

1

Introduction to Computing

Department of Electrical Engineering
University of Engineering and Technology Lahore 2016
Instructor: Kh. Shahzada Shahid

Name _____

Registration Number _____

Procedure

1. Students should read the Pre-lab Reading before coming to lab.
2. In the lab, students should complete Labs activities 1.1 through 1.3 in sequence. Your instructor will give further instructions regarding completion and grading of the lab.

Purpose

1. To understand the basics of Computer System Programming
2. To learn the usage of Cygwin.
3. To compose and run a simple program from scratch.

Pre-lab Reading

Computer Systems:

A computer system consists of all the components (hardware and software) used to take information from the computer users, process it and generate output. It is generally broken down into five basic components:

Arithmetic Logic Unit (ALU): The arithmetic and logic unit (ALU) of a computer system is the place where the actual calculations are performed and all comparisons (decisions) are made.

Central Processing Unit (CPU): The control Unit and ALU are jointly known as the Central Processing Unit. The CPU is the brain of any computer system. In a human body, all major decisions are taken by the brain and the other parts of the body act as directed by the brain. Similarly, in a computer system, all major calculations and comparisons are made inside the CPU and the CPU is also responsible for activating and controlling the operations of other units of a computer system.

Main Memory:	The main memory of the computer is also known as RAM, standing for Random Access Memory. It provides temporary application data storage. Instead of having to go back and find information on the hard drive every time a computer needs a piece of data or instructions, the computer temporarily stores frequently used files on RAM, making them easier to find. When the computer shuts down, all the data in RAM is erased or transferred to the hard drive, making room for new data when the computer begins operating again.
Secondary Storage:	The most common form of external memory is a hard disc which is permanently installed in the computer and typically has a capacity of hundreds of megabytes. It is sometimes called <i>backing store</i> or <i>secondary memory</i> . It stores programs or data not actively being used by the other units. It retains the information even when the computer's power is turned off.
Input Unit:	This "receiving" section obtains information (data and computer programs) from input devices such as keyboards, touch screens and mouse devices.
Output Unit:	This "shipping" section takes information that the computer has processed and places it on various output devices to make it available for use outside the computer. Most information that's output from computers today is displayed on screens, printed on paper, played as audio or video on PCs. "receiving" section obtains information (data and computer programs) from input devices such as keyboards, touch screens and mouse devices.

Program is a set of instructions used to solve a particular problem. A computer is a dumb machine. It cannot do anything on its own. It can only act on the given instructions.

Programing:

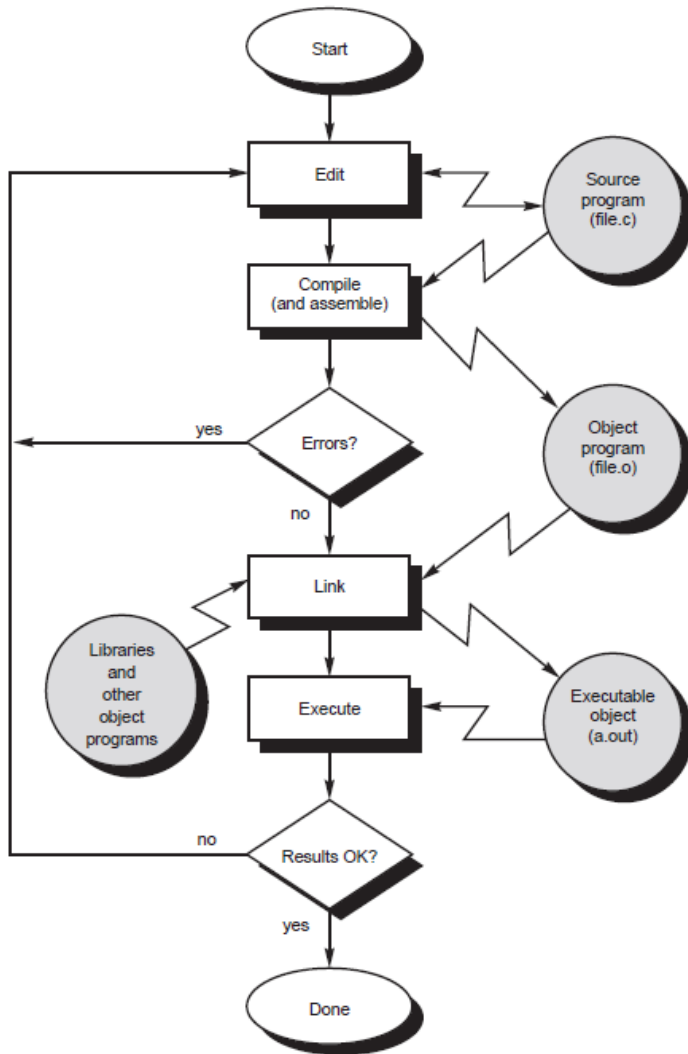
Computer can only understand a sequence of 1s and 0s. The following looks strange to use but, in fact is how computer reads and executes everything that it does:

