University of Mauritius

Credits Dr. O Moonian

Labsheet 1

The first labsheet will help you get acquainted with the C programming language and the Linux environment. You will essentially need to get yourself acquainted with the following in C programming language:

- Performing Inputs and Outputs.
- Using conditional statements.
- Using loops

Input/Output in C are different from C++. However, conditional statements and loops are essentially the same.
In case the one lab session is not sufficient, you are advised to dedicate some of your own free time to get acquainted with the essential programming constructs in the C programming language.

1. Login onto Linux.
2. Create a directory for your lab works.
3. Open a text editor. Write the required C-codes that allow you to display the message “Hello World” on the screen. Save the file as OS12016.c
4. Compile and run the program (This will be explained in the lab)

Henceforth for all programs, in CSE 2022Y, it is understood that the programs will be written in C, compiled and run.

5. Write a program to input the radius of a sphere and output the volume and surface area of the sphere.
6. A student taking a module at the University is assessed both on the exam paper and continuous assessment. Continuous assessment is on 50 marks while the exam is on 100 marks. To obtain a pass in a Module a student must score more than 40% in each component. If a student scores more than 80% in one component but has scored between 35% and 40% (inclusive) in the other component he/she is “Pushed Up”. Write a program to input a student’s marks in each component and enter whether he/she has passed, failed or been “pushed up”.
7. Write a program to input a number of positive values terminated by -1 and display the number of even values input as well as the sum of the even values.
8. Write a program that allows the input of an integer **n**, validates that **n** is

positive and displays whether **n** is prime.

9. Write a program that allows the input of an integer value **n**, followed by **n** more integer values. It displays the largest value input among the n integer values.
10. Write a program that allows the input on an integer value **n**, representing the number of patients that visit a doctor over a given time period. The program should then allow the input of **n** sets of **three** values, representing the patient id, weight and fast blood sugar (**FBS**) level of the **n** patients. It displays the patient id, FBS and weight of the patient having the highest FBS level and also for the patient having the highest weight. If the two are the same patient, it should display the information only once.

Labsheet 2

In this labsheet, you will work with functions and arrays in C. For all programs requiring you to write functions, your program should also contain required codes to test the functions.
Note: Reference Parameters in C is through the use of pointers.

Note: All Programs must be written in C.

Question 1

A function f(x) is defined as $f(x)=x^4 +x^3 + 3x^2 + 2$. Write a function **poly**() that takes an integer, x, as parameter and it implements the function f(x), i.e for any given value x, poly(x) should return the value of $x^4 +x^3 + 3x^2 + 2$. You can make use of **pow()** function from the math library.

Note: To use the math library, you have to use #include <math.h> in your programs and for compilation you have to use

gcc <program name> -lm –o <object file>

Eg. **gcc prog1.c -lm –o prog1**

Question 2

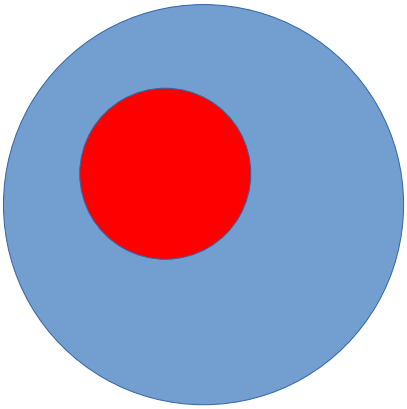
Write a function that has two integer parameters **x** and **y** and swaps the values of the two parameters. Make use of reference parameters, so that the function can be used by other parts of the program to swap values.

Question 3

Write a function that has as parameter the radius of a sphere and it calculates the surface area and volume of the sphere. These values should be passed to the caller.

A toy is made of two balls, one fitted inside the other one as shown in the diagram below. The space in-between the balls is filled with a liquid. Write a main function that allows the input of the radius of each of the balls and it calculates and displays

the volume of the liquid as well as the total surface area that is in contact with the liquid. Make use of the function defined above.



Question 4

Write a function that has as parameters an array A of integers and an integer value n representing the number of elements in the array. The function finds the maximum and minimum values in the array and passes these to the caller.
Write a program (main() function) that allows a user to input a number of integers in an array and then uses the above function to find the smallest as well as the largest value in the array. These values should be displayed.

