Lab Session 1 : The Basics (C)

Some Practice Examples

With your experience of C (G51PRG) and subsequent experience with Java, you should be able to cope with these examples. For each of the first few examples, copy the code into a file, compile it and run it to verify that it works as you expect. Make sure that you understand each of them.

The last part is a practice exercise to try, giving you experience in writing your own program.

All of this practice will be useful in your informal coursework and will show you key things which you will find useful for both the formal coursework and the exam.

Loops

`for` loops work in the same way as for Java and C:

```
#include <cstdio> /* for printf */

int main( int argc, char* argv[] )
{
   int x = 1;
   for ( x = 2; x < 10 ; x++ )
   {
      printf("X is %d\n", x);
   }
   return 0;
}
```

Task 1: Compile and execute the code sample above.

Task 2: Change the above program to use a `while` loop instead of the `for` loop.

Task 3: Change the above program to use a `do` ... `while` loop instead of the `for` loop.

Conditionals

`if` statements, `switch` statements and ternary operators (`? :`) are also the same as for Java and C:

```
#include <cstdio> /* for printf */

int main ( int argc, char* argv[] )
{
   int x = 2;

   if ( x == 4 )
      printf( "X is 4\n" );
   else
```

```
      printf( "X is not 4\n" );

   printf( (x==4) ? "X is 4\n" : "X not 4\n" );

   switch( x )
   {
   case 4: printf("X is 4\n"); break;
   default: printf("X is not 4\n"); break;
   }

   return 0;
}
```

Task 4: Compile and execute the above code sample. Experiment by changing the value of x and verify that it works as you expect.

Random numbers

There are many functions available in the standard C library and many of these will be useful to you when you do the coursework later. Header files are available which include the declarations of the standard functions - you need to include the relevant header file to avoid compilation warnings.

```
#include <cstdlib> /* for rand and srand */
#include <cstdio> /* for printf */
#include <ctime> /* for time */

int main( int argc, char* argv[] )
{
   /* srand(1); */
   /* srand(2); */
   /* srand(3); */
   /* srand( time(0) ); */

   printf( "Roll %d die: %d\n", 1, rand() % 6 + 1 );

   return 0;
}
```

Task 5: Compile and execute the above code sample. What does it do? Execute it multiple times to understand better.

Task 6: There are two `%d` fields in the string in the `printf()` function. What does the `%d` do in the `printf()` function? Feel free to check the documentation for `printf()` if you are not sure. `printf()` is an extremely powerful function with many different optional fields. We will see more about it later.

Task 7: What does the `rand()` function do? Execute the program multiple times if you are not sure. Again, feel free to look up the documentation after you have tried to work it out for yourself.

Task 8: `srand()` is another useful function. Uncomment out the first line (“`srand(1)`”) so that `srand()` is executed with the parameter 1. Execute the program multiple times. What happens?

Now try uncommenting the other `srand()` lines one at a time. What difference do they make?

What does `srand();` do? Check the manual pages of the online module book to see whether you are correct. Or talk to the lab helpers.

Convert between strings and integers

Create a file called ‘`atoidemo.cpp`’ with the following contents:

```
#include <cstdio>
#include <cstdlib>
int main( int argc, char* argv[] )
{
   int max=0, i=0;
   if ( argc < 2 )
   {
      printf("Too few parameters provided.\n");
      return 1; // Failure
   }
   max = atoi(argv[1]);
   for (i = 0 ; i < max ; i++ )
      printf("Line %d\n",i);
   return 0; // Success
}
```

Compile the program to form the executable called ‘`atoidemo`’, in the usual way.

Execute it with the following command line:

```
bann$ ./atoidemo 4
```

Question: What does the program do?

Question: What is the output of the atoidemo program when you execute the program with the following command line:

```
a) atoidemo

b) atoidemo 1

c) atoidemo 5

d) atoidemo fifteen

e) atoidemo x13

f) atoidemo 1.3

g) atoidemo 4.999
```

Question: What effect is the value of the command line argument having?

Question: How can a string command line argument be converted into an integer?

Question: What is the effect if the command line argument does not represent an integer number?

Formatting your output : example and exercise

Note: this extends what you may know about field sizes in `printf()`.

