Worksheet 1: Basics

Updated: 30th July, 2019

The objectives of this practical are:

- to revise basic programming concepts, including data types, variables, expressions and control structures; and
- to get started on simple C programming.

Pre-lab Exercises

1. Revision

Explain what each of the following C programs does:

(a) (Conditions)

```c
#include <stdio.h>

int main(void) {
    int number;
    scanf("%d", &number);

    if(number < 0 || number > 10)
        printf("Out of range\n");
    else if(number != 5)
        printf("Wrong\n");
    else
        printf("Correct\n");

    return 0;
}
```

(b) (Loops)

```c
#include <stdio.h>

int main(void) {
    int count, i;
    scanf("%d", &count);

    i = 0;
    while(i < count) {
        printf("%d ", i);
        i++;
    }
    return 0;
}
```

How could you replace the `while` loop?

(c) (Functions)

```
#include <stdio.h>

int readInt(void) {
    int result;
    printf("Enter an integer");
    scanf("%d", &result);
    return result;
}

void printInt(int x) {
    printf("The result is: ");
    printf("%d", x);
}

int main(void) {
    int a, b, c;
    a = readInt();
    b = readInt();
    c = (a + b) * (a - b);
    printInt(c);
    return 0;
}
```

2. Forward declarations

(Consult the lecture notes on forward declarations.)

Say you have a .c file containing five functions. What forward declarations must there be if:

(a) None of the functions call each other.

(b) Each function calls the next one, in order (i.e. the 1st calls the 2nd, the 2nd calls the 3rd, etc.)

(c) Each function calls the previous one (i.e. the 5th calls the 4th, the 4th calls the 3rd, etc.)

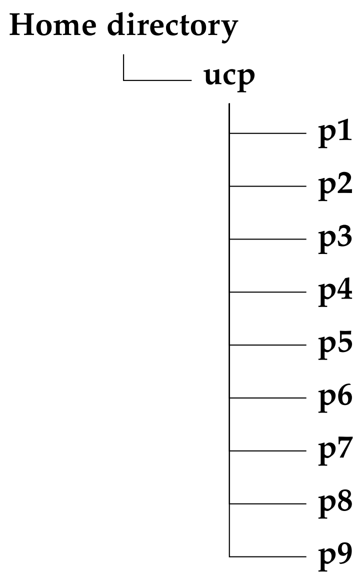
(d) The 1st and 5th functions both call the 2nd, 3rd and 4th.

(e) The 2nd, 3rd and 4th functions all call the 1st and 5th.

Practical Exercises

1. Setting Up

Before anything else, create some directories (folders) to keep your UCP practical work in:



Change into your `p1` directory once you're done.

Remember your basic UNIX commands:

`[user@pc]$ cd`	Change back to the top-level of your home directory.
`[user@pc]$ cd` **dir**	Change into the directory **dir** (which must be inside the current directory).
`[user@pc]$ cd ..`	Change to the parent directory (one level above the current directory).
`[user@pc]$ pwd`	Show the current directory ("print working directory").
`[user@pc]$ ls`	List the contents of the current directory.
`[user@pc]$ ls` **dir**	List the contents of directory **dir**.
`[user@pc]$ mkdir` **dir**	Create the directory **dir** (inside the current directory).
`[user@pc]$ rmdir` **dir**	Remove (delete) the directory **dir**.
`[user@pc]$ cp` **src** **dest**	Copy **src** to **dest** (where **dest** is either a directory or a new filename).
`[user@pc]$ mv` **src** **dest**	Move or rename **src** to **dest**.
`[user@pc]$ rm` **file**	Remove (delete) **file**.

(If you put spaces in your filenames, you will need quotes around them when using these commands.)

2. Writing Conditions

Write a short C program to do the following:

(a) Ask the user to enter two integers.

(b) Check if the first number is divisible by the second.

(c) Output the result – "divisible" or "not divisible".

(For example, if the user enters 10 and 5, your program should output "divisible". If the user enters 11 and 4, your program should output "not divisible".)

Editor:

The main text editor that is to be used for this unit is vim.

```
[user@pc]$ vim question2.c
```

On blackboard is an example config file that can be used to make it a bit easier for you. To setup such file enter

```
[user@pc]$ vim ~/.vimrc
```

And then rewrite the required lines. You are expected to follow the coding standard whether you use this config file or not.

Compile your code:

```
[user@pc]$ gcc -Wall -ansi -pedantic -Werror question2.c -o question2
```

Then run it:

```
[user@pc]$ ./question2
```

Does it do what you expect? If you run into problems, ask your tutor for help!

3. Writing Loops and Functions

Complete the following C code (saving it to a different file; e.g. `question3.c`):

```
#include <stdio.h>

int main(void)
{
    /* To be determined */
}

int factorial(int n)
{
    /* To be determined */
}
```

The `factorial()` function should calculate factorials. For instance, when given a value of 7, it should return 5040 ($7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1$). It should *not* input or output anything.

The `main()` function should:

- Input an integer from the user;
- If the number is zero or positive, call `factorial()` to compute the factorial and print out the result;
- Repeat this process until the user enters a negative number.

Compile and run your program. Make sure it works as required!

End of Worksheet