10101000111001010101011100010101110

This is called Machine language. We can imagine how complicated programming would be if we had to learn this complex language. That in fact, was how programming was done many years ago; however today we are fortunate to have what are called high level languages such as C, java and C++ etc. These languages are more geared for human understanding and thus make programming much easier. However computer only understands machine language, there must be a translation process to convert these high level languages to machine code. This is often done by a **compiler**. Compiler analyzes a program developed in a particular computer (high level) language and then translates it into a form (machine language) that is suitable for execution on your particular computer system. The program that is to be compiled is first typed into a file on the computer system. File should be saved with file extension ".c". For example, the name *prog1.c* might be a valid filename for a C program on your system. A text editor (e.g., Notepad++) is usually used to enter the C program into a file. The program that is entered into the file is known as the *source program* because it represents the original form of the program expressed in the C language. After the source program has been entered into a file, you can then proceed to have it compiled.

In the first step of the compilation process, the compiler examines each program statement contained in the source program and checks it to ensure that it conforms to the syntax and semantics of the language. If any mistakes are discovered by the compiler during this phase, they are reported to the user and the compilation

process ends right there. The errors then have to be corrected in the source program (with the use of an editor), and the compilation process must be restarted. Typical errors reported during this phase of compilation might be due to an expression that has unbalanced parentheses (**syntactic error**), or due to the use of a variable that is not “defined” (**semantic error**).



When all the syntactic and semantic errors have been removed from the program, the compiler then proceeds to take each statement of the program and translates it into a “lower” form. On most systems, this means that each statement is translated by the compiler into the equivalent statement(s) in assembly language needed to perform the identical task. After the program has been translated into an equivalent assembly language program, the next step in the compilation process is to translate the assembly language statements into actual machine instructions. This step might or might not involve the execution of a separate program known as an *assembler*. On most systems, the assembler is executed automatically as part of the compilation process. The assembler takes each assembly language statement and converts it into a binary format known as object code, which is then written into another file on the system. This file typically has the same name as the source file under Unix, with the last letter an “o” (for object) instead of a “c”. Under Windows, the suffix letters “obj” typically replace the “c” in the filename.

After the program has been translated into object code, it is ready to be *linked*. This process is once again performed automatically along with the compilation process. The purpose of the linking phase is to get the program into a final form for execution on the computer. If the program uses other programs that were previously processed by the compiler, then during this phase the programs are linked together. The process of *compiling* and *linking* a program is often called *building*. The final

linked file, which is in an executable object code format, is stored in another file on the system, ready to be run or executed. Under Unix, this file is called a.out by default. Under Windows, the executable file usually has the same name as the source file, with the c extension replaced by an “exe” extension.