Question 5

Write a function called **inputArray()** that has as parameters an integer array, **A,** and an integer, **n**. It allows the input of **n** values into array **A**. Write a function **displayArray()** that also takes as parameters an integer Array, **A,** and an integer, **n,** and performs the display of **n** values from **A**. Write the required program code to test your functions.

Strings in C are arrays of characters.

Question 7

Write a program that performs the input of the surname, first name and address of a person and displays back these values.

Question 8

You have to input the surname, first name of **n** students (n<=20) and their marks in OS. Write a program that contains the following:

- A function to perform the input of the surname, first name and marks of a student.
- A function that has as parameters an array of **floats** and an integer **count**,

and it finds the positions of the maximum and minimum values in the array and passes these to the caller. The integer **count** represents the number of elements in the array.

- A main() function that allows the input of an integer value **n**. It calls the above functions to:
  - perform the inputs of surname, first name and marks of **n** students.
  - Find and display the surname, first name and marks of the student having scored the highest marks and of the student having scored lowest marks.

Labsheet 3 –

In this labsheet, you will work with structs in C. These are exercises that you have worked out in C++. However, the passing of reference parameters in C is different.
For any function that needs to perform input on a struct or modify an attribute in a struct, use reference parameters.
Note: I am allowing 2 weeks for this labsheet. This will allow you to also catch up on any possible backlog.

1. Write a program that defines a struct **student** to hold the following information for a student: surname, other names, address, age. The program should have the following functions:
   - A function **inputStudent()** that takes a student struct as parameter and allows of input of data that are stored in the struct parameter.
   - A function **displayStudent()** that takes a student struct as parameter and displays the attribute values on the screen.

The program should declare two variables of type **student**, allow input of data for two students and store them in those variables. The program should then display the names and address of the older student among the two. If they are of the same age, the program should display the names and addresses of both of them.

2. Modify the program in question 2, so that the **main()** function declares an array of size 20 of the struct student, allows the input of an integer value n, and performs the input of data for n students, then displays them back. The program should then display the name, address and age of the youngest student in the array. If there are several students with the youngest age, the program will display only the first one.

3. Modify the program in question 2 so that it uses an array of pointers to struct. Note: In C, we use **malloc()** to allocate memory and **free()** to release the memory.

4. A garage needs to store information about servicing it provides to vehicles. For each servicing it performs in a day, it wishes to store the servicing number, the registration number of the vehicle, and the surname of the owner and the cost of the servicing. Write a program containing the following:
   - A struct **servicing** containing the following data members,
     ```
     int servicing_no;
     char Reg_no[10];
     char surname[15];
     float cost;
     ```
   - A function **Input_serv()** to input the data of one servicing record, but not the cost.
   - A function **Disp_serv()** to display the data of one servicing record.
   - A function **assign_cost()** that assigns the cost of a servicing, given the servicing number. The function should also take as parameter an array containing servicing records.
   - A main program that declares an array for storing information for up to 20 servicing records (structs). The program then displays a menu that allows users to choose one of the following:
     - Add a new servicing record.
     - To enter a servicing number and the program allows the input of the cost and assigns the cost to the servicing record.
     - To enter a servicing number and the program displays all information for the corresponding record
     - The display of the average costs for records in the array

     The program should then perform the task required by the user. Also an option should be provided for exit the program.

Labsheet 4

This labsheet introduces you to the use of Linux ***System calls***. A system call is a function that resides within the Operating System kernel, that is used either to obtain information about data managed by the kernel or to request the OS to do some work. In the first part of this first labsheet on *system calls*, you will get access to some attributes of the process and also to obtain information about time. In the second part, you will use system calls for performing files I/O.

The prototype for each system call is usually defined in a header file. To use a system call, you need to include the appropriate header file in your program and you also need to know what the function header is like so that you can call the function, i.e you need to know the return type as well as the type and purpose of each parameter. These information are available in the built-in manual **man**. To see the online manual for a *system call* **funct1**, for example, You simply type ***man funct1.*** Since with Ubuntu Linux, man does not automatically include all system calls and commands available, you can use the following address to obtain these information: http://linux.about.com/od/commands/l/blcmdl_2a.htm.

Part 1 – Introduction to System calls

Question 1

The *system calls* **getpid()** and **gettppid()** respectively return the identifier of a process and the identifier of its parent. The synopsis is as given below. The type pid_t can be treated as integers. Write a program to display the pid value of a process as well as that of its parent. Use the **ps –a** command to check if the **pid** displayed by the process is the same as that displayed by

the **ps –a** command. You can use the **sleep()** system call to cause the process to sleep for some time to allow you to use the **ps -a** command.

