm]()	04-Jul-2000 08:16	2k
[c_074s.htm]()	04-Jul-2000 08:16	7k
[c_075.htm]()	04-Jul-2000 08:16	3k
[c_075a.htm]()	04-Jul-2000 08:16	5k
[c_076.htm]()	04-Jul-2000 08:16	3k
[c_077.htm]()	04-Jul-2000 08:16	2k
[c_0771.htm]()	04-Jul-2000 08:16	2k
[c_0771a.htm]()	04-Jul-2000 08:16	1k
[c_077a.htm]()	04-Jul-2000 08:16	2k
[c_077s.htm]()	04-Jul-2000 08:16	6k
[c_078.htm]()	04-Jul-2000 08:16	2k
[c_079.htm]()	04-Jul-2000 08:16	2k
[c_080.htm]()	04-Jul-2000 08:16	2k
[c_081.htm]()	04-Jul-2000 08:16	2k
[c_081a.htm]()	04-Jul-2000 08:16	3k
[c_081s.htm]()	04-Jul-2000 08:16	9k
[c_082.htm]()	04-Jul-2000 08:16	2k
[c_082a.htm]()	04-Jul-2000 08:16	2k
[c_083.htm]()	04-Jul-2000 08:16	2k
[c_083a.htm]()	04-Jul-2000 08:16	1k
[c_084.htm]()	04-Jul-2000 08:16	2k
[c_084a.htm]()	04-Jul-2000 08:16	2k
[c_085.htm]()	04-Jul-2000 08:16	5k

c_085a.htm	04-Jul-2000 08:16	1k
c_086.htm	04-Jul-2000 08:16	3k
c_087.htm	04-Jul-2000 08:16	2k
c_088.htm	04-Jul-2000 08:16	3k
c_088a.htm	04-Jul-2000 08:16	4k
c_089.htm	04-Jul-2000 08:16	2k
c_090.htm	04-Jul-2000 08:16	1k
c_091.htm	04-Jul-2000 08:16	2k
c_092.htm	04-Jul-2000 08:16	2k
c_093.htm	04-Jul-2000 08:16	5k
c_094.htm	04-Jul-2000 08:16	11k
c_095.htm	04-Jul-2000 08:16	2k
c_096.htm	04-Jul-2000 08:16	1k
c_097.htm	04-Jul-2000 08:16	1k
c_098.htm	04-Jul-2000 08:16	2k
c_099.htm	04-Jul-2000 08:16	2k
c_100.htm	04-Jul-2000 08:16	2k
c_101.htm	04-Jul-2000 08:16	1k
c_102.htm	04-Jul-2000 08:16	2k
c_103.htm	04-Jul-2000 08:16	2k
c_104.htm	04-Jul-2000 08:16	2k
c_105.htm	04-Jul-2000 08:16	2k
c_105a.htm	04-Jul-2000 08:16	2k
c_106.htm	04-Jul-2000 08:16	2k
c_107.htm	04-Jul-2000 08:16	4k

c_108.htm	04-Jul-2000 08:16	2k
c_109.htm	04-Jul-2000 08:16	1k
c_110.htm	04-Jul-2000 08:16	20k
cstart.htm	04-Jul-2000 08:16	3k
default.htm	04-Jul-2000 08:16	7k
howtopro.htm	04-Jul-2000 08:16	4k
images/	03-Apr-2000 12:52	-
joystick.htm	04-Jul-2000 08:16	5k
onlinet.htm	04-Jul-2000 08:16	2k
problems.htm	04-Jul-2000 08:16	11k
simple4.htm	04-Jul-2000 08:16	2k

Apache/1.3.12 Server at www.ibilce.unesp.br Port 80

Index of /courseware/cprogram/images

Name	Last modified	Size	Description
Parent Directory	24-May-2000 10:59	-	
ballloc.gif	04-Jul-2000 08:15	1k	
ballrem.gif	04-Jul-2000 08:15	1k	
c_019.gif	04-Jul-2000 08:15	2k	
cam1.jpg	04-Jul-2000 08:15	1k	
disk1.jpg	04-Jul-2000 08:15	1k	
for1.gif	04-Jul-2000 08:15	6k	
for2.gif	04-Jul-2000 08:15	6k	
for3.gif	04-Jul-2000 08:15	6k	
for4.gif	04-Jul-2000 08:15	6k	
for5.gif	04-Jul-2000 08:15	6k	
for6.gif	04-Jul-2000 08:15	6k	
for_loop.avi	04-Jul-2000 08:15	4.3M	
for_loop.jpg	04-Jul-2000 08:15	12k	
for_loop.rm	04-Jul-2000 08:15	740k	
mail.gif	04-Jul-2000 08:15	4k	
menu.gif	04-Jul-2000 08:15	2k	
next.gif	04-Jul-2000 08:15	2k	
previous.gif	04-Jul-2000 08:15	2k	

Apache/1.3.12 Server at www.ibilce.unesp.br Port 80

Index of /courseware/cprogram

Name	Last modified	Size	Description
Parent Directory	04-Jul-2000 08:54	-	
simple4.htm	04-Jul-2000 08:16	2k	
problems.htm	04-Jul-2000 08:16	11k	
onlinet.htm	04-Jul-2000 08:16	2k	
joystick.htm	04-Jul-2000 08:16	5k	
images/	03-Apr-2000 12:52	-	
howtopro.htm	04-Jul-2000 08:16	4k	
default.htm	04-Jul-2000 08:16	7k	
cstart.htm	04-Jul-2000 08:16	3k	
c_110.htm	04-Jul-2000 08:16	20k	
c_109.htm	04-Jul-2000 08:16	1k	
c_108.htm	04-Jul-2000 08:16	2k	
c_107.htm	04-Jul-2000 08:16	4k	
c_106.htm	04-Jul-2000 08:16	2k	
c_105a.htm	04-Jul-2000 08:16	2k	
c_105.htm	04-Jul-2000 08:16	2k	
c_104.htm	04-Jul-2000 08:16	2k	
c_103.htm	04-Jul-2000 08:16	2k	
c_102.htm	04-Jul-2000 08:16	2k	
c_101.htm	04-Jul-2000 08:16	1k	
c_100.htm	04-Jul-2000 08:16	2k	

c_099.htm	04-Jul-2000 08:16	2k
c_098.htm	04-Jul-2000 08:16	2k
c_097.htm	04-Jul-2000 08:16	1k
c_096.htm	04-Jul-2000 08:16	1k
c_095.htm	04-Jul-2000 08:16	2k
c_094.htm	04-Jul-2000 08:16	11k
c_093.htm	04-Jul-2000 08:16	5k
c_092.htm	04-Jul-2000 08:16	2k
c_091.htm	04-Jul-2000 08:16	2k
c_090.htm	04-Jul-2000 08:16	1k
c_089.htm	04-Jul-2000 08:16	2k
c_088a.htm	04-Jul-2000 08:16	4k
c_088.htm	04-Jul-2000 08:16	3k
c_087.htm	04-Jul-2000 08:16	2k
c_086.htm	04-Jul-2000 08:16	3k
c_085a.htm	04-Jul-2000 08:16	1k
c_085.htm	04-Jul-2000 08:16	5k
c_084a.htm	04-Jul-2000 08:16	2k
c_084.htm	04-Jul-2000 08:16	2k
c_083a.htm	04-Jul-2000 08:16	1k
c_083.htm	04-Jul-2000 08:16	2k
c_082a.htm	04-Jul-2000 08:16	2k
c_082.htm	04-Jul-2000 08:16	2k
c_081s.htm	04-Jul-2000 08:16	9k
c_081a.htm	04-Jul-2000 08:16	3k

c_081.htm	04-Jul-2000 08:16	2k
c_080.htm	04-Jul-2000 08:16	2k
c_079.htm	04-Jul-2000 08:16	2k
c_078.htm	04-Jul-2000 08:16	2k
c_077s.htm	04-Jul-2000 08:16	6k
c_077a.htm	04-Jul-2000 08:16	2k
c_0771a.htm	04-Jul-2000 08:16	1k
c_0771.htm	04-Jul-2000 08:16	2k
c_077.htm	04-Jul-2000 08:16	2k
c_076.htm	04-Jul-2000 08:16	3k
c_075a.htm	04-Jul-2000 08:16	5k
c_075.htm	04-Jul-2000 08:16	3k
c_074s.htm	04-Jul-2000 08:16	7k
c_074a.htm	04-Jul-2000 08:16	2k
c_074.htm	04-Jul-2000 08:16	2k
c_073.htm	04-Jul-2000 08:16	2k
c_072.htm	04-Jul-2000 08:16	2k
c_071.htm	04-Jul-2000 08:16	2k
c_070.htm	04-Jul-2000 08:16	2k
c_069.htm	04-Jul-2000 08:16	4k
c_068.htm	04-Jul-2000 08:16	2k
c_067s.htm	04-Jul-2000 08:16	6k
c_067a.htm	04-Jul-2000 08:16	2k
c_067.htm	04-Jul-2000 08:16	2k
c_066.htm	04-Jul-2000 08:16	2k

c_065.htm	04-Jul-2000 08:16	2k
c_064a.htm	04-Jul-2000 08:16	1k
c_064.htm	04-Jul-2000 08:16	2k
c_063.htm	04-Jul-2000 08:16	2k
c_062.htm	04-Jul-2000 08:16	2k
c_061a.htm	04-Jul-2000 08:16	2k
c_061.htm	04-Jul-2000 08:16	2k
c_060.htm	04-Jul-2000 08:16	2k
c_059.htm	04-Jul-2000 08:16	3k
c_058a.htm	04-Jul-2000 08:16	1k
c_058.htm	04-Jul-2000 08:16	1k
c_057.htm	04-Jul-2000 08:16	3k
c_056.htm	04-Jul-2000 08:16	2k
c_055a.htm	04-Jul-2000 08:16	7k
c_055.htm	04-Jul-2000 08:16	4k
c_054s.htm	04-Jul-2000 08:16	7k
c_054a.htm	04-Jul-2000 08:16	2k
c_054.htm	04-Jul-2000 08:16	2k
c_053a.htm	04-Jul-2000 08:16	1k
c_053.htm	04-Jul-2000 08:16	2k
c_052a.htm	04-Jul-2000 08:16	2k
c_052.htm	04-Jul-2000 08:16	2k
c_051a.htm	04-Jul-2000 08:16	1k
c_051.htm	04-Jul-2000 08:16	1k
c_050.htm	04-Jul-2000 08:16	3k

c_049a.htm	04-Jul-2000 08:16	1k
c_049.htm	04-Jul-2000 08:16	2k
c_048.htm	04-Jul-2000 08:16	2k
c_047.htm	04-Jul-2000 08:16	3k
c_046a.htm	04-Jul-2000 08:16	1k
c_046.htm	04-Jul-2000 08:16	5k
c_045a.htm	04-Jul-2000 08:16	1k
c_045.htm	04-Jul-2000 08:16	3k
c_044.htm	04-Jul-2000 08:16	3k
c_043.htm	04-Jul-2000 08:16	2k
c_042.htm	04-Jul-2000 08:16	3k
c_041s.htm	04-Jul-2000 08:16	13k
c_041a.htm	04-Jul-2000 08:16	3k
c_041.htm	04-Jul-2000 08:16	2k
c_040.htm	04-Jul-2000 08:16	2k
c_039.htm	04-Jul-2000 08:16	1k
c_038.htm	04-Jul-2000 08:16	2k
c_037a.htm	04-Jul-2000 08:16	1k
c_037.htm	04-Jul-2000 08:16	2k
c_036.htm	04-Jul-2000 08:16	2k
c_035.htm	04-Jul-2000 08:16	2k
c_034.htm	04-Jul-2000 08:16	3k
c_033a.htm	04-Jul-2000 08:16	1k
c_033.htm	04-Jul-2000 08:16	3k
c_032.htm	04-Jul-2000 08:16	3k

