UCL COMPUTER SCIENCE

COMP0002 Programming Principles

Programming Notes and Exercises 2

(Complete the core and drawing questions by Monday 19th Oct to be on track)

Purpose: Writing programs that use iteration, loops, selection and variables.

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. Some 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 TA during lab sessions.

NOTE: Keep a copy of all your exercise answers.

Introduction

These exercises require you to write programs that use iteration, selection and variables.

C has several kinds of iteration:

a) The while loop

```
while (boolean-exp)
{
// Loop body
}
```

The loop body can be executed zero or more times. "boolean-exp" means an integer expression that evaluates to zero (false) or not zero (true). When you write a loop you provide a boolean expression appropriate to the loop you are trying to write.

b) The for-loop

For loops are typically used for counting from an initial value to a final value determined by the value of a boolean expression, with the value updated by an expression at the end of each loop iteration.

```
for ( initialisation ; boolean-expression ; expression)
{
  // Loop body
}
```

For example:

```
int counter = 0;
for (counter = 0 ; counter < 10 ; counter++)
{
  printf("Hello %i\n", counter);
}
```

`counter = 0` evaluated once at start of loop

`counter < 10` evaluated before each evaluation of the loop body

`counter++` evaluated after each evaluation of the loop body

The while and for loops have a body bracketed by braces (curly brackets). Make sure you always include the braces. Loop bodies can contain any statement sequence, including nested loops.

c) The do loop

```
do
{
  // Loop body
} while (boolean-expression);
```

The loop body is always evaluated at least once as the boolean condition is evaluated after the body.

Remember that loops, and certainly those needed by these exercises, must have a properly determined *termination condition*. This means that one or more statements in the loop body should have some affect on the boolean expression controlling the loop execution. The expression should evaluate to 0 or false within a reasonable number of executions of the loop body.

Selection is provided by the `if` statement:

```
if (boolean-exp)
{
  // Statements executed if boolean-exp is true
}
```

This allows the conditional execution of a statement sequence. A variation is the if-else statement:

```
if (boolean-exp)
{
  // Statements executed if boolean-exp is true
}
else
{
  // Statements executed if boolean-exp is false
}
```

This allows a choice between two different statement sequences.

Boolean expressions are created using relational and boolean operators. Relational operators are used to compare values such as numbers:

less than	greater than	equal to	less than or equal to	greater than or equal to
`a < b`	`a > b`	`a == b`	`a <= b`	`a >= b`

Boolean operators combine boolean expressions:

`and`	`or`	`negation (not)`	`a && b`	`a || b`	`!a`

Operators can be combined to create more complex boolean expressions:

```
 a < 0 && b > 2
```

but you need to make sure that the operators are evaluated in the order you expect. If in any doubt bracket the sub-expressions:

```
(a < 0) && (b > 2)
```

Variables allow values to be stored and manipulated. Variables should be declared and initialised before being used:

```
int x = 0;
double b = 1.234;
```

The variable name should reflect the use of the variable, so one letter variable names such as those above are not ideal! Having said that, loop counter variables are often given a single character name. A variable name must start with a letter (a-z), followed by any combination of letters and numbers. Punctuation symbols, expect for underscore, are not allowed, and there cannot be spaces in a name. By convention, variable names should start with a lowercase letter.

Numeric variables (e.g., integers) can be incremented (increased by one) using the operator ++ and decremented (decreased by one) using `--`:

```
x++;  // increment x
y--;  // decrement y
```

Maths Functions

Several of the questions below, and questions in future exercises, require the use of maths functions such as sin, cos or sqrt (square root). The standard C library provides implementations of these, see here for the documentation for the GNU C version: http://www.gnu.org/s/hello/manual/libc/index.html.

