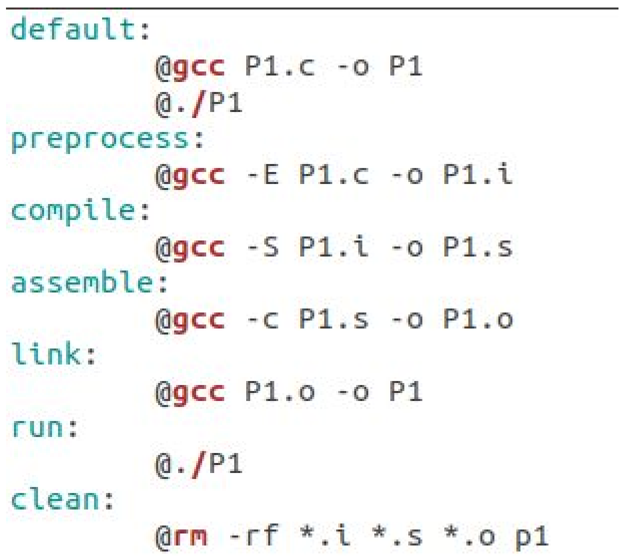
OS Assignment 1 (Ques1)

Gitansh Raj Satija 2019241 CSE

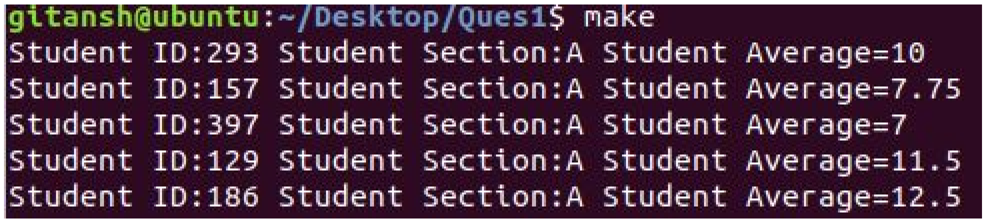
Documentation:

Makefile:

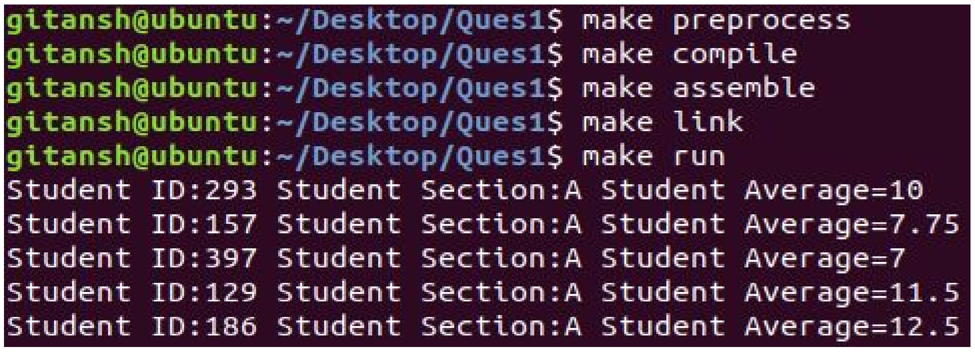


```
default:
	@gcc P1.c -o P1
	@./P1
preprocess:
	@gcc -E P1.c -o P1.i
compile:
	@gcc -S P1.i -o P1.s
assemble:
	@gcc -c P1.s -o P1.o
link:
	@gcc P1.o -o P1
run:
	@./P1
clean:
	@rm -rf *.i *.s *.o p1
```

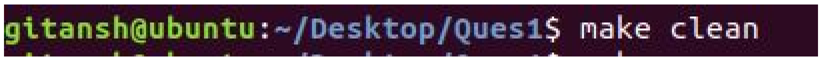
This is the makefile which pauses the compilation at every phase. Here are the outputs using the makefile.



```
gitansh@ubuntu:~/Desktop/Ques1$ make
Student ID:293 Student Section:A Student Average=10
Student ID:157 Student Section:A Student Average=7.75
Student ID:397 Student Section:A Student Average=7
Student ID:129 Student Section:A Student Average=11.5
Student ID:186 Student Section:A Student Average=12.5
```



```
gitansh@ubuntu:~/Desktop/Ques1$ make preprocess
gitansh@ubuntu:~/Desktop/Ques1$ make compile
gitansh@ubuntu:~/Desktop/Ques1$ make assemble
gitansh@ubuntu:~/Desktop/Ques1$ make link
gitansh@ubuntu:~/Desktop/Ques1$ make run
Student ID:293 Student Section:A Student Average=10
Student ID:157 Student Section:A Student Average=7.75
Student ID:397 Student Section:A Student Average=7
Student ID:129 Student Section:A Student Average=11.5
Student ID:186 Student Section:A Student Average=12.5
```



```
gitansh@ubuntu:~/Desktop/Ques1$ make clean
```

Note: Since the actual output contains 400 lines, only a small fraction of the final output is displayed here.

<u>Code Explanation:</u>
The following header files were required to make use of the system calls, operate on the data and do error handling.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include<sys/wait.h>
#include <sysexits.h>
#include <errno.h>
```

We open the csv file that has to be operated using the open() system call, which returns a non-negative integer representing the lowest numbered unused file descriptor upon successful completion and if due to some reasons the file was not opened then it returns -1 and also sets errno corresponding to the error due to which file wasn't opened. We use the perror() function to display the error message.

Then we extract the data to a string using the read() system call (which will return -1 if the call was unsuccessful). The data we have now is not in the form where we can operate it, hence we tokenize the data using strtok() according to our needs. In this code, I first ignore the entries of row1 as they contain the headings on the columns and not data. Next, I separate each row entry and store them in a 2D character array.
Next, to store data in an efficient manner, we create a structure with the following description:

```
struct Student{
        int id;
        char section;
        int m1,m2,m3,m4;
        float avg;
}
```

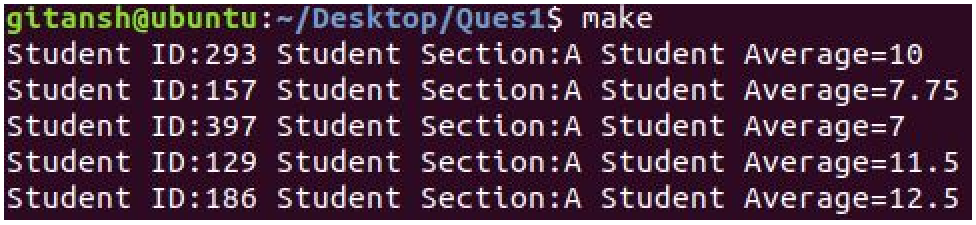
To extract each element from the row entry we use strtok() and separate the values using ‘,’ as delimiter and store it in the struct elements after appropriate typecasting. To store all the 400 entries, we create an array of type struct.

We have a function that finds out the average of every student and prints a formatted output simultaneously using the write() system call.
void calculate_avg(char ch)
This function accepts a character type argument which helps it identify whether it has to operate on students of section ‘A’ or ‘B’.

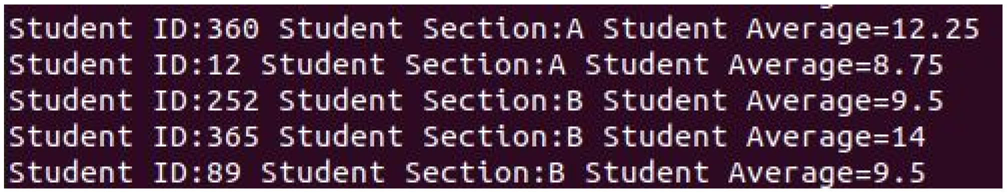
Since we are asked to create a child process, I have implemented a fork() system call which creates a new child process that is an exact copy of the parent process except that it has a unique processing ID which is different from its parent process. Fork() returns 0 in the child process and a non negative integer in the parent process. We use this to identify which out of the child/parent process is being executed. If fork() returns 0 then it mansion is being executed and in this case we pass’ A’ to our function as the child has to operate on student of section A and if the return value is more than 0 then we pass ‘B’ to the function as the parent process has to operate on students of Section B. Fork() returns -1 in case of any errors.

The system usually runs the parent process first and then the child process. But here we want the parent process to operate once the child process is terminated. So for this we use the waitpid() system call which

is used to get the process ID of the terminated child process. This is used to stall the main process so that the child process can run and terminate itself first. exit() helps the child process to terminate.
As we can see in the output as well, the students of section 'A' are printed first as the child process is executed first.



```
gitansh@ubuntu:~/Desktop/Ques1$ make
Student ID:293 Student Section:A Student Average=10
Student ID:157 Student Section:A Student Average=7.75
Student ID:397 Student Section:A Student Average=7
Student ID:129 Student Section:A Student Average=11.5
Student ID:186 Student Section:A Student Average=12.5
```



```
Student ID:360 Student Section:A Student Average=12.25
Student ID:12 Student Section:A Student Average=8.75
Student ID:252 Student Section:B Student Average=9.5
Student ID:365 Student Section:B Student Average=14
Student ID:89 Student Section:B Student Average=9.5
```

Note: Since the actual output contains 400 lines, only a small fraction of the final output is displayed here.

Since the code uses System calls we must do error handling for them in case a system call fails due to some reason. The error message is shown with the help of perror() function.
We do error handling against the open(), read(), fork(), waitpid(),write() system calls. Some errors that the code defends:-

- Search permission is denied on a component of the path prefix, or the file exists and the permissions specified by *oflag* are denied
- A loop exists in symbolic links encountered during resolution of the *path* argument.
- The *fildes* argument is not a valid file descriptor open for reading.
- The read operation was terminated due to the receipt of a signal, and no data was transferred.
- Insufficient storage space is available.
- The system lacked the necessary resources to create another process, or the system-imposed limit on the total number of processes under execution system-wide or by a single user {CHILD_MAX} would be exceeded.
- The calling process has no existing unwaited-for child processes.
- The function was interrupted by a signal. The value of the location pointed to by *stat_loc* is undefined.
- The *fildes* argument is not a valid file descriptor open for reading.
- The read operation was terminated due to the receipt of a signal, and no data was transferred.