c_031.htm	04-Jul-2000 08:15	4k
c_030.htm	04-Jul-2000 08:15	2k
c_029s.htm	04-Jul-2000 08:15	5k
c_029a.htm	04-Jul-2000 08:15	2k
c_029.htm	04-Jul-2000 08:15	2k
c_028a.htm	04-Jul-2000 08:15	2k
c_028.htm	04-Jul-2000 08:15	3k
c_027b.htm	04-Jul-2000 08:15	3k
c_027a.htm	04-Jul-2000 08:15	2k
c_027.htm	04-Jul-2000 08:15	3k
c_026s.htm	04-Jul-2000 08:15	7k
c_026a.htm	04-Jul-2000 08:15	2k
c_026.htm	04-Jul-2000 08:15	2k
c_025.htm	04-Jul-2000 08:15	3k
c_024a.htm	04-Jul-2000 08:15	2k
c_024.htm	04-Jul-2000 08:15	4k
c_023.htm	04-Jul-2000 08:15	4k
c_022.htm	04-Jul-2000 08:15	2k
c_021s.htm	04-Jul-2000 08:15	7k
c_021a.htm	04-Jul-2000 08:15	2k
c_021.htm	04-Jul-2000 08:15	2k
c_020a.htm	04-Jul-2000 08:15	2k
c_020.htm	04-Jul-2000 08:15	2k
c_019b.htm	04-Jul-2000 08:15	1k
c_019a.htm	04-Jul-2000 08:15	4k

[c_019.htm]	04-Jul-2000 08:15	5k
[c_018.htm]	04-Jul-2000 08:15	2k
[c_017s.htm]	04-Jul-2000 08:15	10k
[c_017a.htm]	04-Jul-2000 08:15	2k
[c_017.htm]	04-Jul-2000 08:15	2k
[c_016.htm]	04-Jul-2000 08:15	4k
[c_015a.htm]	04-Jul-2000 08:15	3k
[c_015.htm]	04-Jul-2000 08:15	3k
[c_014s.htm]	04-Jul-2000 08:15	7k
[c_014a.htm]	04-Jul-2000 08:15	1k
[c_014.htm]	04-Jul-2000 08:15	2k
[c_013a.htm]	04-Jul-2000 08:15	1k
[c_013.htm]	04-Jul-2000 08:15	4k
[c_012s.htm]	04-Jul-2000 08:15	8k
[c_012a.htm]	04-Jul-2000 08:15	2k
[c_012.htm]	04-Jul-2000 08:15	2k
[c_011.htm]	04-Jul-2000 08:15	2k
[c_010a.htm]	04-Jul-2000 08:15	1k
[c_010.htm]	04-Jul-2000 08:15	6k
[c_009.htm]	04-Jul-2000 08:15	3k
[c_008.htm]	04-Jul-2000 08:15	3k
[c_007.htm]	04-Jul-2000 08:15	4k
[c_006.htm]	04-Jul-2000 08:15	4k
[c_005a.htm]	04-Jul-2000 08:15	1k
[c_005.htm]	04-Jul-2000 08:15	3k

c_004.htm	04-Jul-2000 08:15	3k
c_003a.htm	04-Jul-2000 08:15	1k
c_003.htm	04-Jul-2000 08:15	2k
c_002.htm	04-Jul-2000 08:15	4k
c_001a.htm	04-Jul-2000 08:15	1k
c_001.htm	04-Jul-2000 08:15	2k
c_000a.htm	04-Jul-2000 08:15	2k
c_000.htm	04-Jul-2000 08:15	4k
advcw3.htm	04-Jul-2000 08:15	43k
advcw2.htm	04-Jul-2000 08:15	35k
advcw1.htm	04-Jul-2000 08:15	31k

Apache/1.3.12 Server at www.ibilce.unesp.br Port 80

Index of /courseware/cprogram/images

Name	Last modified	Size	Description
Parent Directory	24-May-2000 10:59	-	
previous.gif	04-Jul-2000 08:15	2k	
next.gif	04-Jul-2000 08:15	2k	
menu.gif	04-Jul-2000 08:15	2k	
mail.gif	04-Jul-2000 08:15	4k	
for_loop.rm	04-Jul-2000 08:15	740k	
for_loop.jpg	04-Jul-2000 08:15	12k	
for_loop.avi	04-Jul-2000 08:15	4.3M	
for6.gif	04-Jul-2000 08:15	6k	
for5.gif	04-Jul-2000 08:15	6k	
for4.gif	04-Jul-2000 08:15	6k	
for3.gif	04-Jul-2000 08:15	6k	
for2.gif	04-Jul-2000 08:15	6k	
for1.gif	04-Jul-2000 08:15	6k	
disk1.jpg	04-Jul-2000 08:15	1k	
cam1.jpg	04-Jul-2000 08:15	1k	
c_019.gif	04-Jul-2000 08:15	2k	
ballrem.gif	04-Jul-2000 08:15	1k	
ballloc.gif	04-Jul-2000 08:15	1k	

Apache/1.3.12 Server at www.ibilce.unesp.br Port 80

C Programming, v2.7

© Copyright Brian Brown, 1984-2000. All rights reserved.

Notes | Tests | Portuguese Translation | Other Courses

Order the CD-ROM and support this initiative!

Please use a Mirror Site near you

Austria	http://gd.tuwien.ac.at/languages/c/programming-bbrown/
Australia	Northern Territory University
Belgium	Université libre de Bruxelles (Free University of Brussels)
Brazil	Sao Paulo State University, Rio Pretol
Canada	RCC College of Technology - Concord (Toronto), Canada.
Finland	Vantaa Institute of Technology
Macau	Inter-University Institute of Macau
Italy	Università Ca' Foscari di Venezia
New Zealand	Central Institute of Technology
USA	University of Nebraska at Kearney
USA	http://homepages.msn.com/LibraryLawn/brownbr/cprogram/default.htm
USA	Northern Mitchigan University
USA	http://www.xploiter.com/mirrors/cprogram

About this study guide

This is a free study guide on C language programming. It will show you how to write programs in the C language. Learn how to program with this free education guide online course that is free on the Internet. Train yourself using this course and program your own computer today!

The tutorial is **free**, but you can also purchase the CD-ROM with many other of our on-line modules, and is packed with heaps of examples, inter-active tests, and a complete set of notes. This self study guide is not a definitive guide, but is designed to help you program quickly and effectively.

Getting started: A brief introduction to what is needed to program in C

Internet Based Resource Material/Reference Books

List of C resources and tutorials on the Internet

C FAQ

Comprehensive listing of interrupts and hardware details.

Personal C compiler. Fully functional, C WARE
GNU C/C++ Compiler Homepage [MSDOS port]
List of free Compilers
Searchable index of Available Compilers on the Internet

Restrictions on Use

You are free to reference this material on-line and print out copies for personal use. You may not copy it, distribute it, or make it available over the Internet or reproduce it in any other form, printed or electronic for others to access without permission from the author. Educators are free to store the files locally on their own network and to print as many copies as needed for their students, as long as the original copyright remains intact. Mirroring of this learning guide is available for Universities free of charge. Please order the CD-ROM to obtain **ALL** the latest versions of the on-line courses and support this free on-line learning initiative.

C Programming, v2.7

© Copyright Brian Brown, 1984-2000. All rights reserved.

Information about Programming in C or Pascal

Both C and Pascal are considered High Level Languages. They use English type statements that are converted to machine statements which are executed by computers.

C and Pascal programs are simple text files containing program statements. As such, they are created using a text editor. This is called the source program.

Once you have written the program, it must be compiled. The source program is compiled by a special program called a compiler, whose task is to convert the statements in the source code to either an intermediate format (called an object file), or an executable format which can be run on the computer.

There are a number of free compilers available on the Internet. Check the links on the C or Pascal programming pages for places where you might find a free compiler.

File Extension	Type of File
.c	C Source file, created using a text editor, contains source program statements
.pas	Pascal Source file, created using a text editor, contains source program statements
.obj	Object file, intermediate file generated by the compiler
.exe	Executable program generated at the end of the compiler and linking stages, which can be run on the computer
.lib	Library routines, for things like printing and reading the keyboard, they come with the compiler and are combined with .obj files to generate an .exe program

We do not provide compilers with this course. We recommend the use of a simple DOS based compiler in order to learn programming. Turbo Pascal and Turbo C for DOS are both good easy to use compilers. They may still be available in computer stores.

Borland has been kindly providing free downloads of a few of the older version compilers for Turbo Pascal and Turbo C. To access copies of these compilers one must first join Borland's community (it's free) and then go to their museum web page. Community Member Login page is at:
http://wwwapp1.borland.com/login/login.exe
Click the New User button to join.

Index of /courseware/cprogram

Name	Last modified	Size	Description
Parent Directory	04-Jul-2000 08:54	-	
advcw1.htm	04-Jul-2000 08:15	31k	
advcw2.htm	04-Jul-2000 08:15	35k	
advcw3.htm	04-Jul-2000 08:15	43k	
c_000.htm	04-Jul-2000 08:15	4k	
c_000a.htm	04-Jul-2000 08:15	2k	
c_001.htm	04-Jul-2000 08:15	2k	
c_001a.htm	04-Jul-2000 08:15	1k	
c_002.htm	04-Jul-2000 08:15	4k	
c_003.htm	04-Jul-2000 08:15	2k	
c_003a.htm	04-Jul-2000 08:15	1k	
c_004.htm	04-Jul-2000 08:15	3k	
c_005.htm	04-Jul-2000 08:15	3k	
c_005a.htm	04-Jul-2000 08:15	1k	
c_006.htm	04-Jul-2000 08:15	4k	
c_007.htm	04-Jul-2000 08:15	4k	
c_008.htm	04-Jul-2000 08:15	3k	
c_009.htm	04-Jul-2000 08:15	3k	
c_010.htm	04-Jul-2000 08:15	6k	
c_010a.htm	04-Jul-2000 08:15	1k	
c_011.htm	04-Jul-2000 08:15	2k	

c_012.htm	04-Jul-2000 08:15	2k
c_012a.htm	04-Jul-2000 08:15	2k
c_012s.htm	04-Jul-2000 08:15	8k
c_013.htm	04-Jul-2000 08:15	4k
c_013a.htm	04-Jul-2000 08:15	1k
c_014.htm	04-Jul-2000 08:15	2k
c_014a.htm	04-Jul-2000 08:15	1k
c_014s.htm	04-Jul-2000 08:15	7k
c_015.htm	04-Jul-2000 08:15	3k
c_015a.htm	04-Jul-2000 08:15	3k
c_016.htm	04-Jul-2000 08:15	4k
c_017.htm	04-Jul-2000 08:15	2k
c_017a.htm	04-Jul-2000 08:15	2k
c_017s.htm	04-Jul-2000 08:15	10k
c_018.htm	04-Jul-2000 08:15	2k
c_019.htm	04-Jul-2000 08:15	5k
c_019a.htm	04-Jul-2000 08:15	4k
c_019b.htm	04-Jul-2000 08:15	1k
c_020.htm	04-Jul-2000 08:15	2k
c_020a.htm	04-Jul-2000 08:15	2k
c_021.htm	04-Jul-2000 08:15	2k
c_021a.htm	04-Jul-2000 08:15	2k
c_021s.htm	04-Jul-2000 08:15	7k
c_022.htm	04-Jul-2000 08:15	2k
c_023.htm	04-Jul-2000 08:15	4k