Create a file called `formatted.cpp` with the following contents:

```
#include <cstdio>

int main( int argc, char* argv[] )
{
   int j = 0;
   for ( j = 1 ; j < 25 ; j++ )
   {
      printf("Square of %3d is %03d\n", j, j*j);
   }
   return 0; // Success
}
```

Compile the program into an executable called '`formatted`' and run the executable without arguments.

Experiment with the `printf` function, to learn some of the possibilities:

Exercise:

Change the `printf` function in `formatted.c` to each of the following, one at a time. Each time compile and execute the program and note the effects:

```
a) printf("Square of %-3d is %03d\n", j, j*j);

b) printf("Square of %d is %d\n", j, j*j);

c) printf("Square of %2d is %02d\n", j, j*j);

d) printf("Square of 0x%3x is 0x%03x\n", j, j*j);
```

You may wish to read `http://en.wikipedia.org/wiki/Hexadecimal` if this last example is not clear.

1 The main exercise - a more useful program

Please try the following exercise. Doing this will help you to ensure that you understand the basics of C programming before you continue. You will find some hints, help, and comments about the C compiler in sections 2 to 4.

The compiler will give you a lot of feedback about how well you are doing automatically, especially if you turn on extra warnings (see section 4.1). i.e. it will tell you about mistakes (errors) and potential mistakes (warnings). You should also ensure that you talk to the lab demonstrators if you need help.

This exercise involves writing a small program in C to output a formatted table (such as a multiplication table), for a specified mathematical operator. The entire program can consist of a single function of very few lines if you so desire. You do not need to attempt to divide it into functions or structure the code at this point.

This exercise will test your ability to use command line arguments, use C strings and format output. You will need to know how to use `printf()` to output formatted text, how to extract a character from a string (to get the operator to use for the table, e.g. `+` or `*`) and how to use a `switch` statement to change the operator that is used.

1.1 Exercise details

Produce a program to fulfil the following requirements:

1. Your program should accept command line arguments specifying which operator to use and the size of the table.
2. The following eight operators should be supported:

Operator	Meaning	Example
`+`	Addition	`7+3=10`
`-`	Subtraction	`7-3=4`
`*`	Multiplication	`7*3=21`
`/`	Integer division	`7/3=2`
`%`	Modulo (or integer remainder)	`7%3=1`
`&`	Binary AND	`7&3=3`
`|`	Binary OR	`7|3=7`
`^`	Binary XOR	`7^3=4`

   Note: The *, & and | symbols will need to be supplied within quotes to avoid the shell interpreting it as a wildcard symbol. i.e. `"*"`, `"&"` or `"|"`. The shell will remove the surrounding `""` from the argument, so your program will just receive the symbol between the quotes.

   Please see the examples at the end of these instructions.

3. Your program only needs to deal with integer numbers. No floating point values are required at any stage.
4. You should accept three command line arguments. The first should be a single character and will denote the operation to perform. The second should specify the number of rows in the output table. The third should specify the number of columns in the output table.
5. If fewer than three command line arguments are supplied then your program should display an error message and then terminate.
6. If the first command line argument is not one of the supported operators, then display a warning to the user and end the program.

You may, if you wish, just validate the first symbol in the first parameter, and ignore any trailing symbols. e.g. You may, if you wish treat the argument "`*ert`" as "`*`", ignoring all but the first character of the string.

7. Identify the number of rows to display. If this is less than one then display a message to the user and end the program.
8. Identify the number of columns to display. If this is less than one then display a message to the user and end the program.
9. You can assume that, where there are multiple problems with the supplied command line arguments, it is sufficient to describe a single error. e.g. if someone supplies a number of columns and rows of zero each then it is sufficient to report that one of them is incorrect, rather than needing to report all errors at once.
10. Display a table with the correct number of rows and columns.
11. The row number should run from one to the required number of rows, in increments of one.
12. The column number should run from one to the required number of columns, in increments of one.
13. The value in the cells should show the result of applying the operator to the values in the row and column, with the row first then the column. i.e. calculate [row-value] [operator] [column-value] and put the value into the cell.

