

C Programming

Learning Package 12

Odds and Ends

Dr Scott Turner

School of Science and Technology

University of Northampton



**Introduction**

In this Learning Package we are going to look at:

* programs that call operating system commands,
* combining programs together to form a new program,
* using your compiler to open a Windows ‘window’ to display messages;
* operations on bits using C.

**Learning outcomes**

At the end of this Learning Package, the reader should be able to:

* Use system commands inside a C program
* Combine programs together.
* Write a C program that opens a window that displays a message.
* Write a C program to perform bitwise operations

**Study guide**

Session 1:

 Section 12.1 System() function Task 12.1

Section 12.2 Combining programs Task 12.2

Task 12.3

Session 2:

Section 12.3 Introduction to Windows programming SAQ 12.1

Task 12.4

Task 12.5

Session 3:

Section 12.4 Bitwise operations SAQ 12.2

Task 12.6

Task 12.7

# System() function

C has the ability to call operating system commands from within a C program, using the function system().

#include <stdio.h>

#include <stdlib.h>

int main()

{

system("dir c:");

system("PAUSE"); return 0;

}

The operating system instruction is put inside the brackets as a quote (for example system(“dir c:”);)

This program above calls two operating system commands dir c: and pause. These commands will display the filenames for files in the c: directory and make the program to wait for a key press respectively (see Figure 12.1).

Figure 12.1

Task 12.1

#include <stdio.h>

#include <stdlib.h>

int main()

{

system("dir /w");

system("PAUSE");

system("md test\_1"); /\*makes a new directory\*/ system("dir test\_1");

system("PAUSE"); return 0;

}

* + 1. Type in the program above and comment each line within the function main() to describe its operation.
		2. Run the program twice and see if there is any variation in the output.

# Combining programs together.

A lot of C program come from multiple source programs, linked together. So far we have only looked at programs with one source code, using Dev C++ we will combine several programs together.

Task 12.2

* + 1. Open Dev C++ and create a new project as before.
		2. When the window we have used to write our programs appears on the screen, go to FileNewSource File and click yes.
		3. A blank window should appear now type in the following:

#include <stdio.h> void cls1()

{

system("cls");

}

int add(int a, int b, int c)

{

return(a+b+c);

}

* + 1. go to FileSave and save this file as prog1.c
		2. go to FileNewSource File and click yes.
		3. Now copy the following

void cls1()

int add(int a, int b, int c)

* + 1. go to FileSave and save this file as prog1.h (you will need to change the save to type selection to header files).
		2. Now return to the source code currently marked as main.c and type in the following:

#include <stdio.h>

#include <stdlib.h> int main()

{

printf("Before clearing the screen\n\n"); system("PAUSE");

cls1();

printf("after clearing the screen\n\n");

printf("Add three numbers together %d\n\n",add(1,2,3)); system("PAUSE");

return 0;

}

* + 1. Compile and execute as before save the file as combined1.c

Task 12.2 showed how programs could be combined, Dev C++ takes away many of the more complicated elements of joining the programs together. Notice that it is only the program combined1.c has the function main() within it, the program prog1.c does not needed. You have effectively created your own library file so more functions could be added to prog1.c and prog1.h, to make them more versatile.

Task 12.3

Alter the project in task 12.2 so that instead of using system(“PAUSE”); the program in combined 1 uses a new function p\_it() defined by you to do the same job. Hint: you could use system(“PAUSE”) inside the new function.

# Introduction to Windows programming

We have to use different functions and approaches when we use want to the features of Windows.

The program below is a relatively simple Windows program.

#define WIN32\_LEAN\_AND\_MEAN

#include <windows.h>

int WINAPI WinMain (HINSTANCE hThisInstance,

HINSTANCE hPrevInstance, LPSTR lpcmdline,

int ncmdshow)

{

MessageBox(NULL, “What’s up Doc","My First Windows

Program",MB\_OK); return (0);

}

Figure 12.2

In the program above, you will find the following lines

#define WIN32\_LEAN\_AND\_MEAN

#include <windows.h>

We have already met #include before, but this time we have included the main windows library, windows.h . This adds features to the program to allow easy to use windows operations. The #define WIN32\_LEAN\_AND\_MEAN

line forces some of the options in the windows.h file to be used.

int WINAPI WinMain (HINSTANCE hThisInstance,

HINSTANCE hPrevInstance, LPSTR lpcmdline,

int ncmdshow)

{

}

In this relatively simple windows program, where the compiler looks for the starting function has changed. Previously the function main() was used as the starting point. In this case, the starting point has changed to WinMain(). This function is of a new data type WINAPI, used for windows applications.

There are four parameters declared:

hThisInstance: This is the ‘handle’ for the application, we can think of this as ID code unique to this application.

hPrevInstance: This is similar to hThisInstance but is actually is no longer needed, and is the handle for the previous instance.

lpcmdline: a pointer to command line parameters and is a string that could be passed into the function.

ncmdshow: An integer value that tells the application how the window will initially be shown.

MessageBox(NULL,"What's up Doc","My First Windows Program",MB\_OK);

MessageBox has four parameters

* hWnd: A handle to the parent windows. In this case, NULL.
* lpText: A pointer to the message string. In this case, "What's up Doc". This puts the words What’s up Doc inside the window
* lpCaption: A pointer to the title string. In this case, "My First Windows Program". This puts the words My first Windows Program in the title bar at the top of the screen.
* uType: Style of message box. In this case MB\_OK that is the default and puts an OK button in the window you have created.

SAQ 12.1

The function MessageBox has **f r** parameters. The first parameter is a

**h e** to a parent **w w**, which is often null. The second parameter is a **pointer** to a **s g** that will form the **m e** inside the **window**. The third parameter is a **p r** to a string that puts a message in the windows title bar. Finally, a parameter for specifying the **s e** of the window.

Task 12.4

1. Type in the Windows program at the beginning of this section.
2. In place of MB\_OK for uType place the following and write a description of what you saw?
	1. MB\_OK|MB\_ICONEXCLAMATION
	2. MB\_OKCANCEL|MB\_ICONEXCLAMATION
	3. MB\_YESNOCANCEL|MB\_ICONSTOP
	4. MB\_RETRYCANCEL|MB\_ICONQUESTION

Task 12.5

Write a program to display the following

* + - **My Second windows program** in the title bar
		- **This is my own program** as the text inside the window
		- Display two buttons Yes and No and a stop sign

I

Figure 12.3

# Bitwise operations

The basic unit of computer storage is the bit. C has several operators that let you manipulate individual bits.

12.4.1 Shift operators

The shifts operators move the bits (shift) in an integer variable by specific number of positions, either to the left (<<) or the right (>>).

What this means is the bits in a number are moved so many positions in the required direction.

So if x=48 in binary that would be 00110000. If the number is right-shifted 2 places (x>>2) then the bits are shifted two places to the rights so the binary number would now be 00001100 giving a decimal number 12. If the new number is right shift 3 places (x>>3) the binary number is 00000001, a bit has been lost, and this binary number gives the decimal number 1.

So starting again with x=48 and the binary number 00110000, if the number is left shifted 1 place (x<<1) then the binary number is 01100000 which is 96 in decimal. If we left shift this new number 2 places (x<<2) then the binary number for an 8-bit binary number is 10000000, which is the decimal number 128.

SAQ 12.2

What would happen to the results if 16 bits or 32-bits were used to store each integer?

* + 1. AND bitwise operator (&)

There are a set of logical operators used to manipulate bits, these are AND, OR and Exclusive OR. These work in a similar way to AND (&), OR (|) and Exclusive OR (^) you will have met previously, but are applied to individual bits from two numbers are compared. Lets look at the Bitwise AND. In this operation a bit is the new number only results in a 1 only if the corresponding bits in the two numbers are 1.

Example

|  |  |
| --- | --- |
| A | 11110000 |
| B | 01010101 |
| A&B | 01010000 |

* + 1. OR operator (|)

Lets look at the Bitwise OR. In this operation a bit is the new number only results in a 1 only if the either or both of the corresponding bits in the two numbers are 1.

Example

|  |  |
| --- | --- |
| A | 11110000 |
| B | 01010101 |
| A|B | 11110101 |

* + 1. Exclusive OR operator (^)

In this operation a bit is the new number only results in a 1 only if the corresponding bits in the two numbers are different.

Example

|  |  |
| --- | --- |
| A | 11110000 |
| B | 01010101 |
| A^B | 10100101 |

* + 1. Complement operator (~)

In this operation, the new number is the opposite of those in the original Example

A 11110000

~A 00001111

Task 12.6

Write type in the program below, before you run the program, write down what you expected it to and then run the program.

Did it do what you expected? If not why do you think there was a difference? What does unsigned int do?

#include <stdio.h>

#include <stdlib.h> int main()

{

unsigned int y,x=256; int count;

printf("Decimal\t\tshift left by\tresult\n"); for (count=1;count<8;count++)

{

y=x<<count; printf("%d\t\t%d\t\t%d\n",x,count,y);

}

printf("Decimal\t\tshift right by\tresult\n"); for (count=1;count<8;count++)

{

y=x>>count; printf("%d\t\t%d\t\t%d\n",x,count,y);

}

system("PAUSE"); return 0;

}

Task 12.7

Fill in the comments in place of the question marks and execute the program.

What did the program do? what results did it produce?

#include <stdio.h>

#include <stdlib.h>

int main()

{

unsigned int a,b,c,d,e=0; a=240; /\*In binary 11110000 \*/ b=129; /\*In binary 10000001\*/

/\*a AND b should equal 10000000=128\*/ printf("\n a AND b= %d\n",a&b);

/\*a OR b should equal 11110001=241\*/ printf("\n a OR b= %d\n",a|b);

/\*a Exclusive OR b should equal 01110001=113\*/ printf("\n a Exclusive OR b= %d\n",a^b);

c=~a;

/\*complement of a should equal 100001111=15\*/ printf("\n Complement of %d = %u\n",a,c); d=~b;

/\*complement of b should equal what?\*/ printf("\n Complement of %d = %u\n",b,d);

/\*complement of 0 should equal what?\*/ printf("\n Complement of 0 = %u\n",~0); system("PAUSE");

return 0;

}

Figure 12.4

**SAQ Answers**

SAQ 12.1

The function MessageBox has **four** parameters. The first parameter is a **handle** to a parent **window**, which is often a null. The second parameter is a **pointer** to a **string** that will form the **message** inside the **window**. The third parameter is a **pointer** to a string that puts a message in the windows title bar. The last parameter is used to specify the **style** of the window.

SAQ 12.2

What would happen to the results if 16 bits or 32-bits were used to store each integer?

The range of possible number would be larger, but also the numbers produced when the numbers are left shift could be different for the same starting value.

For example x=255 and then left shift 3 places

For 8-bits binary number initially is 11111111 after shifting it is 11111000 giving a decimal number of 248.

For 16-bits binary number initially is 0000000011111111 after shifting it is 0000011111111000 giving a decimal number of 2040.

**Selected tasks** Task 12.3

In prog1.c

#include <stdio.h> void cls1()

{

system("cls");

}

int add(int a, int b, int c)

{

return(a+b+c);

}

void p\_it()

{

system("PAUSE");

}

In prog1.h

void cls1()

int add(int a, int b, int c) void p\_it()

In combined1.h

#include <stdio.h>

#include <stdlib.h> int main()

{

printf("Before clearing the screen\n\n"); p\_it();

cls1();

printf("after clearing the screen\n\n");

printf("Add three numbers together %d\n\n",add(1,2,3));

p\_it(); return 0;

}

Task 12.4

In place of MB\_OK for uType place the following and write a description of what you saw?

1. MB\_OK|MB\_ICONEXCLAMATION- Puts an OK button on the screen and an exclamation mark to the left of the text.
2. MB\_OKCANCEL|MB\_ICONEXCLAMATION - Puts an OK button and an cancel button on the screen and an exclamation mark to the left of the text
3. MB\_YESNOCANCEL|MB\_ICONSTOP - Puts an YES,NO and CANCEL buttons on the screen and a red circle with a white cross (a stop symbol) to the left of the text
4. MB\_RETRYCANCEL|MB\_ICONQUESTION - Puts a RETRY button and an CANCEL button on the screen and a speech bubble with a question mark to the left of the text

Task 12.5

#define WIN32\_LEAN\_AND\_MEAN

#include <windows.h>

int WINAPI WinMain (HINSTANCE hThisInstance,

HINSTANCE hPrevInstance, LPSTR lpcmdline,

int ncmdshow)

{

MessageBox(NULL,"This is my own program","My Second Windows Program",MB\_YESNO|MB\_ICONSTOP);

return (0);

}

Task 12.6

You should have got something like this:

Figure 12.5

C does not store integers in 8-bits (1 Bytes) each number uses 4-bytes (32- bits). Unsigned means that the most significant bit is NOT used to represent whether the number is negative or positive, but has a value. This can only store positive numbers.