Once we have executable code, the program is ready to be run. Hopefully it will run and everything will be fine; however that is not always the case. During “run time”, we may encounter another kind of error called a **run time error**. This error occurs when we ask computer to do something it cannot do, the computer cannot violate the laws of mathematics and other binding restrictions. Asking the computer to divide a number by zero is an example of a run time error. We get executable code; however, when the program tries to execute the command to divide by zero, the program will stop with a run time error. Run time errors, are usually more challenging to find than syntax errors. Once we run our program and get neither syntax nor run time errors, are we free to rejoice? Not exactly. Unfortunately, we may still encounter the worst type of error: the dreaded Logic error. Whenever we ask the computer to do something, but mean for it to do something else, we have a logic error. There must be precise and clear instructions that generate our intentions to the computer. The computer only does what we ask it to do.

Asking it to multiply by 3 when we want something doubled is an example of a **logic error**. This type of error is difficult to identify as compiler cannot detect these errors, the only possible way is to check that the output/result of your program matches with your expectations or not.

Operating System:

An operating system is a program that controls the entire operation of a computer system. All input and output (that is, I/O) operations that are performed on a computer system are channeled through the operating system. The operating system must also manage the computer system's resources and must handle the execution of programs. Microsoft Windows, Linux and Mac OS X are typical example of operating systems.

Program Development Environment:

Many development environments are available in which you can compile, build and run C applications, such as GNU C, Dev C++, Microsoft Visual C++, CodeLite, NetBeans, Eclipse, Xcode, etc. Most C++ development environments can compile both C and C++ programs.

GNU Compiler (GCC):

GCC is a compiler system produced by GNU Project. The Free Software Foundation distributes GCC under GNU Public License. It has played an important role in the growth of free software.

In this lab, we will be using GCC compiler over Windows Operating system through Cygwin software. **Cygwin** is a Unix-like environment and command line interface for Microsoft Windows. And Notepad++ will be used for writing C language code. For installation of Cygwin refer to Appendix A and for notepad++ refer to Appendix B.

Cygwin Terminal Commands:

Before you start compiling your first C program, you need to know a little about basic terminal commands. If you are already familiar with these commands then skip this section and move to the next section.

Command	Description
pwd	The pwd command will allow you to know in which directory you're located (pwd stands for "print working directory"). Example: "pwd" in the Desktop directory will show "~/Desktop".
ls	The ls command will show you ('list') the files in your current directory. Used with certain options, you can see sizes of files, when files were made, and permissions of files. Example: "ls ~" will show you the files that are in your home directory.
cd	The cd command will allow you to change directories. When you open a terminal you will be in your home directory. To move around the file system you will use cd. Examples: 1. To navigate into the root directory, use "cd /" 2. To navigate to your home directory, use "cd" or "cd ~" 3. To navigate up one directory level, use "cd .." 4. To navigate to the previous directory (or back), use "cd -" 5. To navigate through multiple levels of directory at once, specify the full directory path that you want to go to.
cp	The cp command will make a copy of a file for you. Example: "cp file usama" will make an exact copy of "file" and name it "usama", but the file "file" will still be there. If you are copying a directory, you must use "cp -r directory WinSoft".

mv	The mv command will move a file to a different location or will rename a file. Examples are as follows: "mv file foo" will rename the file "file" to "foo"." mv foo ~/Desktop" will move the file "foo" to your Desktop directory, but it will not rename it. You must specify a new file name to rename a file.
rm	Use this command to remove or delete a file in your directory.
rmdir	The rmdir command will delete an empty directory. To delete a directory and all of its contents recursively, use "rm -r" instead.
mkdir	The mkdir command will allow you to create directories. Example:"mkdir music"will create a directory called "music".
man	The man command is used to show you the manual of other commands. Try "man man" to get the man page form an itself. See the "Man& Getting Help" section down the page for more information.

Pre-lab Assignment

1. Compilers detect _____ errors.
2. Computers fetches instructions from _____ memory for execution
3. Attaching other pre-written routines/functions to your program is done by the _____ process
4. If compilation is successful and there are no run time errors, still you are not getting desired output; this corresponds to _____ error
5. Dividing by zero is an example of _____ error.

