CO222: Programming Methodology
Lab: 11 - Part 2

Deadline: June 22nd 2020 @ 11.55PM

Objective 1: Introduction to the debugging tool (GDB) to find bugs (errors) in your program

Bugs: Errors are very common while developing a program. These errors may prevent the program from running correctly. Debugging and finding those mistakes is an important part of programming. Basically there are three types of errors.

Types of errors

- Compile-time errors
- Runtime errors
- Logic errors

Usually, the errors become more difficult to find and fix as you move down the above list.

Compile-time errors:

These errors occur because of wrongly typed statements, which are not according to the syntax or rules (grammatical rules) of the language. The compiler detects these errors at the time of compilation of the program. Such errors are easy to detect and remove.

Some example in C programs such errors are:

- Missing the semicolon at the end of a statement
- Using a variable that has not been declared
- Not including the header file for a function

Runtime errors:

These errors occur during the execution of the programs although the program is free from compile-time errors. These typically occur when your program attempts an operation that is impossible to carry out.

Some of the most common reasons for these errors are:

- Division by zero
- Lack of free memory space
- Input data is not in the correct format

There is no way for the compiler to know about these kinds of errors when the program is compiled. Runtime errors are usually more difficult to find and fix than compile-time errors.

Logical errors:

Your code may compile and run without errors, but the result of an operation may produce a result that you did not expect.

Common examples are:

- The sequence of instructions used in a program may be incorrect
- Mathematical formulas used in program instructions may be incorrect

Logical errors could not be detected by the compiler, and thus, programmers have to check the entire coding of a program line by line. It is a very time consuming and lengthy process.

Debugging

The process of finding and removing bugs in a program is known as debugging. It is the process of removing bugs.

- Use print statements to print variable values
- Use a debugging tool

GDB is the standard debugger for the GNU operating system and works for many programming languages. (C, C++, Fortran, Java etc...)

Note: Testing is different

- **Testing** is the process of finding bugs. You should have to have a functioning program to test.

[P.T.O]

GDB (GNU Debugger)

GDB allows you to see what is going on 'inside' a program while it executes or what the program was doing at the moment it crashed.

- Using the GCC compiler on GNU/Linux, the code must be compiled using the **–g flag** in order to include appropriate debug information in the binary.

  gcc –o <ex3> <ex3.c> –g

- Load the program:

  gdb <ex3>

- Run the program

  run <arguments>

  short version:

  r <arguments>

If the program finished running, it would print "exited normally" message. Otherwise, it will print when it is broken.
Now you can see at which line it crashed. There is a big list of GDB commands but following commands are among the more useful gdb commands

- Display the source code of the compiled file:

  list

- Setting a breakpoint:
  If we set a breakpoint, we could stop a program at the desired line.

  break <line_number>
  break <function_name>
  break <where> if <condition>

  If you have multiple files in your program

  break <filename> <line_number>

- Printing variable values:
  Once you have reached a breakpoint, you can inspect the current values of variables there.

  print <variable_name>

  Keep printing the variable value at each step

  display

  To remove

  undisplay

- Stepping:
  Allows you to go step by step after a breakpoint (All command use the first letter as the short format)

  ➔ ***step***
  - into functions

- step to the next line. If there is a function call, it will step into the function also

➔ ***next***

- over functions
- step to the next line. This command will not go inside functions

➔ ***finish***

- finish the current function and return

- Continue from the breakpoint:
  continue

- Show all the breakpoints
  info break
  or
  info b

- Clear a breakpoint:
  clear <breakpoint number>

- Delete all breakpoints:
  delete

- Change the value of a given variable:
  set <variable_name> = <value>

- Calls a specific function:
  call <function_name>

- Show the parent function of the current function:
  backtrace
  (arguments are also shown)

- Watch a variable when the value changes:
  watch

- Quit from gdb
  q

Use the source codes given in lab11_part2.zip to explore the functionalities above and fix any bugs in the codes. Write down a small summary of the found/ fixed bugs along with the relevant source file name in a text file.

Submit **a single text file** with answers named as **e17XXX_answers.txt** where XXX is your registration number **to the link given on FEeLS.**

[P.T.O]

Objective 2: Programming with File IO

Write a C program to copy contents of one file to another using basic C File IO techniques.

The User Interface should be as follows:

```
Enter the filename to read:
inFile.txt
Enter the filename to write:
outFile.txt

Contents copied to outFile.txt
```

Figure 1: Basic UI

```
Enter the filename to read:
input.txt
Cannot open file input.txt
```

Figure 2: Error in file opening

Submit **a single C file** named **e17XXX_lab11.c** where XXX is your registration number **to the link given on FEeLS**

(You should submit both e17XXX_answers.txt & e17XXX_lab11.c to the same FEeLS link)

Please follow the file naming conventions thoroughly, otherwise, you will get zero marks for Lab11.