    For example, with operator +, in the row for the value `3`, and the column for the value `6` you should insert the value `9`, since `(3 + 6 = 9)`.
14. You should allow enough room for four digit numbers, and ensure that there is still a gap between the columns if four digit numbers are displayed.
15. You should not use the tab character to pad columns. Using this character will make the table format dependent upon the tab spacing. Instead you should pad the columns with leading spaces.
16. You should add a line of '`-`' characters separating the column headings from the column values.
17. You should add a column of '`|`' characters separating the row heading from the row data.
18. Where the column of '`|`' intersects the row of '`-`' you should use a '`+`' character. Please refer to the examples in these instructions if this is not clear.
19. Program output should be to `stdout`, so using a function such as `printf` is fine.

Hint: When you have a working program, test it using the command lines provided in the examples section below and verify that the output from your program is equivalent to that in the examples.

1.2 Examples of input and output

The following shows example input (in terms of command line parameters) and output from the program. Further example command lines will be used for evaluating the performance of your program.

1.2.1 Validating the command line arguments

```
bann$ ./myprog
Usage: ./myprog <operator> <rows> <columns> : too few arguments

bann$ ./myprog / 1 cols
Usage: ./myprog <operator> <rows> <columns> : must have at least one column
```

```
bann$ ./myprog / 1 0
Usage: ./myprog <operator> <rows> <columns> : must have at least one column

bann$ ./myprog / 0 2
Usage: ./myprog <operator> <rows> <columns> : must have at least one row

bann$ ./myprog / rows 3
Usage: ./myprog <operator> <rows> <columns> : must have at least one row

bann$ ./myprog ? 4 3
Usage: ./myprog <operator> <rows> <columns> : unrecognised operator '?'

bann$ ./myprog op 4 3
Usage: ./myprog <operator> <rows> <columns> : unrecognised operator 'o'
```

1.2.2 Examples with correct arguments

```
bann$ ./myprog % 3 3
  % |    1    2    3
----+---------------
  1 |    0    1    1
  2 |    0    0    2
  3 |    0    1    0

bann$ ./myprog % 4 3
  % |    1    2    3
----+---------------
  1 |    0    1    1
  2 |    0    0    2
  3 |    0    1    0
  4 |    0    0    1

bann$ ./myprog "*" 4 3
  * |    1    2    3
----+---------------
  1 |    1    2    3
  2 |    2    4    6
  3 |    3    6    9
  4 |    4    8   12

bann$ ./myprog / 6 6
  / |    1    2    3    4    5    6
----+------------------------------
  1 |    1    0    0    0    0    0
  2 |    2    1    0    0    0    0
  3 |    3    1    1    0    0    0
  4 |    4    2    1    1    0    0
  5 |    5    2    1    1    1    0
  6 |    6    3    2    1    1    1

bann$ ./myprog "&" 8 8
  & |    1    2    3    4    5    6    7    8
----+----------------------------------------
  1 |    1    0    1    0    1    0    1    0
  2 |    0    2    2    0    0    2    2    0
  3 |    1    2    3    0    1    2    3    0
  4 |    0    0    0    4    4    4    4    0
  5 |    1    0    1    4    5    4    5    0
  6 |    0    2    2    4    4    6    6    0
  7 |    1    2    3    4    5    6    7    0
  8 |    0    0    0    0    0    0    0    8

bann$ ./myprog "|" 12 10
  | |    1    2    3    4    5    6    7    8    9   10
----+----------------------------------------------------
  1 |    1    3    3    5    5    7    7    9    9   11
  2 |    3    2    3    6    7    6    7   10   11   10
  3 |    3    3    3    7    7    7    7   11   11   11
  4 |    5    6    7    4    5    6    7   12   13   14
  5 |    5    7    7    5    5    7    7   13   13   15
  6 |    7    6    7    6    7    6    7   14   15   14
  7 |    7    7    7    7    7    7    7   15   15   15
  8 |    9   10   11   12   13   14   15    8    9   10
  9 |    9   11   11   13   13   15   15    9    9   11
 10 |   11   10   11   14   15   14   15   10   11   10
 11 |   11   11   11   15   15   15   15   11   11   11
 12 |   13   14   15   12   13   14   15   12   13   14

bann$ ./myprog ^ 12 10
  ^ |    1    2    3    4    5    6    7    8    9   10
----+----------------------------------------------------
  1 |    0    3    2    5    4    7    6    9    8   11
  2 |    3    0    1    6    7    4    5   10   11    8
  3 |    2    1    0    7    6    5    4   11   10    9
  4 |    5    6    7    0    1    2    3   12   13   14
  5 |    4    7    6    1    0    3    2   13   12   15
  6 |    7    4    5    2    3    0    1   14   15   12
  7 |    6    5    4    3    2    1    0   15   14   13
  8 |    9   10   11   12   13   14   15    0    1    2
  9 |    8   11   10   13   12   15   14    1    0    3
 10 |   11    8    9   14   15   12   13    2    3    0
 11 |   10    9    8   15   14   13   12    3    2    1
 12 |   13   14   15    8    9   10   11    4    5    6

bann$ ./myprog - 12 10
  - |    1    2    3    4    5    6    7    8    9   10
----+----------------------------------------------------
  1 |    0   -1   -2   -3   -4   -5   -6   -7   -8   -9
  2 |    1    0   -1   -2   -3   -4   -5   -6   -7   -8
  3 |    2    1    0   -1   -2   -3   -4   -5   -6   -7
  4 |    3    2    1    0   -1   -2   -3   -4   -5   -6
  5 |    4    3    2    1    0   -1   -2   -3   -4   -5
  6 |    5    4    3    2    1    0   -1   -2   -3   -4
  7 |    6    5    4    3    2    1    0   -1   -2   -3
  8 |    7    6    5    4    3    2    1    0   -1   -2
  9 |    8    7    6    5    4    3    2    1    0   -1
 10 |    9    8    7    6    5    4    3    2    1    0
 11 |   10    9    8    7    6    5    4    3    2    1
 12 |   11   10    9    8    7    6    5    4    3    2
```

```
bann$ ./myprog - 12 14
  - |   1   2   3   4   5   6   7   8   9  10  11  12  13  14
----+--------------------------------------------------------
  1 |   0  -1  -2  -3  -4  -5  -6  -7  -8  -9 -10 -11 -12 -13
  2 |   1   0  -1  -2  -3  -4  -5  -6  -7  -8  -9 -10 -11 -12
  3 |   2   1   0  -1  -2  -3  -4  -5  -6  -7  -8  -9 -10 -11
  4 |   3   2   1   0  -1  -2  -3  -4  -5  -6  -7  -8  -9 -10
  5 |   4   3   2   1   0  -1  -2  -3  -4  -5  -6  -7  -8  -9
  6 |   5   4   3   2   1   0  -1  -2  -3  -4  -5  -6  -7  -8
  7 |   6   5   4   3   2   1   0  -1  -2  -3  -4  -5  -6  -7
  8 |   7   6   5   4   3   2   1   0  -1  -2  -3  -4  -5  -6
  9 |   8   7   6   5   4   3   2   1   0  -1  -2  -3  -4  -5
 10 |   9   8   7   6   5   4   3   2   1   0  -1  -2  -3  -4
 11 |  10   9   8   7   6   5   4   3   2   1   0  -1  -2  -3
 12 |  11  10   9   8   7   6   5   4   3   2   1   0  -1  -2

bann$ ./myprog % 10 14
  % |   1   2   3   4   5   6   7   8   9  10  11  12  13  14
----+--------------------------------------------------------
  1 |   0   1   1   1   1   1   1   1   1   1   1   1   1   1
  2 |   0   0   2   2   2   2   2   2   2   2   2   2   2   2
  3 |   0   1   0   3   3   3   3   3   3   3   3   3   3   3
  4 |   0   0   1   0   4   4   4   4   4   4   4   4   4   4
  5 |   0   1   2   1   0   5   5   5   5   5   5   5   5   5
  6 |   0   0   0   2   1   0   6   6   6   6   6   6   6   6
  7 |   0   1   1   3   2   1   0   7   7   7   7   7   7   7
  8 |   0   0   2   0   3   2   1   0   8   8   8   8   8   8
  9 |   0   1   0   1   4   3   2   1   0   9   9   9   9   9
 10 |   0   0   1   2   0   4   3   2   1   0  10  10  10  10
```

1.3 Hints

Only read these if you cannot see how to do something.

- A `char` holds a number. You can `switch()` on a `char`.
- A character literal is the number represented by that character, so you can use character literals in `case` or `if` statements. Alternatively, look up the `strcmp()` and `strncmp()` functions.
- A C-type string is an array, so you can extract a character from it relatively easily. Including getting the first character (which will be a `char`).
- Examine the demo lecture samples for demo lecture 2 and the following sections for clues as to how to do specific parts of the exercise.