Lab Activity

1.1 Writing and Running First Program:

1. Open the Cygwin Terminal from the Desktop. Type the “pwd” and press Enter. It will give you the path of your current directory (folder) with respect to root directory of Cygwin.

```
YourUserName@PC-Name ~  
$ pwd  
/home/YourUserName
```

If you have installed Cygwin in C:/Cygwin64, then the complete path will be as:
C:/Cygwin64/home/YourUserName

2. Create a new directory (folder) named Lab1 by following command:

```
YourUserName@PC-Name ~  
$ mkdir Lab1
```

3. Now move into *Lab1* directory (folder) by issuing following command:

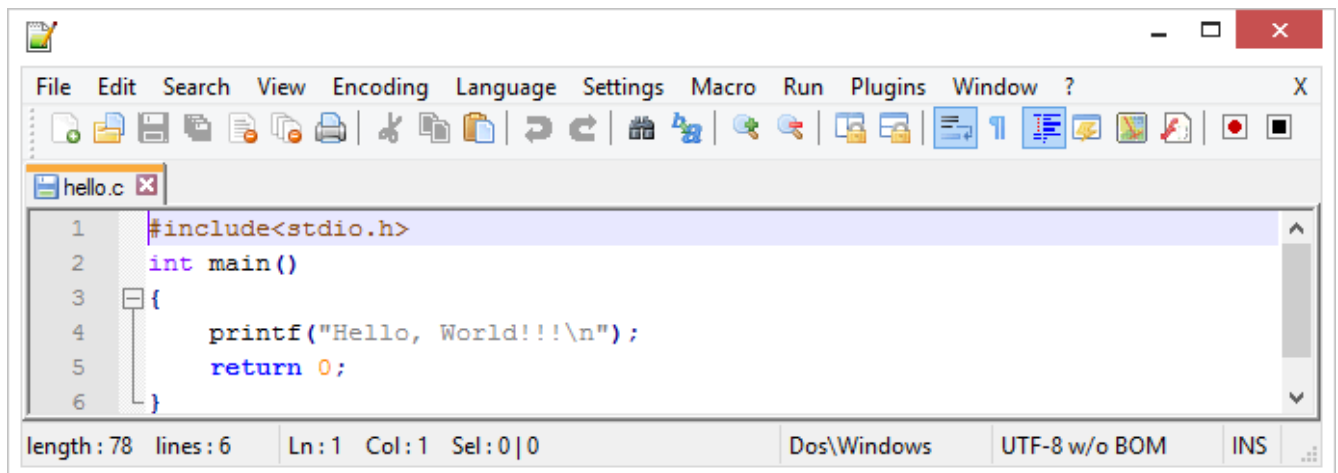
```
YourUserName@PC-Name ~  
$ cd Lab1  
YourUserName@PC-Name ~/Lab1  
$
```