To use a maths function you should add the following line at the top of your source code file:

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

When you compile your code you may need to add an additional *flag* -lm to tell the compiler to link the maths library code with your code (some versions of gcc won't need this).

```
gcc -o graph graph.c graphics.c -lm
```

This means that the code implementing the maths functions gets added to the executable program you are creating. If the flag is missing then gcc will give linker error messages stating that there are undefined symbols as it will not be able to locate the function definitions (code) needed.

Example Questions and Answers

The following are examples of questions and answers, to show how the C syntax works, and to illustrate the kinds of programs you should be writing. Pay very careful attention to the layout of your program and the use of indentation. You will be asked to tidy up poorly presented code.

Remember, source code is primarily for people to read and understand.

Example 2.1 Write a program using a while loop to print out a message 10 times. Each message should be on a separate line using this format:

```
1: A message
2: A message
3: A message
and so on...
```

Numbering should start from 1. Substitute whatever message you like.

Answer:

Take care to get all those loop conditions and counter increments correct. Easy to make mistakes...

```
#include <stdio.h>

int main(void)
{
  int n = 1;
  while (n < 11)
  {
    printf("%i: A Message\n",n);
    n = n + 1;
  }
  return 0;
}
```

You can also have less have verbose variations such as:

```
#include <stdio.h>

int main(void)
{
  int n = 0; // Note, initialised to zero
  while (n++ < 10) // Note condition
  {
    printf("%i: A Message\n",n);
  }
  return 0;
}
```

With both programs note the use of indentation and the way braces have been lined up. Use of indentation, blank space and blank lines has no effect of the execution of programs, but their proper use greatly increases readability and makes mistakes easier to see.

Example 2.2 Repeat Example 2.1 using a for-loop.

Answer:

```
#include <stdio.h>

int main(void)
{
  int n = 0;
  for( n = 1 ; n < 11 ; n++)
  {
    printf("%i: A Message\n",n);
  }
  return 0;
}
```

Watch out for the loop counting, start and end conditions. Note also that in this example the initialisation of n when it is declared is redundant.

Example 2.3 Repeat Example 2.1 a do loop

```
#include <stdio.h>

int main(void)
{
  int n = 1;
  do
  {
    printf("%i: A Message\n",n);
  } while (++n < 11);
  return 0;
}
```

Look at the boolean expression at the end of the loop. Is it correct? Can it be written a different way?

Example 2.4 Write a program using loops to display the following:

```
****
****
****
****
```

You may only print one character at a time.

Hint: nested loops.

Answer:

Note that you can only output characters from left to right across a line, and each line must be complete before starting the next. You cannot move the output position around to output characters in an arbitrary order.

```
#include <stdio.h>

int main(void)
{
  int row;
  for (row = 0 ; row < 4 ; row++)
  {
    int column;
    for (column = 0 ; column < 4 ; column++)
    {
      printf("*");
    }
    printf("\n");
  }
  return 0;
}
```

This example used a *nested for loop* to count through the columns in a row (stars in a line). As a for loop is a statement it is entirely possible to have a for loop contained in the statement sequence of another for loop body. Note that the variable column is declared inside the outer for loop and that the variables are not initialised in their definitions as they are initialised by the for loops.

You may have noticed that all the examples above have:

```
return 0;
```

as the last statement. Why? The `main` function returns a value to the shell, denoting the final state of the program. Zero means the program terminated normally, non zero means some error occurred. Use the command:

```
echo $?
```

to see the value returned by the last command or program run.

Core Questions

You should answer all the following questions! In all these questions you should be using while or for loops, or iterations and closures (try them all).

Q2.1 Write a program using a while loop to display the thirteen times table, like this:

```
1 * 13 = 13
2 * 13 = 26
3 * 13 = 39
```

and so on.

Note: the multiplication operator is *.

Q2.2 Repeat Q2.1 using a for loop.

Q2.3 Repeat Q2.1 using do loop.

