UCL COMPUTER SCIENCE

UCL

COMP0002 Programming Principles

Programming Notes and Exercises 1

Purpose: Programming exercises for becoming familiar with the C programming tools and writing some simple programs.

Goal: Complete as many of the exercise questions as you can. If you are keeping up, you need to do at least the core questions. The additional questions are more challenging and are designed to stretch the more confident programmers. Don't worry if you can't do them now, but be prepared to come back and try them later on.

Feedback: It is important that you get feedback on your exercise answers so that you know they are correct, that you are not making common mistakes, that the program code is properly presented and that you are confident you have solved the problem properly. To do this, get your answers reviewed by a lab demonstrator or student mentor during lab and mentor sessions.

NOTE: You should keep all your exercise answers. Add them to your Portfolio.

Introduction - Drawing Pictures

To display graphics you should download the graphics.zip file from Moodle. Create a working directory, such as one called house and copy the zip file into it. Then use the command:

```
unzip graphics.zip
```

Several files will appear in the directory: graphics.h and graphics.c, and drawapp.jar. The first two files let you write programs in C that output instructions for drawing pictures made up of lines, rectangles and other shapes. drawapp.jar is a Java program that can read the instructions and display the picture. As an example, this C code draws a simple house using lines and rectangles:

```
#include "graphics.h"

int main(void)
{
  drawRect(50,100,200,150);
  drawLine(50,100,150,25);
  drawLine(150,25,250,100);
  drawRect(130,190,40,60);
  drawRect(70,195,40,30);
  drawRect(190,195,40,30);
  drawRect(70,125,40,30);
  drawRect(190,125,40,30);
  return 0;
}
```

#include is pronounced ‘hash include’ and is known as the *include directive*. The contents of the named file is included at the point the directive appears.

What you see here is called *source code*, the textual version of the program that you create and edit.

Source code must exactly follow the grammar and syntax rules to be valid.

The program should be typed in exactly as it appears here and saved into a file called `house.c`. It is important that the file name ends with `.c` as this denotes that the content is C source code. An editor like Emacs (or Atom, or Sublime Text, or Visual Studio Code) should be used to create and edit the source code. Moreover, the source code must be saved as simple text, not in a specialised format like a Microsoft Word file. Emacs and other source code editors work by default on simple text, and any other tool you use should do the same (so don't use Notepad, Word, Textedit or similar).

The house program consists of a sequence of *function call statements* to draw lines and rectangles. The `#include` line at the start includes the file containing the *declarations* for the *drawing functions* named `drawRect` and `drawLine`. Functions have to be *declared* before they can be used in a program and the file `graphics.h` that is named by the include contains the declarations for the drawing functions. You can look at the file yourself to see what is there.

Declaring something means naming it before it is used, to identify what kind of thing the name is for.

The `drawLine` function has four *parameters*. A parameter is a value passed to the function, for the function to work with. Think of a mathematical function like cosine, where you have to supply a value to find the cosine of, for example: `cos(25)`. The value 25 is a parameter or argument value passed to the cosine function, and we say that cosine takes one parameter.

The `drawLine` and `drawRect` functions both have four parameters, forming a *parameter list*. For `drawLine` the parameters give the (x,y) co-ordinate of the start of the line and the (x,y) co-ordinate of the end of the line. `drawRect` also has four parameters, the (x,y) co-ordinate of the top left corner, then the width and the height.

Notice that the program itself is also a function, called `main`. As it is a function that we have written you can see the entire function, referred to as the *function body*. Each line inside the function body is a *statement*, carrying out one step of the program. All programs must have a function called `main`, as this is the first function called when the program is run. If `main` is missing an executable program cannot be created.

Source code has to be converted into executable code by a tool called the *compiler*. This checks that the source code is valid and then generates an equivalent program in the instruction code of the processor in your computer. This checking and conversion process is called *compilation* and we talk about *compiling* a program.