Note: Use two terminals: One to run your program and another to type in the command **ps –a**.

SYNOPSIS

```
#include <sys/types.h>
#include <unistd.h>

pid_t getpid(void);
pid_t getppid(void);
```

Question 2

Study the system call **gettimeofday()**. Write a program that contains the following:

- A function **Power**, defined by you to calculate X^n, where X is of type float and n is an integer. You can use iteration or recursion, but **do not** make use of the pre-defined function **pow()**.
- A **main()** function that calculates the value of a polynomial, such as $X^{100} + X^{99} + X^{98}$, for some chosen value X.
- Use **gettimeofday()** to find out for how long the program runs. (Time should be displayed in number of seconds and milliseconds).

Question 3

Write a program that executes a loop causing it to sleep for 2 seconds (using the sleep() system call), wakes up and displays its pid as well as the total number of milliseconds elapsed since it started. It can run the loop any number of times you choose. Your output can be in the form:

Program with pid <pid> has run for <number of milliseconds> milliseconds.

Note: Make use of sleep() and gettimeofday() system calls.

Part 2 – File I/O System calls

In this part of the labsheet you will work with file I/O system calls. Make use of the following system calls.

- open
- close
- read
- write
- lseek

For the include statements, use the following:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
```

To use a file you first need to open it through the **open()** system call. The prototypes for opening a file is:

```
int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);
```

Note: The open call requires: The pathname of the file, the flags which indicate whether the file is opened for reading, writing, read & write etc., and the mode, which specifies the different types of access rights to the file.
To open a file for reading, use the first prototype. To open a file for writing, use the second one. Note that open() returns an integer. The returned value is called a file descriptor and is used for accessing the file.
In this labsheet, you have been requested to use open(), read(), write()...etc. **Do not use** fopen(), fread(), fwrite().
The **read()** and **write()** system calls use the file descriptor returned by open(). They both use pointers to void types, which allow any type of data (including structs) to be cast.

The include file for read and write are: <unistd.h>

```
#include <unistd.h>

ssize_t read(int fd, void *buf, size_t count);
```

ssize_t write(int *fd*, const void **buf*, size_t *count*);

You can take **ssize_t** to be **int**.

An example for opening the file file1.dat for reading 20 bytes is:

```
void main(){
int fd;
char buf[25];

fd=open("file1.dat",O_RDONLY);
n=read(fd,buf,20);
// code for using the data should be inserted here.....
.....
.....
close(fd);
}
```

When opening a file for writing, you want the file to be created if it doesn't exist. For that you should use the flag O_WRONLY, ored with O_CREAT as follows:

```
fd=open("file1.dat",O_WRONLY|O_CREAT);
```

You should also give the access rights, through the mode:
Eg.: fd=open("file1.dat",O_WRONLY|O_CREAT, S_IRWXU);

An example

```
void main(){
int fd;
```

```
char buf[]=”Hello”;