Q2.4 Write a program to display the following:

```
*****
*   *
*   *
*****
```

You may display only one character at a time. You cannot have a statement such as:

```
printf(“*****”);
```

to output an entire line at once. A loop must be used to write characters one at a time. Space characters should be output to display the spaces inside the shape.

Hint: nested loops.

Q2.5 Write a program to display the following:

```
******
 *****
  ****
   ***
    **
     *
```

You may display only one character at a time.

Q2.6 Write a program to display the following:

```
 *****
  *****
   *****
    *****
     *****
    *****
   *****
  *****
 *****
```

You may display only one character at a time.

Q2.7 Write a program to display the following:

```
********
*      *
* #### *
* #  # *
* #### *
*      *
********
```

You may print only one character at a time.

Hint: Find out about the if statement.

Q2.8 Write a program to display the following:

```
*#*#*#
#*#*#*
*#*#*#
#*#*#*
*#*#*#
#*#*#*
```

You may display only one character at a time.

Q2.9 Write a program to display the following:

```
*
**
* *
*  *
*   *
*  *
* *
**
*
```

You may print only one character at a time.

Q2.10 Write a program to display the following:

```
*******
#*    *
##*   *
# #*  *
#  #* *
#   #**
#    #*
#######
```

You may display only one character at a time.

Q2.11 Write the following programs:

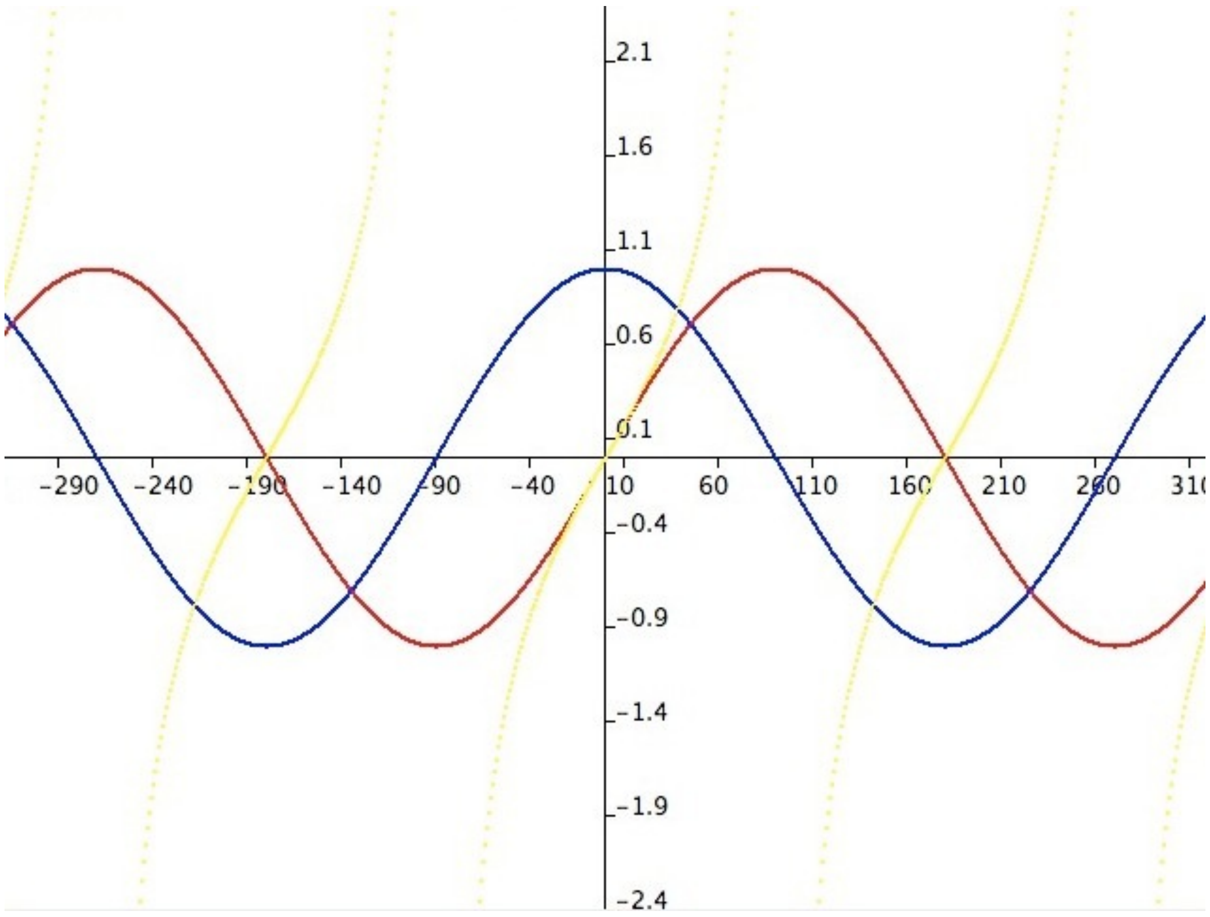
a) To print the squares of the numbers from 1 to 100.

b) To print the squares of the even numbers between 1 and 101.

c) To print the prime numbers between 1 and 100.

Drawing Questions

Q2.12 Write a drawing program that draws a graph showing the curves y=sin(x), y=cos(x) and y = tan(x). Include properly labelled axes, to look like the example below:



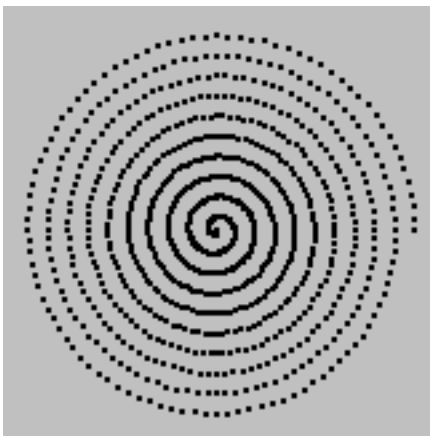
Choose suitable ranges for the axes of the graph to show the curves to the best effect.

Hints: Break the drawing of the graph down into a series of steps: draw the x-axis, label the x-axis, draw the y-axis, label the y-axis, etc. Use loops to draw the curves using points. A point can be drawn using a 1x1 rectangle. Remember that on the window co-ordinate system (0,0) is top left, so you will have to take care when you compute the coordinates of lines and points.

How do you compute Sin, Cos, Tan? See the functions in the standard C library. For example, double sin(double) will return the sin of double value (note value needs to be specified in radians or convert degrees to radians).

Q2.13 Write a drawing program to plot a spiral using a series of points, like this:

Hint: If you can draw a circle, just keep increasing the radius.



Challenges

Q2.14 Write a *function* to display rectangles of any character like the following:

```
*****
*****    %%%%%%%%%
*****    %%%%%%%%%
***** or %%%%%%%%%
```

The function parameters should give the number of rows and columns, and the character to use.

Use the function to display various rectangles of different sizes.

Q2.15 Write a drawing program that includes functions for drawing rectangles and triangles of any size and at any location. Draw a picture using the functions.

Q2.16

a) Write a program that takes the digits 123456789 and inserts + or minus signs between the numbers such that the expression adds up to 100. For example:

1 + 2 + 3 - 4 + 5 + 6 + 78 + 9 = 100

The digits must remain in order, + or - is not required between every digit, as illustrated by the example. Find *all* the expressions that give 100.

b) Extend your program so that it can find the expressions that evaluate to any number you specify.

c) Investigate what happens if the digits can be in any order, and/or you allow multiplication and division.

If you have got this far and are looking for more advanced exercises to work on then look for exercises in C text books, in particular try C Lab 2: OpenCV in the book Head First C, end of chapter 8. Also work through the exercises in the K&R book (The C Programming language by Kernighan and Ritchie).