The house program is compiled using the command:

```
gcc –o house house.c graphics.c
```

For this to work correctly you should have the files `house.c`, `graphics.c` and `graphics.h` all in the same directory. The compilation command line does not include the file `graphics.h`, just `graphics.c`. Header or `.h` files are included in a `.c` file and the `.c` file is then compiled, a `.h` file is never compiled directly.

If the code compiles without error you will see that a file called house appears. This is an *executable file*, meaning that it contains the instruction code needed for the processor to run the program. If you do get error messages then check the source code carefully for mistakes. Even one missing or incorrect character can cause an error.

You can run the program using the command `./house` and you will see this output:

```
DR 50 100 200 150
DL 50 100 150 25
DL 150 25 250 100
DR 130 190 40 60
DR 70 195 40 30
DR 190 195 40 30
DR 70 125 40 30
DR 190 125 40 30
```

> The **./** in **./house** means run the program **house** found in the current directory. On some computers you may be able to just use **house** on its own but it is good practice to use **./**.

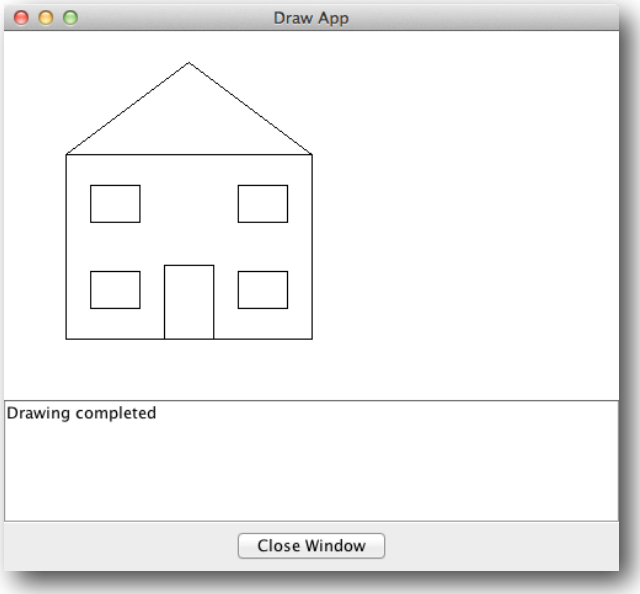
These are the instructions for drawing the picture in a simple *Domain Specific Language* or DSL. What this means is that the house program doesn't draw a picture itself but instead creates a series of instructions that another program will use to do the actual drawing. This reflects a common Unix strategy of breaking a problem into steps where each step is carried out by a separate program.

To see the picture, use this command:

```
./house | java -jar drawapp.jar
```

The vertical bar is the bar symbol on the keyboard. This command means run the `house` program and send its output (the DSL code) to the Java program `drawapp`. The `drawapp` program is able to display graphics and draw pictures (one reason for this is that it is much easier to display graphics in Java than C).

When run this window will appear on the screen:



Remember that you used the command:

```
gcc -o house house.c graphics.c
```

to compile this program. The `-o house` part of the command specifies the name of the executable file, which is why it was called `house`. You could specify the name to be any valid filename, but it is a very good idea to use names that are related to what the program does; it perhaps would have been better to choose the name `drawAHouse`. The `-o` is an example of a *command line argument* used to specify how some part of the command being run should behave, in this case to give the executable file the name you want.

The house program could also be compiled using this shorter version of the command:

```
gcc house.c graphics.c
```

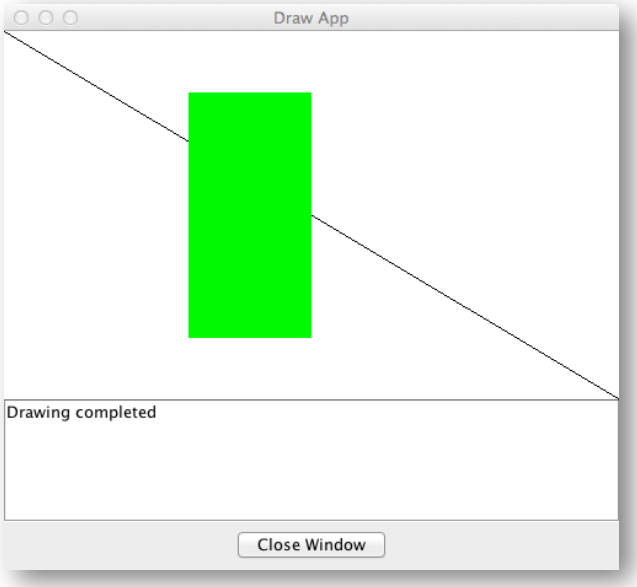
Here the -o is not used, so the name of the executable file will be given the default, which is `a.out`. You can run it using `./a.out`. You might ask why the name `a.out`? Good question! The answer is buried in Unix history; do some searching to see if you can find out why.

Here is another simple example program:

```
#include "graphics.h"

int main(void)
{
  drawLine(0,0,499,299);
  setColour(green);
  fillRect(150,50,100,200);
  return 0;
}
```

This makes use of the functions `drawLine`, `setColour` and `fillRect` and displays this:



The source code should be saved in a file named `picture.c` (or whatever name you want providing it ends with .c) and then compiled and run in the same way as the `house` program but using the different names. As before, the `graphics.c` and `graphics.h` files must be in the same directory.

Drawing lines works much the same way as drawing on a piece of graph paper. Notice that the diagonal line slopes from top to bottom, left to right. The co-ordinate of the top left of the window is (0,0), while the co-ordinate at the bottom right is (499,299). Based on this you should be able to work out how to draw a line in any position (just like drawing on graph paper, except you have to remember that (0,0) is at top left not bottom left).

The size of the drawing area is 500 pixels in width and 300 in height.

Other Drawing Functions

What else can be drawn? Here is a list of some of the possibilities:

- `drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)`. Draw an arc inside the area defined by the rectangle with top left corner at (x,y) and size given by `width` and `height`. `startAngle` is the angle in degrees that specifies where the arc starts (0 is the 3 o'clock position). `arcAngle` is the angle in degrees that specifies the end of the arc, relative to `startAngle`.
  Don't follow this? Write a test program and experiment!!

- `drawOval(int x, int y, int width, int height)`. Draw an oval inside the area defined by the rectangle with top left corner at (x,y) and size `width` by `height`.
- `drawRect(int x, int y, int width, int height)`. Draw a rectangle with top left hand corner at (x,y) and size `width` by `height`.
- `drawString(char*, int x, int y)`. A string is a sequence of characters (words, sentences, paragraphs and so on). This method displays a string at the position (x,y), providing a way of 'drawing' characters. Quotes are used to denote a string so that the characters can be distinguished from other pieces of source code, for example: `drawString("Hello",40,40)`.
- `setColour(colour)`. Set the colour used to draw with. The list of available colours is: black, blue,cyan, darkgray, gray, green, lightgray, magenta, orange, pink, red, white, yellow. To set a colour just use the name, for example: `setColour(green)`.

Example Question and Answer

Example 1. Write a graphics program to display an octagon.

Problem solving steps:

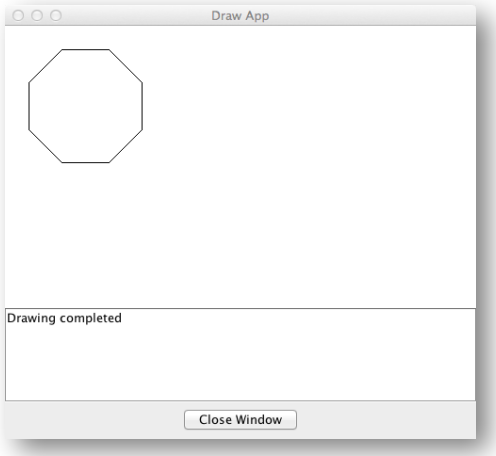
1. Do you understand the question? Read it again to make sure you understand what you are required to do. Don't waste time answering the wrong question!

2. How many sides does an octagon have? Answer: 8.

3. To draw an octagon you need to make two decisions:
i) How large it will be, and hence the length of each side.
ii) The position it will be drawn at.

4. Then you need to work out the coordinates of where the lines representing each side of the octagon start and end. Good old trigonometry comes into play to determine the coordinates. But remember that an octagon is a symmetric shape so only a small number of coordinates need to be calculated the hard way, the rest are just offsets from those already calculated. Always look for short-cuts!
It would be a good idea to double check that you can work out the co-ordinates yourself — have I got those in the example code correct? Don't assume I have!

5. Write the program with the sequence of statements needed to draw the shape. When run it should look like this:



6. If the result is not correct then double check everything. If there is no obvious problem, go back to step 1 and work you way through the steps again. Question and check everything. All too often the thing you think must be correct is wrong!

My program to draw the octagon is:

```
#include "graphics.h"
```

```
int main(void)
{
  drawLine(60,25,110,25);
  drawLine(110,25,145,60);
  drawLine(145,60,145,110);
  drawLine(145,110,110,145);
  drawLine(110,145,60,145);
  drawLine(60,145,25,110);
  drawLine(25,110,25,60);
  drawLine(25,60,60,25);
  return 0;
}
```

Notice that each *statement* is on a separate line. Statement is the name used for a syntactically complete section of code. Each of the `drawline(...)` is a statement.

Program Presentation

The text of a program, called the the source code, should always be presented neatly and made as readable as possible. Source code is primarily read by people (you!) so make sure it communicates effectively.

- Don't attempt to cram everything into the minimum amount of space — it doesn't make any difference to the speed of the program when it is run!
- Use blank spaces and empty lines to lay out the program source code neatly.
- Use indentation to reflect the structure of the program.
- Line up the start of lines and opening/closing brackets.

Comments

Source code can be mixed with comment lines that start with '//', like this:

```
// This is a comment
```

A comment is not part of the source code and is ignored by the compiler and when a program is run. Comments are just that, textual comments about the source code.

Adding comments to your code requires careful thought. Your primary aim is to make the C source code as readable and clear as possible without using comments. If comments are used then they should *add* information to the source code, not repeat what it already says. Don't use comments to state what is obvious from reading the code and don't use comments to attempt to make poorly written code more understandable. Source code is meant to be read and understood by humans; learn to write well.

What comments might you include? Often it is useful to have a comment at the start of the program stating what the program does, who wrote it and when it was written or last modified. Also a comment can be used to add additional information, for example 'this code implements the algorithm described in some book on page nn'. This is the kind of information that can't be cleanly expressed in source code and justifies the use of a comment.

Absolutely never (and I mean never!) use the style of commenting each line of source code, as advocated by some (especially for A-levels). This is a disaster, looks horrible and is hard to read. Worse, the comments will not be properly updated if the code is modified and makes changing the code an unnecessary chore.

Use comments economically and make sure any comments you do add are kept up to date.

If you ask the demonstrator to give feedback on messy or badly commented code, you will be asked to tidy it up first!

Core questions - Answer these questions yourself

Q1.1

a) Type in, compile and run this program:

```
// Written by <put your name and current date here>
// This program displays my name.

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

int main(void)
{
  char myName[] = "A Person";
  printf("Hello, my name is %s\n", myName);

  int index = strlen(myName) - 1;
  while (index > -1)
  {
    printf("%c", myName[index]);
    index = index - 1;
  }
  printf("\n");
  return 0;
}
```

The source code should be saved in a file named something `.c`, such as `q1.c`. The program consists of a single function, which must be named `main` as this is the function that is called first when the program is run. To compile the code use `gcc q1.c`.

The line `char myName[] = "A Person";` defines a *character array variable* called myName. An array is a sequence of values that can be accessed in turn using *indexing*. In this case the values are characters, so the sequence of characters can be used to store a name. A character array is often referred to as a *string* (i.e., string of characters), and text in double quotes, like `"A Person"` is a string constant. The phrase *string handling* is used to describe code or programs that work with strings.

`printf` is a *function* used to display a string on the computer screen. A statement using `printf` is referred to as an output statement. `printf` takes two arguments, which determine what to print and how to print it. The first argument is a formatting string that determines how to format the second argument string when it is printed.

The arguments to the first call to printf are (`"Hello, my name is %s\n", myName`). The formatting string contains normal text that will be printed, along with `%s` which marks where the second argument will be placed. The second argument is the string olding the name, so %s is the marker used to position the value of a string into the output. The `\n` characters mean output a newline.

The second print statement outputs a single character and the formatting string contains `%c`, which is the marker for a character.

The function call `strlen(myName)` returns the number of characters in the name. The *while loop* then iterates through the characters in reverse order, printing out the name backwards. The expression `myName[index]`, is a indexing operation used to access a specific character within the sequence of characters storing the name.

Notice that at the start of the program there are two include directives, one to include standard input/output functions (`stdio.h`), and the other to include the functions to work with strings (`string.h`). The program uses the `printf` and `strlen` functions, which are part of the standard C libraries (code that provides standard functions to be added to any program that needs them).

Once you have compiled and run the program successfully, edit the comment at the start of the program so that it uses your name and the current date. Also edit the string `'A Person'` to your own name.

Have you started to organise your filestore? Where did you save the .c file?
For each set of exercises it would be a good idea to create and use a new directory.

If you have made no mistakes typing in the source code the program will compile without complaint. More likely you will have made one or more mistakes and you will see one or more error messages. Don't Panic! It is a fact of life that you will frequently see error messages when you compile and run your programs and, certainly at first, the error messages can be confusing and difficult to understand. Go back and check your source code carefully and try again.

Start familiarising yourself with the content of error messages and learn how to read them. This will greatly help you to find and fix errors more quickly. Once you have got the program working try editing it to deliberately introduce errors. For example, remove a plus sign, delete a bracket or change the spelling of one of the words making up the program. Observe the error messages you get when you try to run the modified program. At first they may not make too much sense but will become familiar! Learn what each error message means and how to resolve the problem reported.

After you have finished experimenting, restore the program to its working condition.

Q1.2 Modify the program in question 1.1 to display the address of the Computer Science Department rather than just your name. Each line of the address will need to be stored in a separate variable. For example,

```
char name[] = "Dept. of Computer Science";
char address1[] = "Malet Place Engineering Building";
etc.
```

Don't forget that you use the printf function to print strings.

You will need to give your program a new name and save it to a new file, otherwise you will overwrite your answer to Q1.1.

Q1.3 Type in the octagon drawing program presented earlier. Save it to a file and then run the program. Make sure the shape displayed is the same as the one in the picture. Depending on how good the aspect ratio of your computer screen actually is, the octagon you see may or may not appear to be completely regular.

Q1.4 Modify your program to draw a heptagon (7 sides). Don't forget to save it to a new file, otherwise your answer to 1.3 will get overwritten. Make sure the heptagon is drawn accurately, don't just guess at the co-ordinates so it seems to look correct! Hint: Heptagons are surprisingly tricky to draw, check Wikipedia for more information.

Q1.5 Write a program to draw two rectangles:
One with with top left corner at (30,30) and horizontal sides of length 90 and vertical sides of length 50, and another with top left corner at (150,50) and horizontal sides of length 60 and vertical sides of length 140.

Draw the first rectangle with drawLine and the second with drawRect.

Q1.6 Write a program to draw a square inside a circle, such that the corners of the square touch the circle.

Use drawArc to draw the circle.

Q1.7 Write a program to draw a representation of the BT Tower (many people still call it the Post Office Tower), this is the tall thin round tower visible from many parts of the main UCL campus.

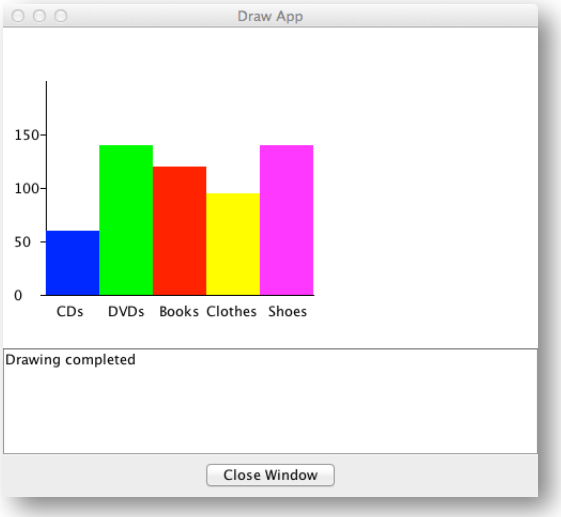
This will need a longer sequence of drawing statements. Plan your drawing before trying to write your code.

Q1.8 Write a program to draw a series of ovals of increasing size. Use `drawOval` for this.

Q1.9 Write a program to display the CS department address (see the CS web page at http://www.cs.ucl.ac.uk) in a drawing. Each part of the address should appear on a separate line in the normal way.

Use `drawString` to do this.

Q1.10 Create a program to draw a bar chart with labels. This will need quite a long list of drawing statements. Here is an example:



Q1.11 Write a program to draw a sine wave. The C math library provides a sine function with this *signature*: `double sin(double x)`. This means it takes a double, or floating point, value as an argument and returns a double result. To use the function in your code, the statement

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

needs to appear at the top of you source code file. The function works with *radians*, so you either need to work with radians yourself or use degrees and convert to radians when calling the `sin` function (multiply by pi/180).

Double values produced by the `sin` function needs to be converted to integers, type `int`, when passed as values to drawing functions. This can be done using a `cast` expression:

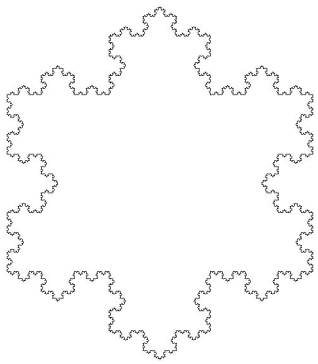
```
double y = 1.23;
int x = (int)y;
```

The value 1 will be assigned to x, as the digits after the decimal point are discarded.

Challenge Questions

These questions are challenges. Try searching on the web for information - there are lots of interesting websites and examples to be found.

Q1.12 Write a program to draw a fractal shape using a recursive algorithm, like this one:



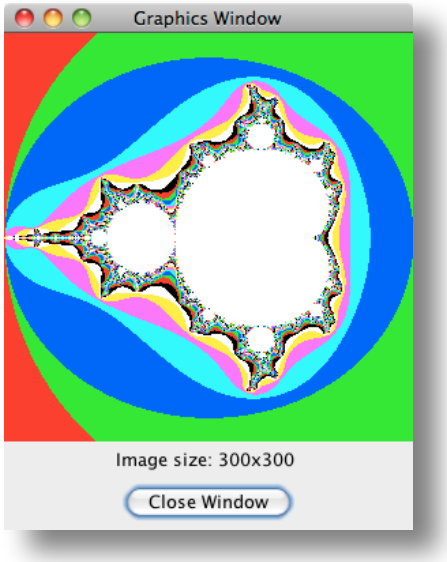
Note that the shape of the sub-elements reflect the shape of the whole. See https://en.wikipedia.org/wiki/Koch_snowflake for more information.

Q1.13 Write a program to display the Mandelbrot Set in colour.

Drawing a basic Mandelbrot is actually quite straightforward and needs only a relatively few lines of code. When working on this question get a basic working program first and then extend it.

A Mandelbrot looks like this:

Search the web for an explanation of what a Mandelbrot is and how to display one (e.g., https://en.wikipedia.org/wiki/Mandelbrot_set).



Q1.14 Write a program that can test if a positive integer number is a prime number. It should be able to work with numbers up to 100 digits in length. Such large numbers cannot be represented directly in C as a single value using a built in integer type such as `int` or `long`. You will need to find a way of representing very large numbers, and then how to add, multiply or divide them (do you need to multiply and divide though?).