fd=open(“file1.dat”,O_WRONLY|O_CREAT, S_IRWXU);
n=write(fd,buf,10);
.......
......
close(fd);
}
```

You are required to explore the uses of the system calls, to find out how to specify the size of data to use in read() and write() calls and to find out the values of n in each case. You will also need to find out how to read a file until we reach the end of the file.

Question 4
Using a text editor, create a file called “lab44.dat”. Write a few sentences in it. Write a program to read the text from the file and display on the screen.

Question 5
Write a program that opens a file called “lab45.dat” for writing. If the file does not exist, the program creates it. If it exists already, the program will simply overwrite it. Your program should then read data for 5 students from the terminal and write them into the file. Your data should consist of the names of students (20 characters), a student id (10 characters), date of birth (10 characters), gender (7 characters (male/female)) and marital status (10 characters(single/married)). Write the data in the file as individual fields.

Question 6
Write a program to open the file in question 5, read all the data and display them on the screen.

Question 7
Write a program to open the file in question 5 in Append mode and add information for 5 more students. Use the program in question 3 to read the information from the file and display on the screen.

Question 8
Modify the programs in questions 5 and 6, so that they use structs for storing students information. Each write or read should write/read one struct. Note: you have to make use of sizeof() to obtain the required number of bytes to write/read.

Question 9
Write a program that opens the above file, allows you to enter an integer value, (eg. 5) on the terminal, go to that particular student in the file and displays the information for that student. (Hint: use lseek).

Question 10
Write a program that creates a file called “input13.dat”. It then allows you to enter the names and address of a number of persons. For each person it allows you to

input the name and address, it computes the length of the names and address and writes in the file “input13.dat”, the length of the name, the name, the length of the address and the address.

Question 11
Write a program that reads the file “input13.dat” (created in question 10), displays the name, the size of the name, the address and the size of the address for the person with the longest name as well as for the person with the longest address. The program should also display the average size of names and average size of addresses in the file.

Labsheet 5

This labsheet introduces you to the concept of threads and the use of mutex locks. The URL http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html provides a reasonably good reference to Pthread.
Note: In the example given example, the main thread executes a loop alternating between sleeping and displaying a message. This ensures that the process continues to be alive while the child thread executes. If main exits, the process will be killed. To avoid this, use the call pthread_exit() instead of return in main().
The system call pthread_create allows the creation of threads in Linux systems. The following program creates a thread, the parent executes a loop 1000 times, then displays a message, sleeps for 2 seconds the whole is repeated 10 times. The created thread is similar to the parent except that it

sleeps for 1 second instead of 2.

```
#include <pthread.h>

void * funct1(void * arg);

int main()
{
  pthread_t threadid;
  int i,j;
  int x=1;

 pthread_create(&threadid,NULL,funct1,(void *)&x);
  for (j=0; j <10; j++)
  { for (i=0; i<1000; i++);
     printf("Hi I'm the parent\n");
       sleep(2);
   }

return 0;
}
 void * funct1(void * arg)
  { int i,j;

  for (j=0;j<10; j++)
    { for (i=0; i<1000; i++);
     printf("Hi I'm the created thread\n");
       sleep(1);
   }
        }
```

1. Type in the program, compile and execute it.

   The command for compilation is:

   gcc <programname.c> -lpthread –o <objectfilename>

Note: The purpose of the **sleep()** call in the main thread is to ensure that the process remains alive for 2 seconds, after execution of the main (parent thread), to allow the created thread (child thread) enough time to complete its execution. A better option is to use **pthread_exit().**

2. Write a program that creates an array of 10 student names and an array of 10 student ids. You can choose to input the values or directly assign them. The program should then create two threads. The first created thread executes a loop that displays all the names, while the second thread should execute a loop displaying the ids. In the mean time the

parent thread should sleep for 5 seconds, then display “Parent Thread Exiting” and then exits.

3. The formulae for calculating sin(x) and cos(x),by Mc Claurin’s series, where x is in radians, is given below.

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots \text{ for all } x$$

$$\cos x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \cdots \text{ for all } x$$

(i) Write suitable C-functions for calculating sin(x) and cos(x) using these formulae.

(ii) Write a **main** function that allows the input of an angle x in degrees, converts it to radians, creates two threads. One thread calculates and displays sin(x), while the other one calculates and displays cos(x).

4. Values can be passed from a parent thread to the created thread. The value is cast to (void *) in the function pthread_create(). The value can then be obtained from the function argument in the called function, by casting back the argument. Modify question 2 so that the parent thread can pass to the created thread a number (from 1 to 10) and the child thread should display that number of values. The number passed to each created thread can be a different one.
   Note: In this case the number should be cast as void when being passed to pthread_create and cast back to integer within the thread.

5. Mutex locks provides mutual exclusion over variables shared between threads. Modify the initial program so that it declares a global variable, **I**, initialized as 0. Both the parent and child thread should increment **I** each time they go into the loop. **I** should be protected by the use of mutex locks.

Note: To use a mutex lock, it has to be declared of type **pthread_mutex_t**.
Eg. pthread_mutex_t mylock;
The mutex lock has to be initialized. This can be achieved at declaration time using the macro PTHREAD_MUTEX_INITIALIZER.

Eg. pthread_mutex_t mylock = PTHREAD_MUTEX_INITIALIZER;

The commands **pthread_mutex_lock** and **pthread_mutex_unlock** perform the locking and unlocking of mutex lock types.
Eg. **pthread_mutex_lock(&mylock);**
pthread_mutex_unlock(&mylock);

6. Modify the program in question 3 so that it works as follows:
   - The main program declares a 2-D array with 3 columns. In the first column, it fills in a number of angles of your choice between 0 and 2π radians (you can choose the intervals eg. 0, π/6, π/3, etc. It then opens a file "angles.dat" for writing and finally creates two threads.
   - The threads calculate respectively the values of **cos** and **sin** for different values of x and store them in the array, with **cos** in the 1st column and **sin** in the second column. (Note: thread 1 calculates **cos** and thread 2 calculates **sin**)
   - Thread 1 reads the different angles and write the angles and their **cos** values in the file **"angles.dat"**.
   - Thread 2 reads the different angles and write the angles and their **sin** values in the file.
   - The thread that completes the work last closes the file (You can use a shared variable to indicate if one thread has already completed the work).

   (Note: Either thread should write the values for all the angles before the other one start writing the values. – Use mutex lock for the purpose).

Labsheet 6

The function call **fork()** is different from other system calls in that it creates

a new process (which is referred to as the **child process-** The initial process is called the **parent process**). The child process is an exact copy of the parent, with the only difference that the value returned by **fork()** in the child is 0, while in the parent, the value returned is a non-zero value (the pid of the child process).

1. Write a program that forks to create a child process. Each of the parent and the child should execute a loop causing it to sleep for 1 second, wake up, display its pid and say whether it is the parent or the child and also display its parent's pid.

   Also use the command **ps -a,** in a separate console window, to check the pids of the parent and child processes.

2. Write a program that reads a number of integer values from the terminal and places them in an array. The program should then fork. The parent process should display the values in all the odd numbered elements while the child displays values in the even numbered elements. For each display the process displaying should indicate whether it is the parent or the child.
3. Modify the program in question 2 such that after the fork, the child modifies values in all odd numbered elements. Each process should now display all its elements.

Labsheet 7 – One Week

The system call signal() allows programmers to register signal handlers. A **signal handler** is a function that executes when the specified signal is received by the process. Whenever the signal occurs, the program will leave its normal execution and execute the signal handler. Eg. The alarm() call causes a SIGALRM signal to be generated in a given no. of seconds.signal(SIGALRM, funct1) will cause the function funct1 to be executed when the SIGALRM signal is triggered.

The pause() system call causes the program to suspend, waiting for an event, such as a signal. In the program below, the **main()** function registers the signal handler function **handler()** to be called when an alarm is triggered. It then sets an alarm to be triggered in 10 seconds.

```
#include <stdio.h>
#include <unistd.h>
#include<signal.h>
void handler(){
   printf("signal handler called\n");
}

int main(){
    signal(SIGALRM,handler);
    alarm(10);
    pause();
}
```

Question 1

Type in the given program, compile and execute it. Then modify the

program so that triggers it an alarm in 5 seconds. When the alarm occurs, it displays the message “Alarm triggered” and exits. Change the name of the signal handler function to **funct1()**.

Question 2

Write a program that repeats the above for 200 seconds and additionally, each time the alarm is triggered, it displays the no. of seconds and microseconds elapsed since the program started execution. (Hint: Simply place the alarm system call in a loop and use gettimeofday()).

Question 3

The setitimer() call allows the programmer to reset a timer at a regular time interval. It also provides for finer time intervals than the alarm, since it can set the timer at milliseconds. Check its use and write a program that opens a file, triggers a timer every 500 ms, write in the file the total time elapsed since it started executing, every time the timer is triggered. The program should run for a total of 30 seconds.

Question 4

SIGALRM is one kind of signal and it is triggered when an alarm (or timer) fires. There are other kinds of signals that are triggered in other situations as well. For example, a SIGFPE signal is triggered when there is an error in an arithmetic operation such as a division by zero. Usually a program causing a division by zero aborts. By registering a signal handler for SIGFPE, you can display a suitable message for the user. Write a program that allows the input of two integer values, x and y. It calculates and displays the value of y/x. However if x happens to be zero, it uses the signal handler to display a message for the user. **Note: The handler will keep executing indefinitely. Use exit() to exit from the handler. Unfortunately the SIGFPE signal doesn’t provide ways to return to the caller to do something useful.**

The signals SIGUSR1 and SIGUSR2 are two user-defined signals, i.e they are signal values that can be used in programs but the system attaches no special meanings to them. The kill system call allows a process to send a signal to another process (Do not worry about the name **kill**. This term was

used since most signals cause the receiving process to terminate.). The syntax of the **kill** system call is **kill(pid_t pid, int sig)**. For a process P1 to send a signal to another process, P2, P1 must know the pid of P2. This can only happen if P1 is the parent of P2 (the pid of P2 is returned to P1 after a fork()) or if P1 is a child of P2 (use of getppid()) if P1 and P2 have a common ancestor and they can communicate their pid's through the common ancestor.
The program below creates a child process and performs an exchange of signals between the parent and the child. Note the sleep(1) in the parent. This is to give the child the time to execute and register the signal handler before the parent sends it the signal. Otherwise the signal will be lost.