c_024.htm	04-Jul-2000 08:15	4k
c_024a.htm	04-Jul-2000 08:15	2k
c_025.htm	04-Jul-2000 08:15	3k
c_026.htm	04-Jul-2000 08:15	2k
c_026a.htm	04-Jul-2000 08:15	2k
c_026s.htm	04-Jul-2000 08:15	7k
c_027.htm	04-Jul-2000 08:15	3k
c_027a.htm	04-Jul-2000 08:15	2k
c_027b.htm	04-Jul-2000 08:15	3k
c_028.htm	04-Jul-2000 08:15	3k
c_028a.htm	04-Jul-2000 08:15	2k
c_029.htm	04-Jul-2000 08:15	2k
c_029a.htm	04-Jul-2000 08:15	2k
c_029s.htm	04-Jul-2000 08:15	5k
c_030.htm	04-Jul-2000 08:15	2k
c_031.htm	04-Jul-2000 08:15	4k
c_032.htm	04-Jul-2000 08:16	3k
c_033.htm	04-Jul-2000 08:16	3k
c_033a.htm	04-Jul-2000 08:16	1k
c_034.htm	04-Jul-2000 08:16	3k
c_035.htm	04-Jul-2000 08:16	2k
c_036.htm	04-Jul-2000 08:16	2k
c_037.htm	04-Jul-2000 08:16	2k
c_037a.htm	04-Jul-2000 08:16	1k
c_038.htm	04-Jul-2000 08:16	2k

c_039.htm	04-Jul-2000 08:16	1k
c_040.htm	04-Jul-2000 08:16	2k
c_041.htm	04-Jul-2000 08:16	2k
c_041a.htm	04-Jul-2000 08:16	3k
c_041s.htm	04-Jul-2000 08:16	13k
c_042.htm	04-Jul-2000 08:16	3k
c_043.htm	04-Jul-2000 08:16	2k
c_044.htm	04-Jul-2000 08:16	3k
c_045.htm	04-Jul-2000 08:16	3k
c_045a.htm	04-Jul-2000 08:16	1k
c_046.htm	04-Jul-2000 08:16	5k
c_046a.htm	04-Jul-2000 08:16	1k
c_047.htm	04-Jul-2000 08:16	3k
c_048.htm	04-Jul-2000 08:16	2k
c_049.htm	04-Jul-2000 08:16	2k
c_049a.htm	04-Jul-2000 08:16	1k
c_050.htm	04-Jul-2000 08:16	3k
c_051.htm	04-Jul-2000 08:16	1k
c_051a.htm	04-Jul-2000 08:16	1k
c_052.htm	04-Jul-2000 08:16	2k
c_052a.htm	04-Jul-2000 08:16	2k
c_053.htm	04-Jul-2000 08:16	2k
c_053a.htm	04-Jul-2000 08:16	1k
c_054.htm	04-Jul-2000 08:16	2k
c_054a.htm	04-Jul-2000 08:16	2k

c_054s.htm	04-Jul-2000 08:16	7k
c_055.htm	04-Jul-2000 08:16	4k
c_055a.htm	04-Jul-2000 08:16	7k
c_056.htm	04-Jul-2000 08:16	2k
c_057.htm	04-Jul-2000 08:16	3k
c_058.htm	04-Jul-2000 08:16	1k
c_058a.htm	04-Jul-2000 08:16	1k
c_059.htm	04-Jul-2000 08:16	3k
c_060.htm	04-Jul-2000 08:16	2k
c_061.htm	04-Jul-2000 08:16	2k
c_061a.htm	04-Jul-2000 08:16	2k
c_062.htm	04-Jul-2000 08:16	2k
c_063.htm	04-Jul-2000 08:16	2k
c_064.htm	04-Jul-2000 08:16	2k
c_064a.htm	04-Jul-2000 08:16	1k
c_065.htm	04-Jul-2000 08:16	2k
c_066.htm	04-Jul-2000 08:16	2k
c_067.htm	04-Jul-2000 08:16	2k
c_067a.htm	04-Jul-2000 08:16	2k
c_067s.htm	04-Jul-2000 08:16	6k
c_068.htm	04-Jul-2000 08:16	2k
c_069.htm	04-Jul-2000 08:16	4k
c_070.htm	04-Jul-2000 08:16	2k
c_071.htm	04-Jul-2000 08:16	2k
c_072.htm	04-Jul-2000 08:16	2k

c_073.htm	04-Jul-2000 08:16	2k
c_074.htm	04-Jul-2000 08:16	2k
c_074a.htm	04-Jul-2000 08:16	2k
c_074s.htm	04-Jul-2000 08:16	7k
c_075.htm	04-Jul-2000 08:16	3k
c_075a.htm	04-Jul-2000 08:16	5k
c_076.htm	04-Jul-2000 08:16	3k
c_077.htm	04-Jul-2000 08:16	2k
c_0771.htm	04-Jul-2000 08:16	2k
c_0771a.htm	04-Jul-2000 08:16	1k
c_077a.htm	04-Jul-2000 08:16	2k
c_077s.htm	04-Jul-2000 08:16	6k
c_078.htm	04-Jul-2000 08:16	2k
c_079.htm	04-Jul-2000 08:16	2k
c_080.htm	04-Jul-2000 08:16	2k
c_081.htm	04-Jul-2000 08:16	2k
c_081a.htm	04-Jul-2000 08:16	3k
c_081s.htm	04-Jul-2000 08:16	9k
c_082.htm	04-Jul-2000 08:16	2k
c_082a.htm	04-Jul-2000 08:16	2k
c_083.htm	04-Jul-2000 08:16	2k
c_083a.htm	04-Jul-2000 08:16	1k
c_084.htm	04-Jul-2000 08:16	2k
c_084a.htm	04-Jul-2000 08:16	2k
c_085.htm	04-Jul-2000 08:16	5k

c_085a.htm	04-Jul-2000 08:16	1k
c_086.htm	04-Jul-2000 08:16	3k
c_087.htm	04-Jul-2000 08:16	2k
c_088.htm	04-Jul-2000 08:16	3k
c_088a.htm	04-Jul-2000 08:16	4k
c_089.htm	04-Jul-2000 08:16	2k
c_090.htm	04-Jul-2000 08:16	1k
c_091.htm	04-Jul-2000 08:16	2k
c_092.htm	04-Jul-2000 08:16	2k
c_093.htm	04-Jul-2000 08:16	5k
c_094.htm	04-Jul-2000 08:16	11k
c_095.htm	04-Jul-2000 08:16	2k
c_096.htm	04-Jul-2000 08:16	1k
c_097.htm	04-Jul-2000 08:16	1k
c_098.htm	04-Jul-2000 08:16	2k
c_099.htm	04-Jul-2000 08:16	2k
c_100.htm	04-Jul-2000 08:16	2k
c_101.htm	04-Jul-2000 08:16	1k
c_102.htm	04-Jul-2000 08:16	2k
c_103.htm	04-Jul-2000 08:16	2k
c_104.htm	04-Jul-2000 08:16	2k
c_105.htm	04-Jul-2000 08:16	2k
c_105a.htm	04-Jul-2000 08:16	2k
c_106.htm	04-Jul-2000 08:16	2k
c_107.htm	04-Jul-2000 08:16	4k

c_108.htm	04-Jul-2000 08:16	2k
c_109.htm	04-Jul-2000 08:16	1k
c_110.htm	04-Jul-2000 08:16	20k
cstart.htm	04-Jul-2000 08:16	3k
default.htm	04-Jul-2000 08:16	7k
howtopro.htm	04-Jul-2000 08:16	4k
images/	03-Apr-2000 12:52	-
joystick.htm	04-Jul-2000 08:16	5k
onlinet.htm	04-Jul-2000 08:16	2k
problems.htm	04-Jul-2000 08:16	11k
simple4.htm	04-Jul-2000 08:16	2k

Apache/1.3.12 Server at www.ibilce.unesp.br Port 80

Index of /courseware/cprogram/images

Name	Last modified	Size	Description
Parent Directory	24-May-2000 10:59	-	
ballloc.gif	04-Jul-2000 08:15	1k	
ballrem.gif	04-Jul-2000 08:15	1k	
c_019.gif	04-Jul-2000 08:15	2k	
cam1.jpg	04-Jul-2000 08:15	1k	
disk1.jpg	04-Jul-2000 08:15	1k	
for1.gif	04-Jul-2000 08:15	6k	
for2.gif	04-Jul-2000 08:15	6k	
for3.gif	04-Jul-2000 08:15	6k	
for4.gif	04-Jul-2000 08:15	6k	
for5.gif	04-Jul-2000 08:15	6k	
for6.gif	04-Jul-2000 08:15	6k	
for_loop.avi	04-Jul-2000 08:15	4.3M	
for_loop.jpg	04-Jul-2000 08:15	12k	
for_loop.rm	04-Jul-2000 08:15	740k	
mail.gif	04-Jul-2000 08:15	4k	
menu.gif	04-Jul-2000 08:15	2k	
next.gif	04-Jul-2000 08:15	2k	
previous.gif	04-Jul-2000 08:15	2k	

