National University of Computer and Emerging Sciences



## Laboratory Manual

*for*

## Operating Systems Lab (CL-220)

|  |  |
| --- | --- |
| Course Instructor | Mr. Namra Absar |
| Lab Instructor (s) | Saba Tariq |
| Section | BCS-4A |
| Semester | Spring 2024 |

Department of Computer Science FAST-NU, Lahore, Pakistan

Objectives:

* MakeFile
* Pipes

# MakeFile

* Make is a Unix utility that is designed to start the execution of a makefile. A makefile is a special file, containing shell commands you create and name makefile (or Makefile depending upon the system). While in the directory containing this makefile, you will type *make* and the commands in the makefile will be executed. If you create multiple makefile, be certain you are in the correct directory before typing make.
* Make keeps track of the last time files (normally object files) were updated and only update those files that are required (ones containing changes) to keep the source file up to date. If you have a large program with many sources and/or header files, when you change a file on which others depend, you must recompile all the dependent files. Without a makefile, this is an extremely time-consuming task.
* As a makefile is a list of shell commands, it must be written for the shell to process the makefile. A makefile that works well in one shell may not execute properly in another shell.
* However, for a large project where we have thousands of source code files, it becomes difficult to maintain the binary builds. The **make** command allows you to manage large programs or groups of programs.
* The **make** program is an intelligent utility and works based on the changes you make in your source files. If you have four files main.cpp, hello.cpp, factorial.cpp and functions.h, then all the remaining files are dependent on functions.h, and main.cpp is dependent on both hello.cpp and factorial.cpp. Hence if you make any changes in functions.h, then the **make** recompiles all the source files to generate new object files. However, if you make any change in main.cpp, as this is not dependent on any other file, then only main.cpp file is recompiled, and help.cpp and factorial.cpp are not.
* While compiling a file, the **make** checks its object file and compares the time stamps. If the source file has a newer time stamp than the object file, then it generates a new object file assuming that the source file has been changed.

# Structure of Makefile:

Target: dependencies Action

# Naming of Makefile:

By default, when making looks for the makefile, it tries the following names, in order: `GNUmakefile', `makefile', and `Makefile'. You can give any of the three names to your makefile. The convention is to use the name “Makefile” (capital M).

# Running the Makefile:

Simply run the command “make”. The current working directory should be where the intended makefile is placed.

# Benefits of Makefile:

Makefile checks the last modified time of both the source file and the output file. If the output file’s last modified time is later, then it will not compile the source files since the outputfile is already the latest. However, if any of the source files are modified after the creation of the output file, then it will run the command since the output file is outdated.

# Example:

Suppose we have two cpp files: main.cpp, lib.cpp, and a header file lib.h. Suppose the main function in main.cpp makes use of several functions from lib.cpp. To compile our program, we will create the makefile as follows:

main.out: main.cpp lib.cpp

g++ main.cpp lib.cpp -o main.out

You can get help from this link for makefile https://[www.youtube.com/watch?v=\_r7i5X0rXJk](http://www.youtube.com/watch?v=_r7i5X0rXJk)

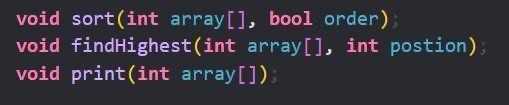
**In-Lab Tasks**

## Question 1: (MakeFile)

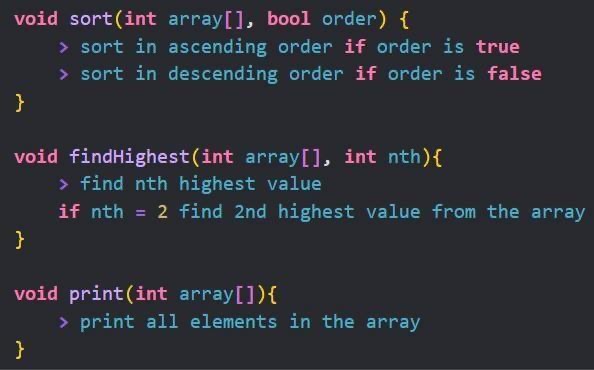
Create 3 files

* + main.c or .cpp
  + functions.c or .cpp
  + header.h

header.h file contains the following function prototypes



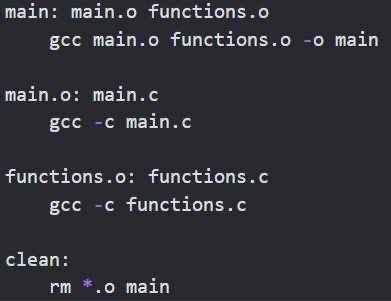
functions.c file contains the following 3 functions along with their logic



In main.c you will accept command line arguments including 3 things

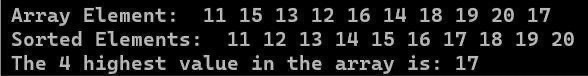
* + an array of integers
  + order of sort (1 for ascending order and 0 for descending order)
  + nth position to get the nth highest number from the array

Use makefile to execute all these files. Your makefile will look like this.



**Example:**

Input: ./main 11 15 13 12 16 14 18 19 20 17 Output:



**Question 2: (Pipes)**

Imagine you part of a special team working on a cool project! Here's the deal: You've got a teacher, like the leader of the team, and a student, like the helper. The teacher writes a document, like a message for the team, and asks the student to check it for any weird stuff, like odd symbols or numbers. Once the student's done, they give the document back to the teacher for final fixes. It's teamwork at its best! Now, let's write a program that does just that, but with code instead of paper and pen!

Write a C/C++ program named **document\_proofreading.c/cpp** that implements the collaborative document editing tool described above. Your program should utilize pipes and **fork()** for communication between the teacher and student processes. The teacher process should be responsible for uploading the document, while the student process should proofread it and send the cleaned version back to the teacher.

Ensure proper error handling for system calls and close unused ends of the pipe in both processes.

## Scenario Details:

* The teacher process represents the teacher preparing a document for proofreading.
* The student process represents the student proofreading the document and removing any errors such as special characters or numbers.
* The communication between the teacher and student processes should allow for effective collaboration and document editing.