```
#include <stdio.h>
#include <unistd.h>
#include<signal.h>
#include <stdlib.h>

void fromParent(int x){
  printf("Hi!-Received signal from Parent\n");
}

void fromChild(){
  printf("Hi!-Received signal from Child\n");
}
int main(){
   int p;

   p=fork();
   if (p==0){
     signal(SIGUSR1,fromParent);
     pause();
     kill(getppid(),SIGUSR2);
     }
   else{
      sleep(1);
      kill(p,SIGUSR1);
      signal(SIGUSR2,fromChild);
      pause();
       }
   return 0;
```

}

Question 5

Write a program that forks to create two children. The parent then sleeps and wakes up every 10 seconds. Each time it wakes up it sends a signal SIGUSR1 to the first child and SIGUSR2 to the second child. After two minutes it interchanges the signal sent and sends SIGUSR2 to the first child and SIGUSR1 to the second child and continues for a further 2 minutes. The children, each, keeps two counters C1 and C2. C1 is initially 0 while C2 is initially 1. Whenever a child receives a signal SIGUSR1, it increments C1 by 1 and when it receives SIGUSR2 it multiplies C2 by 2. In either case it displays its pid and both C1 and C2.

Labsheet 8 – One week

Using Pipes

Pipes are means of communication between a parent and a child. The **pipe**() system call takes as parameter an integer array of size 2, and returns two file descriptors. The descriptor in element 0 of the array is used for reading from the pipe while the one in element 1 is used for writing.
Eg. if we have:

```
int filedes[2];
pipe(filedes);
```

A process can read from filedes[0], while the other one can write into filedes[1]. In the program below, the parent writes “hello word” to the pipe, while the child reads it and displays on the screen.

Note: After the fork, each of the parent and the child will have a copy of both fd[0] and fd[1]. Each one is closing the one it doesn't need.

```
#include <stdio.h>
#include <stdlib.h>
```

```
void main()
{
 int fd[2];
 int p;
 int n;
 int x;
char line[20];

 x=pipe(fd);
 p=fork();

  if (p==0)
    { printf("Hello- I'm the child\n");
      close(fd[1]);
      n=read(fd[0],line,12);
      printf("Child - Line read was %s \n", line); }
  else
     { close(fd[0]);
       write(fd[1],"hello world \n",12);
       }

   exit(0);
}
```

Question 1
Type in the given programme and run.

Question 2
Using a text editor, create a file called "**myfile.txt**", containing some text (say one or two paragraphs). Create a program that opens the file "**myfile.txt**" for reading, creates a pipe, then forks. The parent process reads data from the file in blocks of 20 characters, sends it to the child through the pipe. The child reads the data from the pipe and displays it on the screen.

Question 3
In C, programs can take arguments. For that the main can have two parameters, called **argc** and **argv** and is declared as follows:

```
int main(int argc, char *argv[]){
.....  }
```

When the program is run with arguments, **argc** will keep count of number of arguments given (including the program name), while **argv** will contain the arguments, as follows**: argv[0]** will contain the program name, **argv[1]** will contain the first argument, **argv[2]**, the second argument etc.

Eg. If my programme is called **prog1** and I run it as follows:

prog1 file1 file2

argc will have value 3, argv[0] will have value **prog1**, **argv[1]** will have value **file1** and **argv[2]** will have value **file2**.

Modify the program in question 2, so that the program runs with two arguments: an input file and an output file. The parent process reads from the input file and writes into the pipe, while child reads from the pipe and writes to the output file.

Question 4

Modify the program in question 3 so that when the parent reaches the end of the file, it sends a signal to the child, wait for 1 second and exits. On receiving the signal, the child also exits.