Apache/1.3.12 Server at www.ibilce.unesp.br Port 80

Advanced C, Joystick Interfacing

INDEX Previous

```
From: ajmyrvold@violet.waterloo.edu (Alan Myrvold)
Newsgroups: comp.sys.ibm.pc
Subject: Joystick routines
Date: 24 Mar 88 07:35:25 GMT
Keywords: Game Port, Joy Stick
How to read the joystick on an IBM PC.

This program, written for Turbo C, version 1.0 shows how the joystick
postion may be read. I am indebted to uunet!netxcom!jallen (John Allen) for
answering my previous posting on the subject and providing technical
assistance.
Text from his EMAIL to me is included in the program.
-------------------------------------------------------------------
Alan Myrvold     ajmyrvold@violet.waterloo.edu
-------------------------------------------------------------------


----------------------------- JOY.C    ----------------------------
/* Demonstration of reading joy stick postition
    Written March 24, 1988
    Written by : ajmyrvold@violet.waterloo.edu (Alan J. Myrvold)
Technical assistance from : uunet!netxcom!jallen (John Allen)
Turbo C Version 1.0
*/

#include <stdio.h>
#include <dos.h>
#include <bios.h>

typedef struct {
    int sw1,sw2;
    int x,y;
    int cenx,ceny;
    } joy_stick;

joy_stick joy;

#define keypressed (bioskey(1) != 0)
#define kb_clear() while keypressed bioskey(0);

void GotoXY(int x,int y)
{
    union REGS r;
    /* Set XY position */
```

```
    r.h.ah = 2;
    r.h.bh = 0;             /* Assume Video Page 0 */
    r.h.dh = (char) y;
    r.h.dl = (char) x;
    int86(16,&r,&r);
}

void ClrScr()
{
    union REGS r;

    /* Get video mode */
    r.h.ah = 15;
    int86(16,&r,&r);
    /* Set video mode */
    r.h.ah = 0;
    int86(16,&r,&r);
}


/*
From: uunet!netxcom!jallen (John Allen)
1.  Trigger the joystick oneshots with an 'out' to 0x201.
    This will set all of the joystick bits on.

2.  Read (in) 0x201, finding:

        Bit             Contents
        0               Joystick A X coordinate
        1               Joystick A Y coordinate
        2               Joystick B X coordinate
        3               Joystick B Y coordinate
        4               Button A 1
        5               Button A 2
        6               Button B 1
        7               Button B 2

3.  Continue reading 0x201 until all oneshots return to zero,
    recording the loop during which each bit falls to zero.
    The duration of the pulse from each oneshot may be used to
    determine the resistive load (from 0 to 100K) from each
    Joystick, as: Time = 24.2msec. + .011 (r) msec.

4.  To do this correctly, I recommend calibrating the joystick;
    have the user move the stick to each corner, then center it,
    while recording the resulting values.

*/
```

```
void disp_stick(int line,joy_stick *joy)
{
    GotoXY(0,line);
    printf("sw1 %d sw2 %d",joy -> sw1,joy -> sw2);
    GotoXY(0,line+1);
    printf("x %4d y %4d",joy -> x,joy -> y);
}

void read_stick(int stick,joy_stick *joy)
{
    int k,jx,jy;
    int c,m1,m2,m3,m4,m5;

    /* Define masks for the chosen joystick */
    if (stick == 1) m4 = 1; else
    if (stick == 2) m4 = 4; else
        printf("Invalid stick %d\n",stick);
    m5 = m4 << 1;
    m1 = m4 << 4;
    m2 = m5 << 4;
    m3 = m4 + m5;

    /* Trigger joystick */
    outportb(0x201,0xff);
    c = inportb(0x201);
    /* Read switch settings */
    joy -> sw1 = (c & m1) == 0;
    joy -> sw2 = (c & m2) == 0;
    /* Get X and Y positions */
    for (k = 0; (c & m3) != 0; k++) {
        if ((c & m4) != 0) jx = k;
        if ((c & m5) != 0) jy = k;
        c = inportb(0x201);
    }
    joy -> x = jx - (joy -> cenx);
    joy -> y = jy - (joy -> ceny);
}

int choose_stick(joy_stick *joy)
{
    int init_swa,init_swb,swa,swb;
    int c,retval;

    printf("Center joystick and press fire, or press any key\n");
    kb_clear();
    outportb(0x201,0xff);
    c = inportb(0x201);
    init_swa = c & 0x30;
    init_swb = c & 0xc0;
```

```
    do {
       outportb(0x201,0xff);
       c = inportb(0x201);
       swa = c & 0x30;
       swb = c & 0xc0;
    } while ((swa == init_swa) && (swb == init_swb) && !keypressed);
        if (swa != init_swa) {
       printf("Joystick 1 selected\n");
       retval = 1;
    } else if (swb != init_swb) {
       printf("Joystick 2 selected\n");
       retval = 2;
    } else {
       printf("Keyboard selected\n");
       kb_clear();
       retval = 0;
    }

    if (retval != 0) { /* Determine Center */
       joy -> cenx = joy -> ceny = 0;
       read_stick(retval,joy);
       joy -> cenx = joy -> x;
       joy -> ceny = joy -> y;
    }
    return(retval);
}

main()
{
   int k;

   k = choose_stick(&joy);
   ClrScr();
   if (k != 0) while (!keypressed) {
        read_stick(k,&joy);
        disp_stick(0,&joy);
   }
}
```

Previous

Index of /courseware/cprogram

Name	Last modified	Size	Description
Parent Directory	04-Jul-2000 08:54	-	
images/	03-Apr-2000 12:52	-	
advcw1.htm	04-Jul-2000 08:15	31k	
advcw2.htm	04-Jul-2000 08:15	35k	
advcw3.htm	04-Jul-2000 08:15	43k	
c_000.htm	04-Jul-2000 08:15	4k	
c_000a.htm	04-Jul-2000 08:15	2k	
c_001.htm	04-Jul-2000 08:15	2k	
c_001a.htm	04-Jul-2000 08:15	1k	
c_002.htm	04-Jul-2000 08:15	4k	
c_003.htm	04-Jul-2000 08:15	2k	
c_003a.htm	04-Jul-2000 08:15	1k	
c_004.htm	04-Jul-2000 08:15	3k	
c_005.htm	04-Jul-2000 08:15	3k	
c_005a.htm	04-Jul-2000 08:15	1k	
c_006.htm	04-Jul-2000 08:15	4k	
c_007.htm	04-Jul-2000 08:15	4k	
c_008.htm	04-Jul-2000 08:15	3k	
c_009.htm	04-Jul-2000 08:15	3k	
c_010.htm	04-Jul-2000 08:15	6k	
c_010a.htm	04-Jul-2000 08:15	1k	

c_011.htm	04-Jul-2000 08:15	2k
c_012.htm	04-Jul-2000 08:15	2k
c_012a.htm	04-Jul-2000 08:15	2k
c_012s.htm	04-Jul-2000 08:15	8k
c_013.htm	04-Jul-2000 08:15	4k
c_013a.htm	04-Jul-2000 08:15	1k
c_014.htm	04-Jul-2000 08:15	2k
c_014a.htm	04-Jul-2000 08:15	1k
c_014s.htm	04-Jul-2000 08:15	7k
c_015.htm	04-Jul-2000 08:15	3k
c_015a.htm	04-Jul-2000 08:15	3k
c_016.htm	04-Jul-2000 08:15	4k
c_017.htm	04-Jul-2000 08:15	2k
c_017a.htm	04-Jul-2000 08:15	2k
c_017s.htm	04-Jul-2000 08:15	10k
c_018.htm	04-Jul-2000 08:15	2k
c_019.htm	04-Jul-2000 08:15	5k
c_019a.htm	04-Jul-2000 08:15	4k
c_019b.htm	04-Jul-2000 08:15	1k
c_020.htm	04-Jul-2000 08:15	2k
c_020a.htm	04-Jul-2000 08:15	2k
c_021.htm	04-Jul-2000 08:15	2k
c_021a.htm	04-Jul-2000 08:15	2k
c_021s.htm	04-Jul-2000 08:15	7k
c_022.htm	04-Jul-2000 08:15	2k

c_023.htm	04-Jul-2000 08:15	4k
c_024.htm	04-Jul-2000 08:15	4k
c_024a.htm	04-Jul-2000 08:15	2k
c_025.htm	04-Jul-2000 08:15	3k
c_026.htm	04-Jul-2000 08:15	2k
c_026a.htm	04-Jul-2000 08:15	2k
c_026s.htm	04-Jul-2000 08:15	7k
c_027.htm	04-Jul-2000 08:15	3k
c_027a.htm	04-Jul-2000 08:15	2k
c_027b.htm	04-Jul-2000 08:15	3k
c_028.htm	04-Jul-2000 08:15	3k
c_028a.htm	04-Jul-2000 08:15	2k
c_029.htm	04-Jul-2000 08:15	2k
c_029a.htm	04-Jul-2000 08:15	2k
c_029s.htm	04-Jul-2000 08:15	5k
c_030.htm	04-Jul-2000 08:15	2k
c_031.htm	04-Jul-2000 08:15	4k
c_032.htm	04-Jul-2000 08:16	3k
c_033.htm	04-Jul-2000 08:16	3k
c_033a.htm	04-Jul-2000 08:16	1k
c_034.htm	04-Jul-2000 08:16	3k
c_035.htm	04-Jul-2000 08:16	2k
c_036.htm	04-Jul-2000 08:16	2k
c_037.htm	04-Jul-2000 08:16	2k
c_037a.htm	04-Jul-2000 08:16	1k

c_038.htm	04-Jul-2000 08:16	2k
c_039.htm	04-Jul-2000 08:16	1k
c_040.htm	04-Jul-2000 08:16	2k
c_041.htm	04-Jul-2000 08:16	2k
c_041a.htm	04-Jul-2000 08:16	3k
c_041s.htm	04-Jul-2000 08:16	13k
c_042.htm	04-Jul-2000 08:16	3k
c_043.htm	04-Jul-2000 08:16	2k
c_044.htm	04-Jul-2000 08:16	3k
c_045.htm	04-Jul-2000 08:16	3k
c_045a.htm	04-Jul-2000 08:16	1k
c_046.htm	04-Jul-2000 08:16	5k
c_046a.htm	04-Jul-2000 08:16	1k
c_047.htm	04-Jul-2000 08:16	3k
c_048.htm	04-Jul-2000 08:16	2k
c_049.htm	04-Jul-2000 08:16	2k
c_049a.htm	04-Jul-2000 08:16	1k
c_050.htm	04-Jul-2000 08:16	3k
c_051.htm	04-Jul-2000 08:16	1k
c_051a.htm	04-Jul-2000 08:16	1k
c_052.htm	04-Jul-2000 08:16	2k
c_052a.htm	04-Jul-2000 08:16	2k
c_053.htm	04-Jul-2000 08:16	2k
c_053a.htm	04-Jul-2000 08:16	1k
c_054.htm	04-Jul-2000 08:16	2k

c_054a.htm	04-Jul-2000 08:16	2k
c_054s.htm	04-Jul-2000 08:16	7k
c_055.htm	04-Jul-2000 08:16	4k
c_055a.htm	04-Jul-2000 08:16	7k
c_056.htm	04-Jul-2000 08:16	2k
c_057.htm	04-Jul-2000 08:16	3k
c_058.htm	04-Jul-2000 08:16	1k
c_058a.htm	04-Jul-2000 08:16	1k
c_059.htm	04-Jul-2000 08:16	3k
c_060.htm	04-Jul-2000 08:16	2k
c_061.htm	04-Jul-2000 08:16	2k
c_061a.htm	04-Jul-2000 08:16	2k
c_062.htm	04-Jul-2000 08:16	2k
c_063.htm	04-Jul-2000 08:16	2k
c_064.htm	04-Jul-2000 08:16	2k
c_064a.htm	04-Jul-2000 08:16	1k
c_065.htm	04-Jul-2000 08:16	2k
c_066.htm	04-Jul-2000 08:16	2k
c_067.htm	04-Jul-2000 08:16	2k
c_067a.htm	04-Jul-2000 08:16	2k
c_067s.htm	04-Jul-2000 08:16	6k
c_068.htm	04-Jul-2000 08:16	2k
c_069.htm	04-Jul-2000 08:16	4k
c_070.htm	04-Jul-2000 08:16	2k
c_071.htm	04-Jul-2000 08:16	2k

c_072.htm	04-Jul-2000 08:16	2k
c_073.htm	04-Jul-2000 08:16	2k
c_074.htm	04-Jul-2000 08:16	2k
c_074a.htm	04-Jul-2000 08:16	2k
c_074s.htm	04-Jul-2000 08:16	7k
c_075.htm	04-Jul-2000 08:16	3k
c_075a.htm	04-Jul-2000 08:16	5k
c_076.htm	04-Jul-2000 08:16	3k
c_077.htm	04-Jul-2000 08:16	2k
c_0771.htm	04-Jul-2000 08:16	2k
c_0771a.htm	04-Jul-2000 08:16	1k
c_077a.htm	04-Jul-2000 08:16	2k
c_077s.htm	04-Jul-2000 08:16	6k
c_078.htm	04-Jul-2000 08:16	2k
c_079.htm	04-Jul-2000 08:16	2k
c_080.htm	04-Jul-2000 08:16	2k
c_081.htm	04-Jul-2000 08:16	2k
c_081a.htm	04-Jul-2000 08:16	3k
c_081s.htm	04-Jul-2000 08:16	9k
c_082.htm	04-Jul-2000 08:16	2k
c_082a.htm	04-Jul-2000 08:16	2k
c_083.htm	04-Jul-2000 08:16	2k
c_083a.htm	04-Jul-2000 08:16	1k
c_084.htm	04-Jul-2000 08:16	2k
c_084a.htm	04-Jul-2000 08:16	2k

[c_085.htm]	04-Jul-2000 08:16	5k
[c_085a.htm]	04-Jul-2000 08:16	1k
[c_086.htm]	04-Jul-2000 08:16	3k
[c_087.htm]	04-Jul-2000 08:16	2k
[c_088.htm]	04-Jul-2000 08:16	3k
[c_088a.htm]	04-Jul-2000 08:16	4k
[c_089.htm]	04-Jul-2000 08:16	2k
[c_090.htm]	04-Jul-2000 08:16	1k
[c_091.htm]	04-Jul-2000 08:16	2k
[c_092.htm]	04-Jul-2000 08:16	2k
[c_093.htm]	04-Jul-2000 08:16	5k
[c_094.htm]	04-Jul-2000 08:16	11k
[c_095.htm]	04-Jul-2000 08:16	2k
[c_096.htm]	04-Jul-2000 08:16	1k
[c_097.htm]	04-Jul-2000 08:16	1k
[c_098.htm]	04-Jul-2000 08:16	2k
[c_099.htm]	04-Jul-2000 08:16	2k
[c_100.htm]	04-Jul-2000 08:16	2k
[c_101.htm]	04-Jul-2000 08:16	1k
[c_102.htm]	04-Jul-2000 08:16	2k
[c_103.htm]	04-Jul-2000 08:16	2k
[c_104.htm]	04-Jul-2000 08:16	2k
[c_105.htm]	04-Jul-2000 08:16	2k
[c_105a.htm]	04-Jul-2000 08:16	2k
[c_106.htm]	04-Jul-2000 08:16	2k

c_107.htm	04-Jul-2000 08:16	4k
c_108.htm	04-Jul-2000 08:16	2k
c_109.htm	04-Jul-2000 08:16	1k
c_110.htm	04-Jul-2000 08:16	20k
cstart.htm	04-Jul-2000 08:16	3k
default.htm	04-Jul-2000 08:16	7k
howtopro.htm	04-Jul-2000 08:16	4k
joystick.htm	04-Jul-2000 08:16	5k
onlinet.htm	04-Jul-2000 08:16	2k
problems.htm	04-Jul-2000 08:16	11k
simple4.htm	04-Jul-2000 08:16	2k

Apache/1.3.12 Server at www.ibilce.unesp.br Port 80

Index of /courseware/cprogram/images

Name	Last modified	Size	Description
Parent Directory	24-May-2000 10:59	-	
ballloc.gif	04-Jul-2000 08:15	1k	
ballrem.gif	04-Jul-2000 08:15	1k	
c_019.gif	04-Jul-2000 08:15	2k	
cam1.jpg	04-Jul-2000 08:15	1k	
disk1.jpg	04-Jul-2000 08:15	1k	
for1.gif	04-Jul-2000 08:15	6k	
for2.gif	04-Jul-2000 08:15	6k	
for3.gif	04-Jul-2000 08:15	6k	
for4.gif	04-Jul-2000 08:15	6k	
for5.gif	04-Jul-2000 08:15	6k	
for6.gif	04-Jul-2000 08:15	6k	
for_loop.avi	04-Jul-2000 08:15	4.3M	
for_loop.jpg	04-Jul-2000 08:15	12k	
for_loop.rm	04-Jul-2000 08:15	740k	
mail.gif	04-Jul-2000 08:15	4k	
menu.gif	04-Jul-2000 08:15	2k	
next.gif	04-Jul-2000 08:15	2k	
previous.gif	04-Jul-2000 08:15	2k	

Apache/1.3.12 Server at www.ibilce.unesp.br Port 80

Index of /courseware/cprogram

Name	Last modified	Size	Description
Parent Directory	04-Jul-2000 08:54	-	
simple4.htm	04-Jul-2000 08:16	2k	
problems.htm	04-Jul-2000 08:16	11k	
onlinet.htm	04-Jul-2000 08:16	2k	
joystick.htm	04-Jul-2000 08:16	5k	
howtopro.htm	04-Jul-2000 08:16	4k	
default.htm	04-Jul-2000 08:16	7k	
cstart.htm	04-Jul-2000 08:16	3k	
c_110.htm	04-Jul-2000 08:16	20k	
c_109.htm	04-Jul-2000 08:16	1k	
c_108.htm	04-Jul-2000 08:16	2k	
c_107.htm	04-Jul-2000 08:16	4k	
c_106.htm	04-Jul-2000 08:16	2k	
c_105a.htm	04-Jul-2000 08:16	2k	
c_105.htm	04-Jul-2000 08:16	2k	
c_104.htm	04-Jul-2000 08:16	2k	
c_103.htm	04-Jul-2000 08:16	2k	
c_102.htm	04-Jul-2000 08:16	2k	
c_101.htm	04-Jul-2000 08:16	1k	
c_100.htm	04-Jul-2000 08:16	2k	
c_099.htm	04-Jul-2000 08:16	2k	

[c_098.htm]	04-Jul-2000 08:16	2k
[c_097.htm]	04-Jul-2000 08:16	1k
[c_096.htm]	04-Jul-2000 08:16	1k
[c_095.htm]	04-Jul-2000 08:16	2k
[c_094.htm]	04-Jul-2000 08:16	11k
[c_093.htm]	04-Jul-2000 08:16	5k
[c_092.htm]	04-Jul-2000 08:16	2k
[c_091.htm]	04-Jul-2000 08:16	2k
[c_090.htm]	04-Jul-2000 08:16	1k
[c_089.htm]	04-Jul-2000 08:16	2k
[c_088a.htm]	04-Jul-2000 08:16	4k
[c_088.htm]	04-Jul-2000 08:16	3k
[c_087.htm]	04-Jul-2000 08:16	2k
[c_086.htm]	04-Jul-2000 08:16	3k
[c_085a.htm]	04-Jul-2000 08:16	1k
[c_085.htm]	04-Jul-2000 08:16	5k
[c_084a.htm]	04-Jul-2000 08:16	2k
[c_084.htm]	04-Jul-2000 08:16	2k
[c_083a.htm]	04-Jul-2000 08:16	1k
[c_083.htm]	04-Jul-2000 08:16	2k
[c_082a.htm]	04-Jul-2000 08:16	2k
[c_082.htm]	04-Jul-2000 08:16	2k
[c_081s.htm]	04-Jul-2000 08:16	9k
[c_081a.htm]	04-Jul-2000 08:16	3k
[c_081.htm]	04-Jul-2000 08:16	2k

c_080.htm	04-Jul-2000 08:16	2k
c_079.htm	04-Jul-2000 08:16	2k
c_078.htm	04-Jul-2000 08:16	2k
c_077s.htm	04-Jul-2000 08:16	6k
c_077a.htm	04-Jul-2000 08:16	2k
c_0771a.htm	04-Jul-2000 08:16	1k
c_0771.htm	04-Jul-2000 08:16	2k
c_077.htm	04-Jul-2000 08:16	2k
c_076.htm	04-Jul-2000 08:16	3k
c_075a.htm	04-Jul-2000 08:16	5k
c_075.htm	04-Jul-2000 08:16	3k
c_074s.htm	04-Jul-2000 08:16	7k
c_074a.htm	04-Jul-2000 08:16	2k
c_074.htm	04-Jul-2000 08:16	2k
c_073.htm	04-Jul-2000 08:16	2k
c_072.htm	04-Jul-2000 08:16	2k
c_071.htm	04-Jul-2000 08:16	2k
c_070.htm	04-Jul-2000 08:16	2k
c_069.htm	04-Jul-2000 08:16	4k
c_068.htm	04-Jul-2000 08:16	2k
c_067s.htm	04-Jul-2000 08:16	6k
c_067a.htm	04-Jul-2000 08:16	2k
c_067.htm	04-Jul-2000 08:16	2k
c_066.htm	04-Jul-2000 08:16	2k
c_065.htm	04-Jul-2000 08:16	2k

c_064a.htm	04-Jul-2000 08:16	1k
c_064.htm	04-Jul-2000 08:16	2k
c_063.htm	04-Jul-2000 08:16	2k
c_062.htm	04-Jul-2000 08:16	2k
c_061a.htm	04-Jul-2000 08:16	2k
c_061.htm	04-Jul-2000 08:16	2k
c_060.htm	04-Jul-2000 08:16	2k
c_059.htm	04-Jul-2000 08:16	3k
c_058a.htm	04-Jul-2000 08:16	1k
c_058.htm	04-Jul-2000 08:16	1k
c_057.htm	04-Jul-2000 08:16	3k
c_056.htm	04-Jul-2000 08:16	2k
c_055a.htm	04-Jul-2000 08:16	7k
c_055.htm	04-Jul-2000 08:16	4k
c_054s.htm	04-Jul-2000 08:16	7k
c_054a.htm	04-Jul-2000 08:16	2k
c_054.htm	04-Jul-2000 08:16	2k
c_053a.htm	04-Jul-2000 08:16	1k
c_053.htm	04-Jul-2000 08:16	2k
c_052a.htm	04-Jul-2000 08:16	2k
c_052.htm	04-Jul-2000 08:16	2k
c_051a.htm	04-Jul-2000 08:16	1k
c_051.htm	04-Jul-2000 08:16	1k
c_050.htm	04-Jul-2000 08:16	3k
c_049a.htm	04-Jul-2000 08:16	1k

c_049.htm	04-Jul-2000 08:16	2k
c_048.htm	04-Jul-2000 08:16	2k
c_047.htm	04-Jul-2000 08:16	3k
c_046a.htm	04-Jul-2000 08:16	1k
c_046.htm	04-Jul-2000 08:16	5k
c_045a.htm	04-Jul-2000 08:16	1k
c_045.htm	04-Jul-2000 08:16	3k
c_044.htm	04-Jul-2000 08:16	3k
c_043.htm	04-Jul-2000 08:16	2k
c_042.htm	04-Jul-2000 08:16	3k
c_041s.htm	04-Jul-2000 08:16	13k
c_041a.htm	04-Jul-2000 08:16	3k
c_041.htm	04-Jul-2000 08:16	2k
c_040.htm	04-Jul-2000 08:16	2k
c_039.htm	04-Jul-2000 08:16	1k
c_038.htm	04-Jul-2000 08:16	2k
c_037a.htm	04-Jul-2000 08:16	1k
c_037.htm	04-Jul-2000 08:16	2k
c_036.htm	04-Jul-2000 08:16	2k
c_035.htm	04-Jul-2000 08:16	2k
c_034.htm	04-Jul-2000 08:16	3k
c_033a.htm	04-Jul-2000 08:16	1k
c_033.htm	04-Jul-2000 08:16	3k
c_032.htm	04-Jul-2000 08:16	3k
c_031.htm	04-Jul-2000 08:15	4k

c_030.htm	04-Jul-2000 08:15	2k
c_029s.htm	04-Jul-2000 08:15	5k
c_029a.htm	04-Jul-2000 08:15	2k
c_029.htm	04-Jul-2000 08:15	2k
c_028a.htm	04-Jul-2000 08:15	2k
c_028.htm	04-Jul-2000 08:15	3k
c_027b.htm	04-Jul-2000 08:15	3k
c_027a.htm	04-Jul-2000 08:15	2k
c_027.htm	04-Jul-2000 08:15	3k
c_026s.htm	04-Jul-2000 08:15	7k
c_026a.htm	04-Jul-2000 08:15	2k
c_026.htm	04-Jul-2000 08:15	2k
c_025.htm	04-Jul-2000 08:15	3k
c_024a.htm	04-Jul-2000 08:15	2k
c_024.htm	04-Jul-2000 08:15	4k
c_023.htm	04-Jul-2000 08:15	4k
c_022.htm	04-Jul-2000 08:15	2k
c_021s.htm	04-Jul-2000 08:15	7k
c_021a.htm	04-Jul-2000 08:15	2k
c_021.htm	04-Jul-2000 08:15	2k
c_020a.htm	04-Jul-2000 08:15	2k
c_020.htm	04-Jul-2000 08:15	2k
c_019b.htm	04-Jul-2000 08:15	1k
c_019a.htm	04-Jul-2000 08:15	4k
c_019.htm	04-Jul-2000 08:15	5k

c_018.htm	04-Jul-2000 08:15	2k
c_017s.htm	04-Jul-2000 08:15	10k
c_017a.htm	04-Jul-2000 08:15	2k
c_017.htm	04-Jul-2000 08:15	2k
c_016.htm	04-Jul-2000 08:15	4k
c_015a.htm	04-Jul-2000 08:15	3k
c_015.htm	04-Jul-2000 08:15	3k
c_014s.htm	04-Jul-2000 08:15	7k
c_014a.htm	04-Jul-2000 08:15	1k
c_014.htm	04-Jul-2000 08:15	2k
c_013a.htm	04-Jul-2000 08:15	1k
c_013.htm	04-Jul-2000 08:15	4k
c_012s.htm	04-Jul-2000 08:15	8k
c_012a.htm	04-Jul-2000 08:15	2k
c_012.htm	04-Jul-2000 08:15	2k
c_011.htm	04-Jul-2000 08:15	2k
c_010a.htm	04-Jul-2000 08:15	1k
c_010.htm	04-Jul-2000 08:15	6k
c_009.htm	04-Jul-2000 08:15	3k
c_008.htm	04-Jul-2000 08:15	3k
c_007.htm	04-Jul-2000 08:15	4k
c_006.htm	04-Jul-2000 08:15	4k
c_005a.htm	04-Jul-2000 08:15	1k
c_005.htm	04-Jul-2000 08:15	3k
c_004.htm	04-Jul-2000 08:15	3k

c_003a.htm	04-Jul-2000 08:15	1k
c_003.htm	04-Jul-2000 08:15	2k
c_002.htm	04-Jul-2000 08:15	4k
c_001a.htm	04-Jul-2000 08:15	1k
c_001.htm	04-Jul-2000 08:15	2k
c_000a.htm	04-Jul-2000 08:15	2k
c_000.htm	04-Jul-2000 08:15	4k
advcw3.htm	04-Jul-2000 08:15	43k
advcw2.htm	04-Jul-2000 08:15	35k
advcw1.htm	04-Jul-2000 08:15	31k
images/	03-Apr-2000 12:52	-

Apache/1.3.12 Server at www.ibilce.unesp.br Port 80

C Programming

© Copyright Brian Brown, 1984-2000. All rights reserved.

Notes | Tests | Home Page

On-line Interactive Tests

JavaScript (Requires compatible browser)	Topics Covered
Test 1	defining variables
Test 2	simple assignments
Test 3	printf() and scanf()
Test 4	for loops
Test 5	while loops and if else
Test 6	compound relationals and switch
Test 7	arrays
Test 8	functions
Test 9	structures
Test 9a	files
Test 10	pointers
Test 11	pointers and structures

C Programming

© Copyright Brian Brown/Peter Henry, 1984-2000. All rights reserved.

| Notes | Tests | Home Page |

Sample Programming Problems

1. Enter 2 numbers and print their sum.

2. Enter 3 numbers and print their average.

3. Enter the temperature in Centigrade, convert it to Fahrenheit and print it out.
   To convert to Fahrenheit, mulitply by 1.8 and add 32.
   E.g. 10C = 10*1.8+32 = 50F

4. Enter your height in feet and inches and convert to metres, given 1 inch equals 0.0254 metres. Print out your height in metres.

5. Enter 2 numbers and print out the largest.

6. Enter 3 numbers and print out the smallest.

7. A salesman is paid a commission on the following basis

Sale Value	Commission
up to $100	zero
over $100 to $1000	2%
over $1000	3%

Enter the sale value and print out the commission value (use a maximum entry value of $32000)

8. Hutt Valley Energy charges its customers for electricity as follows.

Kilowatt-Hours	Cost($)
0 to 500	10
501 to 1000	10 + 0.05 for each kwh over 500
over 1000	35 + 0.03 for each kwh above 1000

Enter the meter reading, calculate the cost and print out how much is charged.

9. Print 8 asterisks (*) down the page.

Output

```
*
*
*
*
*
*
```

*
*

10. Enter a character and a number and print that number of copies of that character.

Example Input
Enter a character: F
Enter a number: 4
F
F
F
F

11. Enter a list of numbers, print the number and continue until the number is greater than 10.

Example Input
Enter a number: 5
The number is 5
Enter a number: 3
The number is 3
Enter a number: 7
The number is 7
Enter a number: 11

12. Enter a list of numbers, print out that number of asterisks (*) on that line until the number entered is 0.

Example Input
Enter a number: 5
*****
Enter a number: 1
*
Enter a number: 0

13. Enter a number between 2 and 20 and print a filled square with sides of that number of asterisks (*).

Example Input
Enter a number: 3
***
***
***

14. Enter a number and print out its multiplication table from 1 to 10.

Example Input
Enter a number: 3
1 times 3 = 3

2 times 3 = 6
.
.
10 times 3 = 30

15. Enter a list of 5 numbers and print their total.

16. Enter a list of numbers terminated by a -1 and print their total.

17. Enter a list of numbers terminated by a -1 and print how many numbers where entered in.

18. Enter a list of numbrs terminated by a -1 and print the smallest number (assume all numbers entered are positive).

19. Enter a list of names terminated by a Z, and print out the alphabetically smallest.

20. Enter a string of results for a History exam terminated by a -1. The pass mark is 50. Print the number of passes and the number of fails.

21. Input the time started and finished at work in hours and minutes, then print out the time spent at work in hours and minutes.

Example Input
Enter start time: 8 30
Enter finish time: 11 15
2 hours 45 minutes

22. Enter a persons weight in kilograms and height in metres. Calculate the persons Quetelet Index (kilos / (metres*metres)). Print out the Quetelet Index and an appropriate message as indicated by the table below.

Below 20	Underweight
20 to below 25	Healthy weight
25 to below 30	Mildly overweight
30 to below 40	Very overweight
40 and above	Extremely overweight

23. Enter a list of numbers terminated by a -1 and print the difference between each pair of numbers.

Example Input
Enter a number: 3
Enter a number: 5
Difference is 2
Enter a number: 6
Difference is 1
Enter a number: 10
Difference is 4

Enter a number: -1

24. Enter 2 numbers and print them out. Then print the next 13 numbers in the sequence, where the next number is the sum of the previous two.

Example Input
Enter the first number: 1
Enter the second number: 3
1 3 4 7 11 18 29 47 76 123 322 521 843 1364

25. Enter your name and a number and print that number of copies of your name.

Example Input
Enter your name: Fred
Enter a number: 4
Fred
Fred
Fred
Fred

26. Enter your name, convert to uppercase, reverse and print.

Example Input
Enter your name: Fred
DERF

27. Input a sentence, count and print the number of spaces.

Example Input
Enter the sentence: A cat sat on the mat.
Number of spaces = 5

28. Input a sentence and print one word per line. Assume one space between words and the sentence is terminated with a period.

Example Input
Enter the sentence: A cat sat on the mat.
A
cat
sat
on
the
mat

29. Input a sentence and print out if it is a palidrome

Example Input
Enter the sentence: Madam I'm Adam
It is a palidrome

Example Input
Enter the sentence: Fred
Not a palidrome

30. Input ten numbers into an array, and print the numbers in reverse order.

Example Input
Enter the numbers: 5 4 6 7 22 19 12 15 2 1
1 2 15 12 19 22 7 6 4 5

31. Input ten numbers into an array, and print these 3 times.

Example Input
Enter the numbers: 5 4 6 7 22 19 12 15 2 1
5 4 6 7 22 19 12 15 2 1
5 4 6 7 22 19 12 15 2 1
5 4 6 7 22 19 12 15 2 1

32. Input ten numbers into an array, calculate and print the average, and print out those values below the average.

Example Input
Enter the numbers: 5 4 6 7 22 19 12 15 2 1
Average = 9.3
Those numbers below the average 5 4 6 7 2 1

33. Input ten numbers into an array, and print out the largest.

Example Input
Enter the numbers: 5 4 6 7 22 19 12 15 2 1
The largest number is 22

34. Input ten numbers into an array, using values of 0 to 99, and print out all numbers except for the largest number..

Example Input
Enter the numbers: 5 4 6 7 22 19 12 15 2 1
5 4 6 7 19 12 15 2 1

35. Input ten numbers into an array, using values of 0 to 99, and print the values in ascending order.

Example Input
Enter the numbers: 5 4 6 7 22 19 12 15 2 1
1 2 4 5 6 7 12 15 19 22

INDEX

© Copyright Brian Brown/Peter Henry, 1984-2000. All rights reserved.

Index of /courseware/cprogram

Name	Last modified	Size	Description
Parent Directory	04-Jul-2000 08:54	-	
images/	03-Apr-2000 12:52	-	
c_019b.htm	04-Jul-2000 08:15	1k	
c_005a.htm	04-Jul-2000 08:15	1k	
c_033a.htm	04-Jul-2000 08:16	1k	
c_051a.htm	04-Jul-2000 08:16	1k	
c_010a.htm	04-Jul-2000 08:15	1k	
c_085a.htm	04-Jul-2000 08:16	1k	
c_013a.htm	04-Jul-2000 08:15	1k	
c_003a.htm	04-Jul-2000 08:15	1k	
c_058a.htm	04-Jul-2000 08:16	1k	
c_064a.htm	04-Jul-2000 08:16	1k	
c_037a.htm	04-Jul-2000 08:16	1k	
c_046a.htm	04-Jul-2000 08:16	1k	
c_045a.htm	04-Jul-2000 08:16	1k	
c_039.htm	04-Jul-2000 08:16	1k	
c_097.htm	04-Jul-2000 08:16	1k	
c_083a.htm	04-Jul-2000 08:16	1k	
c_109.htm	04-Jul-2000 08:16	1k	
c_001a.htm	04-Jul-2000 08:15	1k	
c_101.htm	04-Jul-2000 08:16	1k	

c_049a.htm	04-Jul-2000 08:16	1k
c_014a.htm	04-Jul-2000 08:15	1k
c_096.htm	04-Jul-2000 08:16	1k
c_058.htm	04-Jul-2000 08:16	1k
c_090.htm	04-Jul-2000 08:16	1k
c_0771a.htm	04-Jul-2000 08:16	1k
c_051.htm	04-Jul-2000 08:16	1k
c_053a.htm	04-Jul-2000 08:16	1k
simple4.htm	04-Jul-2000 08:16	2k
c_012a.htm	04-Jul-2000 08:15	2k
c_020a.htm	04-Jul-2000 08:15	2k
c_003.htm	04-Jul-2000 08:15	2k
c_027a.htm	04-Jul-2000 08:15	2k
c_072.htm	04-Jul-2000 08:16	2k
c_087.htm	04-Jul-2000 08:16	2k
c_070.htm	04-Jul-2000 08:16	2k
c_049.htm	04-Jul-2000 08:16	2k
c_030.htm	04-Jul-2000 08:15	2k
c_029a.htm	04-Jul-2000 08:15	2k
c_052.htm	04-Jul-2000 08:16	2k
c_105a.htm	04-Jul-2000 08:16	2k
c_021.htm	04-Jul-2000 08:15	2k
c_066.htm	04-Jul-2000 08:16	2k
c_021a.htm	04-Jul-2000 08:15	2k
c_083.htm	04-Jul-2000 08:16	2k

	c_067a.htm	04-Jul-2000 08:16	2k
	c_103.htm	04-Jul-2000 08:16	2k
	c_035.htm	04-Jul-2000 08:16	2k
	c_000a.htm	04-Jul-2000 08:15	2k
	c_014.htm	04-Jul-2000 08:15	2k
	c_100.htm	04-Jul-2000 08:16	2k
	c_012.htm	04-Jul-2000 08:15	2k
	c_029.htm	04-Jul-2000 08:15	2k
	c_091.htm	04-Jul-2000 08:16	2k
	c_099.htm	04-Jul-2000 08:16	2k
	c_026.htm	04-Jul-2000 08:15	2k
	c_074a.htm	04-Jul-2000 08:16	2k
	c_089.htm	04-Jul-2000 08:16	2k
	c_001.htm	04-Jul-2000 08:15	2k
	c_063.htm	04-Jul-2000 08:16	2k
	c_073.htm	04-Jul-2000 08:16	2k
	c_074.htm	04-Jul-2000 08:16	2k
	c_062.htm	04-Jul-2000 08:16	2k
	c_020.htm	04-Jul-2000 08:15	2k
	c_053.htm	04-Jul-2000 08:16	2k
	c_0771.htm	04-Jul-2000 08:16	2k
	c_067.htm	04-Jul-2000 08:16	2k
	c_077a.htm	04-Jul-2000 08:16	2k
	c_106.htm	04-Jul-2000 08:16	2k
	c_092.htm	04-Jul-2000 08:16	2k

c_082a.htm	04-Jul-2000 08:16	2k
c_065.htm	04-Jul-2000 08:16	2k
c_026a.htm	04-Jul-2000 08:15	2k
c_054a.htm	04-Jul-2000 08:16	2k
c_017a.htm	04-Jul-2000 08:15	2k
c_040.htm	04-Jul-2000 08:16	2k
c_054.htm	04-Jul-2000 08:16	2k
c_018.htm	04-Jul-2000 08:15	2k
c_102.htm	04-Jul-2000 08:16	2k
c_043.htm	04-Jul-2000 08:16	2k
c_052a.htm	04-Jul-2000 08:16	2k
c_017.htm	04-Jul-2000 08:15	2k
c_061.htm	04-Jul-2000 08:16	2k
c_068.htm	04-Jul-2000 08:16	2k
c_098.htm	04-Jul-2000 08:16	2k
c_084a.htm	04-Jul-2000 08:16	2k
c_011.htm	04-Jul-2000 08:15	2k
c_082.htm	04-Jul-2000 08:16	2k
c_048.htm	04-Jul-2000 08:16	2k
c_080.htm	04-Jul-2000 08:16	2k
c_037.htm	04-Jul-2000 08:16	2k
c_038.htm	04-Jul-2000 08:16	2k
c_060.htm	04-Jul-2000 08:16	2k
c_084.htm	04-Jul-2000 08:16	2k
c_077.htm	04-Jul-2000 08:16	2k

c_036.htm	04-Jul-2000 08:16	2k
c_071.htm	04-Jul-2000 08:16	2k
c_104.htm	04-Jul-2000 08:16	2k
c_078.htm	04-Jul-2000 08:16	2k
c_095.htm	04-Jul-2000 08:16	2k
c_064.htm	04-Jul-2000 08:16	2k
c_079.htm	04-Jul-2000 08:16	2k
c_024a.htm	04-Jul-2000 08:15	2k
c_105.htm	04-Jul-2000 08:16	2k
c_022.htm	04-Jul-2000 08:15	2k
c_041.htm	04-Jul-2000 08:16	2k
onlinet.htm	04-Jul-2000 08:16	2k
c_108.htm	04-Jul-2000 08:16	2k
c_061a.htm	04-Jul-2000 08:16	2k
c_081.htm	04-Jul-2000 08:16	2k
c_028a.htm	04-Jul-2000 08:15	2k
c_056.htm	04-Jul-2000 08:16	2k
c_044.htm	04-Jul-2000 08:16	3k
c_033.htm	04-Jul-2000 08:16	3k
c_027.htm	04-Jul-2000 08:15	3k
c_041a.htm	04-Jul-2000 08:16	3k
c_088.htm	04-Jul-2000 08:16	3k
c_045.htm	04-Jul-2000 08:16	3k
c_050.htm	04-Jul-2000 08:16	3k
c_009.htm	04-Jul-2000 08:15	3k

c_047.htm	04-Jul-2000 08:16	3k
c_059.htm	04-Jul-2000 08:16	3k
c_005.htm	04-Jul-2000 08:15	3k
c_004.htm	04-Jul-2000 08:15	3k
c_015.htm	04-Jul-2000 08:15	3k
c_032.htm	04-Jul-2000 08:16	3k
c_086.htm	04-Jul-2000 08:16	3k
c_028.htm	04-Jul-2000 08:15	3k
c_076.htm	04-Jul-2000 08:16	3k
c_075.htm	04-Jul-2000 08:16	3k
c_057.htm	04-Jul-2000 08:16	3k
c_042.htm	04-Jul-2000 08:16	3k
c_008.htm	04-Jul-2000 08:15	3k
c_015a.htm	04-Jul-2000 08:15	3k
c_027b.htm	04-Jul-2000 08:15	3k
c_081a.htm	04-Jul-2000 08:16	3k
c_034.htm	04-Jul-2000 08:16	3k
c_025.htm	04-Jul-2000 08:15	3k
cstart.htm	04-Jul-2000 08:16	3k
c_019a.htm	04-Jul-2000 08:15	4k
c_000.htm	04-Jul-2000 08:15	4k
c_088a.htm	04-Jul-2000 08:16	4k
c_069.htm	04-Jul-2000 08:16	4k
c_006.htm	04-Jul-2000 08:15	4k
c_016.htm	04-Jul-2000 08:15	4k

c_031.htm	04-Jul-2000 08:15	4k
howtopro.htm	04-Jul-2000 08:16	4k
c_107.htm	04-Jul-2000 08:16	4k
c_002.htm	04-Jul-2000 08:15	4k
c_023.htm	04-Jul-2000 08:15	4k
c_024.htm	04-Jul-2000 08:15	4k
c_007.htm	04-Jul-2000 08:15	4k
c_055.htm	04-Jul-2000 08:16	4k
c_013.htm	04-Jul-2000 08:15	4k
c_046.htm	04-Jul-2000 08:16	5k
c_019.htm	04-Jul-2000 08:15	5k
c_029s.htm	04-Jul-2000 08:15	5k
c_093.htm	04-Jul-2000 08:16	5k
c_085.htm	04-Jul-2000 08:16	5k
joystick.htm	04-Jul-2000 08:16	5k
c_075a.htm	04-Jul-2000 08:16	5k
c_077s.htm	04-Jul-2000 08:16	6k
c_067s.htm	04-Jul-2000 08:16	6k
c_010.htm	04-Jul-2000 08:15	6k
default.htm	04-Jul-2000 08:16	7k
c_074s.htm	04-Jul-2000 08:16	7k
c_055a.htm	04-Jul-2000 08:16	7k
c_021s.htm	04-Jul-2000 08:15	7k
c_054s.htm	04-Jul-2000 08:16	7k
c_014s.htm	04-Jul-2000 08:15	7k

c_026s.htm	04-Jul-2000 08:15	7k
c_012s.htm	04-Jul-2000 08:15	8k
c_081s.htm	04-Jul-2000 08:16	9k
c_017s.htm	04-Jul-2000 08:15	10k
problems.htm	04-Jul-2000 08:16	11k
c_094.htm	04-Jul-2000 08:16	11k
c_041s.htm	04-Jul-2000 08:16	13k
c_110.htm	04-Jul-2000 08:16	20k
advcw1.htm	04-Jul-2000 08:15	31k
advcw2.htm	04-Jul-2000 08:15	35k
advcw3.htm	04-Jul-2000 08:15	43k

Apache/1.3.12 Server at www.ibilce.unesp.br Port 80

Index of /courseware/cprogram/images

Name	Last modified	Size	Description
Parent Directory	24-May-2000 10:59	-	
ballrem.gif	04-Jul-2000 08:15	1k	
ballloc.gif	04-Jul-2000 08:15	1k	
disk1.jpg	04-Jul-2000 08:15	1k	
cam1.jpg	04-Jul-2000 08:15	1k	
next.gif	04-Jul-2000 08:15	2k	
previous.gif	04-Jul-2000 08:15	2k	
menu.gif	04-Jul-2000 08:15	2k	
c_019.gif	04-Jul-2000 08:15	2k	
mail.gif	04-Jul-2000 08:15	4k	
for5.gif	04-Jul-2000 08:15	6k	
for1.gif	04-Jul-2000 08:15	6k	
for4.gif	04-Jul-2000 08:15	6k	
for3.gif	04-Jul-2000 08:15	6k	
for2.gif	04-Jul-2000 08:15	6k	
for6.gif	04-Jul-2000 08:15	6k	
for_loop.jpg	04-Jul-2000 08:15	12k	
for_loop.rm	04-Jul-2000 08:15	740k	
for_loop.avi	04-Jul-2000 08:15	4.3M	