4. Now again enter pwd command

```
YourUserName@PC-Name ~/Lab1  
$ pwd  
/home/YourUserName/Lab1
```

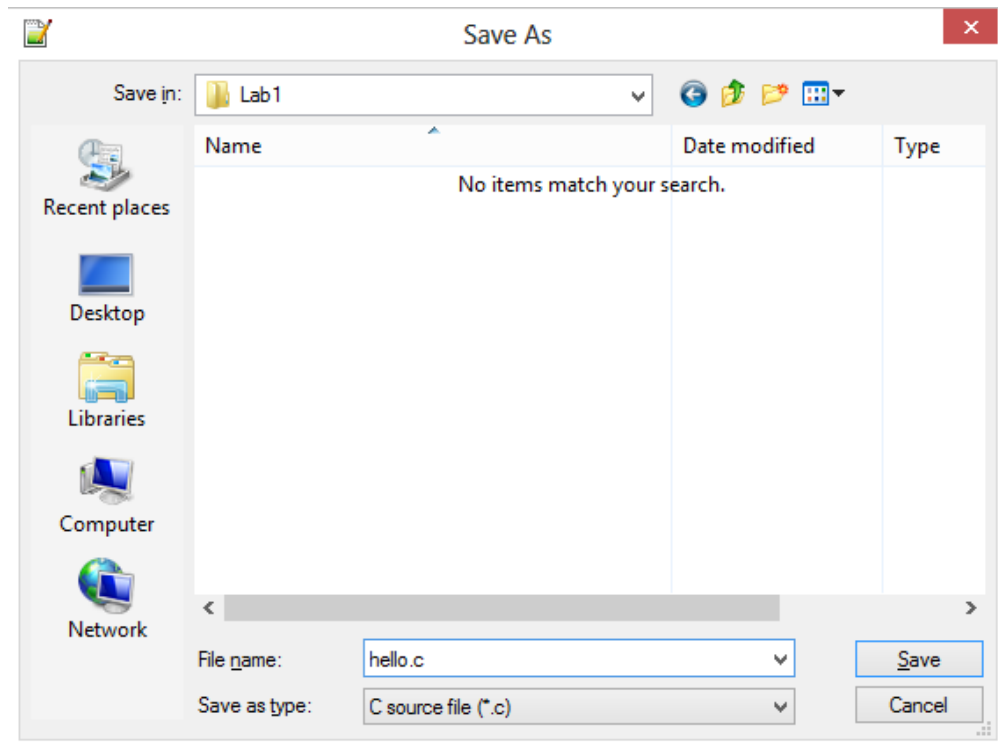
Now your working directory (folder) is “C:/Cygwin64/home/YourUserName/Lab1”. Save all programming files in this directory (folder).

5. Open the Notepad++ from desktop and write the program in C language.



```
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?  
hello.c  
1 #include<stdio.h>  
2 int main()  
3 {  
4     printf("Hello, World!!!\n");  
5     return 0;  
6 }  
length: 78 lines: 6 Ln: 1 Col: 1 Sel: 0|0 Dos\Windows UTF-8 w/o BOM INS
```

6. Click on “File->Save As” and navigate to Lab1 directory **C:\cygwin64\home\YourUserName\Lab1** and save your program by typing any name which should end with extension “.c”. Say our file name is “hello.c”.



7. If you want to edit your code, go to Lab1 directory **C:\cygwin64\home\YourUserName\Lab1** and right click on the file which you have just saved in the above step. Select “Edit with Notepad++” or “open with” and chose open with notepad++. Now your file is open and you can edit it.
8. To compile and run the program, go back to terminal window from the desktop and type the command “ls” to see all files present in Lab1 directory.

```
YourUserName@PC-Name ~/Lab1
$ ls
hello.c
```

9. Type “gcc -c hello.c” and press enter. As a result of this command, “hello.c” will be compile and assembled into “hello.o” file. Now issue ls command to view “hello.o” object file.

```
YourUserName@PC-Name ~/Lab1
$ gcc -c hello.c
YourUserName@PC-Name ~/Lab1
$ ls
hello.c  hello.o
```

10. After successfully compiling and assembling, now it's time to link your assembled "hello.o" file into executable format "hello.out" or "hello.exe". Commands are shown below:

```
YourUserName@PC-Name ~/Lab1
$ gcc hello.o -o hello.out
YourUserName@PC-Name ~/Lab1
$ ls
hello.c  hello.o  hello.out
```

11. [Optional step] You can also do above two steps (Compiling & Assembling and Linking) with a single following command:

```
YourUserName@PC-Name ~/Lab1
$ ls
hello.c
YourUserName@PC-Name ~/Lab1
$ gcc hello.c -o hello.out
SONY@VAIO ~/Lab1
$ ls
hello.c  hello.out
```

12. To run the compiled code, type in "./hello.out".

```
YourUserName@PC-Name ~/Lab1
$ ./hello.out
Hello, world!!!
```

1.2 Write another program that displays your name, registration number and department name.

Write the output of your program below:

1.3 Compile the *GuessNumber* code present in the Lab folder.

Write the errors reported by gcc compiler also mention the type of error in the box given below: