CS108 - Software Systems Lab

Lab 5 - `gdb` and `Makefiles`

Student: Aditya Sanapala, 23b0912@iitb.ac.in
Lecturer: Kameswari Chebrolu, chebrolu@cse.iitb.ac.in

Problem 1: Let's make a `Makefile`!

In the current working directory (`/home/labDirectory`), you will find a `src/` directory with two subdirectories, `src1/` and `src2/`. In each of these we have several `.cpp` files along with their corresponding header files. For each of these C++ files, compile it to a corresponding object file with the same name and a `.o` extension. (Use the flags `-Wall` and `--std=c++17` for these compilations). All these object files should be placed in the current working directory (`/home/labDirectory`).

Apart from the `src/` directory, we have a `main/` directory which contains several C++ files with each of them having a main function. For this case we have files `main1.cpp` to `main9.cpp`. We want to create executables by linking each of them with all the created object files in the current working directory. The name of the executable should be the same as the name of the C++ file. (Use the same flags as earlier). All of these executables should be created in the current working directory.

Finally after running the make command, the current working directory should contain all the object files corresponding to every C++ file in the subdirectories of `src/` and all the executables corresponding to every C++ file in the `main/` directory. No other new file should be present at any other location.

Note: No two of the `src` C++ files or the main C++ files have same names, so you may not worry about name conflicts.

Additionally, create a `clean` target to remove all the object files and executables in the current working directory, and regaining the directory to its initial state.

Marking scheme for this question is as follows:

1. 20 Marks for achieving everything mentioned in the question. (i.e., correctly working `make` and `make clean` and not violating the very purpose of a makefile (Refer Reading 1 below).
2. 10 Marks for achieving the same in less number of target rules than the number of files to be compiled.

Hint: Patterns can be of some help?

Reading 1:

One can easily write a makefile with just 2 targets : `all` and `clean`, and satisfy the creation and removal of required files. Consider the simple case, where we have just three main files, namely `main1.cpp`, `main2.cpp` and `main3.cpp` and no other object files. We wish to write a makefile to manage this small project. Refer to the following two different Makefile codes in `Makefile1.png` and `Makefile2.png`.

What's the difference? When we run `make` for both of these cases, you would notice that when a change is made to `main2.cpp`, re-run of `make` recompiles and creates all executables in the first case, whereas in the second case, only the `main2` executable is recreated. Which one of this do you think is correct?

Reading 2:

You may notice that not all the C++ files in the main directory includes all the header files of the `src/` directory. For example, `main3.cpp` doesn't include "rem.h", which means ideally, any changes to rem.h shouldn't affect the executable created from `main3.cpp` i.e., `main3`. But for simplicity of writing code, let's assume that any change in any header file of `src/` will lead to recompilation of every main file. You see this is a potential improvement that can be made in `make`, There are other tools which are built on top of make and provide this functionality. But for now, let's not get into its details. This is not a part of syllabus. But for the curious ones, refer to this link.

Bonus: (NOT PART OF SYLLABUS)

Can you write a `MakeFile` which can handle cases where there may be multiple subdirectories inside `src/` and each of these subdirectories contains multiple `.cpp` files. There may also be several `.cpp` files in the `main/` directory?

Curious ones can read about wild card functions $*$ in make.

Problem 2: The Quest for the Binary Scroll: Episode 1

In the hallowed halls of IIT Bombay, Guramrit, an ambitious computer science student, discovers himself in a mysterious room in IIT Bombay, filled with ancient scrolls and a strange aura. Excited but puzzled, Guramrit sought the help of his brilliant friend, Sabyasachi. On a parchment, they discover a message from a long-lost professor, challenging them to find the correct scroll which holds the clue to the hidden treasure. Suddenly, the door of the room begins to close with the duo being unable to stop it, worried, they decide to implement the fast binary search algorithm to find the correct scroll. They seek your help to debug the code and find the correct scroll as fast as possible. Can you help them?

In this activity, we have implemented the binary search algorithm to find the position of a given number in a sorted array. The binary search algorithm is a fast search algorithm with run-time complexity of $O(logn)$. The algorithm works by repeatedly dividing in half the portion of the array that could contain the item, until you've narrowed down the possible locations to just one. Does this ring a bell from CS101?

Although this program has some error(s) associated with it. Use `gdb` (and your brain) to find and fix the error(s).

The working directory also contains a folder for testcases. Run your program against each of these corresponding inputs and match the expected outputs.

Note: These testcases are not exhaustive. It is possible that your program is working fine for these testcases but still may fail on other inputs. So, make sure to test your program on self created testcases as well (This is a good practice in general).

Usage Instruction:

```
Compile:
g++ binary.cpp -o binary
Run:
./binary
```

Input: A single integer n representing the size of the array. The next n integers are the elements of the array in sorted order. The next integer is the number to be searched in the array.

Output: A single integer representing the position (0-based index) of the number in the array. If the number is not found, print -1.

Examples:

Input 1:

```
5
1 2 3 4 5
3
```

Output 1:

```
2
```

Input 2: `5`

```
1 2 3 4 5
6
```

Output 2:

```
-1
```

Explanation:

1. In the first example, the array is `[1, 2, 3, 4, 5]` and the key is `3`. The output is `2` as `3` is present at index `2`.
2. In the second example, the array is `[1, 2, 3, 4, 5]` and the key is `6`. The output is `-1` as `6` is not present in the array.

Problem 3: The Cryptic Fast Modular Exponentiation: Episode 2

Thankfully, your quick nerves helped the duo escape the mysterious room with the correct scroll just in time. The scroll they found is a cryptic one and decoding it requires an efficient calculation of large exponents. The scroll contains a number a and a number b and you need to calculate a^b modulo $10^9 + 7$. To solve this problem, Sabyasachi writes a code based on the fast modular exponentiation algorithm but Guramrit figures out that Sabyasachi has made mistake(s) in the code. Can you help him to correct the code?

In this activity, we have implemented fast modular exponentiation. The program takes two integers x and y as input and calculates x^y % 1000000007 using recursive fast modular exponentiation. Understand the code by looking at the `modular.cpp` file.

Although this program has some error(s) associated with it. Use `gdb` (and your brain) to find and fix the error(s).

The working directory also contains a folder for testcases. Run your program against each of these corresponding inputs and match the expected outputs.

Note: These testcases are not exhaustive. It is possible that your program is working fine for these testcases but still may fail on other inputs. So, make sure to test your program on self created testcases as well (This is a good practice in general).

Resource: Modular Arithmetic

Usage Instruction:
```
Compile:
g++ modular.cpp -o modular
Run:
./modular
```

Input: Two integers x and y $(1 \leq x, y \leq 2^{31} - 1)$

Output: A single integer representing (x^y) % 1000000007

Example:

Input 1:
2 3

Output 1:
```
8
```

Input 2:
```
2 32
```

Output:

```
294967268
```

Explanation:

$2^{32} = 4294967296$
$4294967296 \ \% \ 1000000007 = 294967268$

Problem 4: The Dance of Linked Circles: Episode 3

Many thanks to your unmatched wits, the duo were able to find the coordinates of a hidden door which took them into an enigmatic world. The world is a vast, dark, and empty space, with nothing but a series of linked circles. The duo is told that these circular linked lists hold the key to return back to their world. Guramrit with his deep knowledge of data structures and algorithms took charge of these task, but a mischievous alien named Kirmada wreaked havoc on their code, creating infinite loops among several other disruptions in the logic flow of the code. The duo needs your help to fix the code and return back to their world. Can you help them?

In this activity, we have implemented circular linked list. The program takes input from the user and performs the following operations:

1. `PUSH_BACK`: Add a new node at the end of the list
2. `PUSH_FRONT`: Add a new node at the beginning of the list
3. `DELETE`: Delete the first occurrence of the given data from the list
4. `DISPLAY`: Display the list
5. `EXIT`: Exit the program

Although this program has some error(s) associated with it. Use `gdb` (and your brain) to find and fix the error(s). The working directory also contains a folder for testcases. Run your program against each of these corresponding inputs and match the expected outputs.

Note: These testcases are not exhaustive. It is possible that your program is working fine for these testcases but still may fail on other inputs. So, make sure to test your program on self created testcases as well (This is a good practice in general).

Resource: Circular Double Linked Lists

Usage Instructions:
```
Compile:
g++ circular.cpp -o circular
Run:
./circular
```

Input: The input will be of the following form:

`1 x` (To push `x` at the end of the list)
`2 x` (To push `x` at the beginning of the list)
`3 x` (To delete the first occurrence of `x` from the list if it exists)
`4` (To display the list)
`5` (To exit the program)

Output: The output will be of the following form:

1. The list, when input is 4
2. No output, when input is 1, 2, 3, 5

Example:

Input:

```
1 1
1 2
1 3
4
2 4
4
1 2
4
3 2
4
5
```

Output:

```
1 2 3
4 1 2 3
4 1 2 3 2
4 1 3 2
```

Explanation:

1. The first input is `1 1`. So, `1` is added to the back of the list. The list becomes `1`
2. The second input is `1 2`. So, `2` is added to the back of the list. The list becomes `1 2`
3. The third input is `1 3`. So, `3` is added to the back of the list. The list becomes `1 2 3`
4. The fourth input is `4`. So, the list is displayed. The list is `1 2 3`
5. The fifth input is `2 4`. So, `4` is added to the front of the list. The list becomes `4 1 2 3`
6. The sixth input is `4`. So, the list is displayed. The list is `4 1 2 3`
7. The seventh input is `1 2`. So, `2` is added to the back of the list. The list becomes `4 1 2 3 2`
8. The eighth input is `4`. So, the list is displayed. The list is `4 1 2 3 2`

9. The ninth input is `3 2`. So, the first occurrence of `2` is deleted from the list. The list becomes `4 1 3 2`

10. The tenth input is `4`. So, the list is displayed. The list is `4 1 3 2`

11. The eleventh input is `5`. So, the program exits.

Problem 5: The File System Odyssey: Episode 4

It's only your cleverness that Guramrit and Sabyasachi are alive. Although, they successfully managed to escape from the clutches of the evil Kirmada but there is permanent damage to the path back to IIT Bombay. Instead, they are transported to an alien landscape where a pecular extraterrestrial species guards the last clue to the path back to IIT Bombay. Machine expert Sabyasachi identifies that the alien species sought to assimilate file system's power for their cosmic computation. So, Guramrit builds a very basic file system to simulate and hack into the alien system to retrieve the last clue. The file system suffers with some minor bugs, they seek your help to fix those bugs and retrieve the last clue. It's a do or die situation for them. Can you help them?

In this activity, we have implemented a simple file system. The file system has the following commands:

1. `touch <path>` - creates a file at the given path

2. `mkdir <path>` - creates a directory at the given path

3. `rm <path>` - removes the file or directory at the given path

4. `cd <path>` - navigates to the directory at the given path (We only support relative paths and also support `..` to navigate to the parent directory but do not support `../..`)

5. `ls` - lists the contents of the current directory (No arguments can be passed to this command)

6. `pwd` - prints the current working directory (This only prints the name of `pwd`, not the path)

Although this program has some error(s) associated with it. Use `gdb` (and your brain) to find and fix the error(s). The working directory also contains a folder for testcases. Run your program against each of these corresponding inputs and match the expected outputs.

Note: These testcases are not exhaustive. It is possible that your program is working fine for these testcases but still may fail on other inputs. So, make sure to test your program on self created testcases as well (This is a good practice in general).

Big Note: This program has very basic bugs.

Usage instructions:

```
Compile:
g++ fs.cpp -o fs
Run:
./fs
```

Input: The input will be of the following form:

1. `touch <path>` (To create a file at the given path)
2. `mkdir <path>` (To create a directory at the given path)
3. `rm <path>` (To remove the file or directory at the given path)
4. `cd <path>` (To navigate to the directory at the given path)
5. `ls` (To list the contents of the current directory)
6. `pwd` (To print the name of current working directory)
7. `exit` (To exit the program)

Output: The output will be of the following form:

1. The contents of the current directory when input is `ls`
2. The name of the current working directory when input is `pwd`
3. No output when input is `touch`, `mkdir`, `rm` and exit
4. Navigation message when input is `cd`

There are error messages for invalid commands and invalid paths which you can see through the code.

Example:

Input:
mkdir dir1
cd dir1
touch file1
ls
touch file1
rm file1
ls
pwd
cd ..
ls
pwd
cd ..
ls

exit

Output:
Look at image `Q5.png`.

Explanation: Well, does it need any explanation? :)