Apache/1.3.12 Server at www.ibilce.unesp.br Port 80

Index of /courseware/cprogram

Name	Last modified	Size	Description
Parent Directory	04-Jul-2000 08:54	-	
advcw3.htm	04-Jul-2000 08:15	43k	
advcw2.htm	04-Jul-2000 08:15	35k	
advcw1.htm	04-Jul-2000 08:15	31k	
c_110.htm	04-Jul-2000 08:16	20k	
c_041s.htm	04-Jul-2000 08:16	13k	
c_094.htm	04-Jul-2000 08:16	11k	
problems.htm	04-Jul-2000 08:16	11k	
c_017s.htm	04-Jul-2000 08:15	10k	
c_081s.htm	04-Jul-2000 08:16	9k	
c_012s.htm	04-Jul-2000 08:15	8k	
c_026s.htm	04-Jul-2000 08:15	7k	
c_014s.htm	04-Jul-2000 08:15	7k	
c_054s.htm	04-Jul-2000 08:16	7k	
c_021s.htm	04-Jul-2000 08:15	7k	
c_055a.htm	04-Jul-2000 08:16	7k	
c_074s.htm	04-Jul-2000 08:16	7k	
default.htm	04-Jul-2000 08:16	7k	
c_010.htm	04-Jul-2000 08:15	6k	
c_067s.htm	04-Jul-2000 08:16	6k	
c_077s.htm	04-Jul-2000 08:16	6k	

c_075a.htm	04-Jul-2000	08:16	5k
joystick.htm	04-Jul-2000	08:16	5k
c_085.htm	04-Jul-2000	08:16	5k
c_093.htm	04-Jul-2000	08:16	5k
c_029s.htm	04-Jul-2000	08:15	5k
c_019.htm	04-Jul-2000	08:15	5k
c_046.htm	04-Jul-2000	08:16	5k
c_013.htm	04-Jul-2000	08:15	4k
c_055.htm	04-Jul-2000	08:16	4k
c_007.htm	04-Jul-2000	08:15	4k
c_024.htm	04-Jul-2000	08:15	4k
c_023.htm	04-Jul-2000	08:15	4k
c_002.htm	04-Jul-2000	08:15	4k
c_107.htm	04-Jul-2000	08:16	4k
howtopro.htm	04-Jul-2000	08:16	4k
c_031.htm	04-Jul-2000	08:15	4k
c_016.htm	04-Jul-2000	08:15	4k
c_006.htm	04-Jul-2000	08:15	4k
c_069.htm	04-Jul-2000	08:16	4k
c_088a.htm	04-Jul-2000	08:16	4k
c_000.htm	04-Jul-2000	08:15	4k
c_019a.htm	04-Jul-2000	08:15	4k
cstart.htm	04-Jul-2000	08:16	3k
c_025.htm	04-Jul-2000	08:15	3k
c_034.htm	04-Jul-2000	08:16	3k

c_081a.htm	04-Jul-2000 08:16	3k
c_027b.htm	04-Jul-2000 08:15	3k
c_015a.htm	04-Jul-2000 08:15	3k
c_008.htm	04-Jul-2000 08:15	3k
c_042.htm	04-Jul-2000 08:16	3k
c_057.htm	04-Jul-2000 08:16	3k
c_075.htm	04-Jul-2000 08:16	3k
c_076.htm	04-Jul-2000 08:16	3k
c_028.htm	04-Jul-2000 08:15	3k
c_086.htm	04-Jul-2000 08:16	3k
c_032.htm	04-Jul-2000 08:16	3k
c_015.htm	04-Jul-2000 08:15	3k
c_004.htm	04-Jul-2000 08:15	3k
c_005.htm	04-Jul-2000 08:15	3k
c_059.htm	04-Jul-2000 08:16	3k
c_047.htm	04-Jul-2000 08:16	3k
c_009.htm	04-Jul-2000 08:15	3k
c_050.htm	04-Jul-2000 08:16	3k
c_045.htm	04-Jul-2000 08:16	3k
c_088.htm	04-Jul-2000 08:16	3k
c_041a.htm	04-Jul-2000 08:16	3k
c_027.htm	04-Jul-2000 08:15	3k
c_033.htm	04-Jul-2000 08:16	3k
c_044.htm	04-Jul-2000 08:16	3k
c_056.htm	04-Jul-2000 08:16	2k

c_028a.htm	04-Jul-2000 08:15	2k
c_081.htm	04-Jul-2000 08:16	2k
c_061a.htm	04-Jul-2000 08:16	2k
c_108.htm	04-Jul-2000 08:16	2k
onlinet.htm	04-Jul-2000 08:16	2k
c_041.htm	04-Jul-2000 08:16	2k
c_022.htm	04-Jul-2000 08:15	2k
c_105.htm	04-Jul-2000 08:16	2k
c_024a.htm	04-Jul-2000 08:15	2k
c_079.htm	04-Jul-2000 08:16	2k
c_064.htm	04-Jul-2000 08:16	2k
c_095.htm	04-Jul-2000 08:16	2k
c_078.htm	04-Jul-2000 08:16	2k
c_104.htm	04-Jul-2000 08:16	2k
c_071.htm	04-Jul-2000 08:16	2k
c_036.htm	04-Jul-2000 08:16	2k
c_077.htm	04-Jul-2000 08:16	2k
c_084.htm	04-Jul-2000 08:16	2k
c_060.htm	04-Jul-2000 08:16	2k
c_038.htm	04-Jul-2000 08:16	2k
c_037.htm	04-Jul-2000 08:16	2k
c_080.htm	04-Jul-2000 08:16	2k
c_048.htm	04-Jul-2000 08:16	2k
c_082.htm	04-Jul-2000 08:16	2k
c_011.htm	04-Jul-2000 08:15	2k

c_084a.htm	04-Jul-2000 08:16	2k
c_098.htm	04-Jul-2000 08:16	2k
c_068.htm	04-Jul-2000 08:16	2k
c_061.htm	04-Jul-2000 08:16	2k
c_017.htm	04-Jul-2000 08:15	2k
c_052a.htm	04-Jul-2000 08:16	2k
c_043.htm	04-Jul-2000 08:16	2k
c_102.htm	04-Jul-2000 08:16	2k
c_018.htm	04-Jul-2000 08:15	2k
c_054.htm	04-Jul-2000 08:16	2k
c_040.htm	04-Jul-2000 08:16	2k
c_017a.htm	04-Jul-2000 08:15	2k
c_054a.htm	04-Jul-2000 08:16	2k
c_026a.htm	04-Jul-2000 08:15	2k
c_065.htm	04-Jul-2000 08:16	2k
c_082a.htm	04-Jul-2000 08:16	2k
c_092.htm	04-Jul-2000 08:16	2k
c_106.htm	04-Jul-2000 08:16	2k
c_077a.htm	04-Jul-2000 08:16	2k
c_067.htm	04-Jul-2000 08:16	2k
c_0771.htm	04-Jul-2000 08:16	2k
c_053.htm	04-Jul-2000 08:16	2k
c_020.htm	04-Jul-2000 08:15	2k
c_062.htm	04-Jul-2000 08:16	2k
c_074.htm	04-Jul-2000 08:16	2k

c_073.htm	04-Jul-2000 08:16	2k
c_063.htm	04-Jul-2000 08:16	2k
c_001.htm	04-Jul-2000 08:15	2k
c_089.htm	04-Jul-2000 08:16	2k
c_074a.htm	04-Jul-2000 08:16	2k
c_026.htm	04-Jul-2000 08:15	2k
c_099.htm	04-Jul-2000 08:16	2k
c_091.htm	04-Jul-2000 08:16	2k
c_029.htm	04-Jul-2000 08:15	2k
c_012.htm	04-Jul-2000 08:15	2k
c_100.htm	04-Jul-2000 08:16	2k
c_014.htm	04-Jul-2000 08:15	2k
c_000a.htm	04-Jul-2000 08:15	2k
c_035.htm	04-Jul-2000 08:16	2k
c_103.htm	04-Jul-2000 08:16	2k
c_067a.htm	04-Jul-2000 08:16	2k
c_083.htm	04-Jul-2000 08:16	2k
c_021a.htm	04-Jul-2000 08:15	2k
c_066.htm	04-Jul-2000 08:16	2k
c_021.htm	04-Jul-2000 08:15	2k
c_105a.htm	04-Jul-2000 08:16	2k
c_052.htm	04-Jul-2000 08:16	2k
c_029a.htm	04-Jul-2000 08:15	2k
c_030.htm	04-Jul-2000 08:15	2k
c_049.htm	04-Jul-2000 08:16	2k

c_070.htm	04-Jul-2000 08:16	2k
c_087.htm	04-Jul-2000 08:16	2k
c_072.htm	04-Jul-2000 08:16	2k
c_027a.htm	04-Jul-2000 08:15	2k
c_003.htm	04-Jul-2000 08:15	2k
c_020a.htm	04-Jul-2000 08:15	2k
c_012a.htm	04-Jul-2000 08:15	2k
simple4.htm	04-Jul-2000 08:16	2k
c_053a.htm	04-Jul-2000 08:16	1k
c_051.htm	04-Jul-2000 08:16	1k
c_0771a.htm	04-Jul-2000 08:16	1k
c_090.htm	04-Jul-2000 08:16	1k
c_058.htm	04-Jul-2000 08:16	1k
c_096.htm	04-Jul-2000 08:16	1k
c_014a.htm	04-Jul-2000 08:15	1k
c_049a.htm	04-Jul-2000 08:16	1k
c_101.htm	04-Jul-2000 08:16	1k
c_001a.htm	04-Jul-2000 08:15	1k
c_109.htm	04-Jul-2000 08:16	1k
c_083a.htm	04-Jul-2000 08:16	1k
c_097.htm	04-Jul-2000 08:16	1k
c_039.htm	04-Jul-2000 08:16	1k
c_045a.htm	04-Jul-2000 08:16	1k
c_046a.htm	04-Jul-2000 08:16	1k
c_037a.htm	04-Jul-2000 08:16	1k

c_064a.htm	04-Jul-2000 08:16	1k
c_058a.htm	04-Jul-2000 08:16	1k
c_003a.htm	04-Jul-2000 08:15	1k
c_013a.htm	04-Jul-2000 08:15	1k
c_085a.htm	04-Jul-2000 08:16	1k
c_010a.htm	04-Jul-2000 08:15	1k
c_051a.htm	04-Jul-2000 08:16	1k
c_033a.htm	04-Jul-2000 08:16	1k
c_005a.htm	04-Jul-2000 08:15	1k
c_019b.htm	04-Jul-2000 08:15	1k
images/	03-Apr-2000 12:52	-

Apache/1.3.12 Server at www.ibilce.unesp.br Port 80

Index of /courseware/cprogram/images

Name	Last modified	Size	Description
Parent Directory	24-May-2000 10:59	-	
for_loop.avi	04-Jul-2000 08:15	4.3M	
for_loop.rm	04-Jul-2000 08:15	740k	
for_loop.jpg	04-Jul-2000 08:15	12k	
for6.gif	04-Jul-2000 08:15	6k	
for2.gif	04-Jul-2000 08:15	6k	
for3.gif	04-Jul-2000 08:15	6k	
for4.gif	04-Jul-2000 08:15	6k	
for1.gif	04-Jul-2000 08:15	6k	
for5.gif	04-Jul-2000 08:15	6k	
mail.gif	04-Jul-2000 08:15	4k	
c_019.gif	04-Jul-2000 08:15	2k	
menu.gif	04-Jul-2000 08:15	2k	
previous.gif	04-Jul-2000 08:15	2k	
next.gif	04-Jul-2000 08:15	2k	
cam1.jpg	04-Jul-2000 08:15	1k	
disk1.jpg	04-Jul-2000 08:15	1k	
ballloc.gif	04-Jul-2000 08:15	1k	
ballrem.gif	04-Jul-2000 08:15	1k	

Apache/1.3.12 Server at www.ibilce.unesp.br Port 80

C Programming

Simplified Program 4

```
/* sample file handling processing words in a file */

/* to count the number of words in a text file */

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

#define TRUE  1
#define FALSE 0

FILE *fin, *fopen();

main()
{
        char filename[30];      /* name of input file */
        int in_word = FALSE;    /* are we in a word */
        int ch;                 /* character read from file */
        int word_count;

        printf("Please enter name of file.\n");
        scanf("%s", filename );

        fin = fopen( filename, "rt" );
        if( fin == NULL ) {
                printf("Error opening %s\n", filename );
                exit( 1 );
        }

        ch = fgetc( fin );

        while( !feof( fin ) ) {
                switch( ch ) {
                        case ',':
                        case '?':
                        case '!':
                        case '.':
                        case ' ':
                        case '\t':
```

```
                        case '\r':
                        case '\n': if( in_word == TRUE ) {
                                        in_word = FALSE;
                                        word_count++;
                                   }
                                   break;
                        default : in_word = TRUE;
                }
                ch = fgetc( fin );
        }
        printf(" The number of words found was %d\n", word_count );
        fclose( fin );
}
```

©Copyright B Brown. 1984-2000. All rights reserved.