**Lab 01: Getting to Know the 185 Environment**

# Objectives:

* Create the file structure for the lab
* Get familiar with the UNIX-like environment for the labs
* Create and run a basic program

# Starting Point:

* [**lab01-1.c**](https://drive.google.com/open?id=1KNoN4KtreBSTKuuWjX5I1a42RRmE4HCT)
* [**lab01-input.c**](https://drive.google.com/open?id=1GpwUyu1mYWTJTqhhpxCt2wpDcxXcYUdA)
* [**lab01-output.c**](https://drive.google.com/open?id=1Zu0G4CbVs6MaaR2Z46d5Vv1KQ6HuC2C4)

# Turn - In:

**Upload one .pdf file** containing the following before the start of lab next week (Before Lab 2). Homework is to be completed *individually* and *typed* (except base conversions which can be handwritten, but scanned and included in the single PDF submission).

1. The source code for the program you modified in the First Program section and a screenshot of the terminal output.
2. The contents of the next file you created in the UNIX Inter-file Communication section as well as a screenshot of the output.
3. Screenshots of all your code being executed. This can be captured using the snipping tool on available on the lab computers.
4. The companion homework over Base Conversions.

**Upload one .zip** file of your **lab01** folder. Title this **firstname\_lastname\_lab01.zip**. Before zipping, delete all generated .exe files (not the **ds4rd.exe**).

# Process:

You will be using Cygwin, a UNIX-like environment, for your work in this lab.

## File Directory Structure:

1. Open Windows Explorer (“Start>File Explorer>This PC”).
2. Click on U: drive in the new window.

**Important!** Get in the habit of using your U: drive to save all of your files. The U: drive is network storage (10 GB) that you can access from any Windows computer in any Engineering lab on campus; the C: drive is cleared regularly and is only local to that one machine. You will need to routinely access past work no matter the PC. **Saving to the C: drive is unacceptable.**

1. Create a folder in your U: drive called *fall2021*, and within that put a folder named se185*.* **Do not put spaces in the names of the files or directories (folders), as it makes it trickier to use them with the UNIX environment.**
2. Double-click on *se185* to open it.
3. This is your home folder for work with Cygwin for the semester.
4. Create a new folder in *se185 named* *lab01*.

## UNIX Terminal Navigation:

1. Start by opening “Cygwin64 Terminal” (“Start>All Apps>Cygwin>Cygwin64 Terminal”).
2. The terminal works similar to the file explorer, with you working out of a current directory, which is listed as the yellow text above your command input line.
   1. The directory that you are currently working out of is commonly referred to as your “working directory”
3. We will now navigate to the lab01 folder we created in the last section
4. We move to a new directory using the “cd” command which means change directory.
5. Use the following command to navigate to the folder you created using the file explorer: **cd u:fall2021/se185/lab01** Notice that the yellow text now shows the current directory is the one that we navigated to.
6. UNIX commands typically follow this format: *cmd flags arg1 arg2*
   1. *cmd* - the command you wish to run, such as cd
   2. *flags -* typically have a “-” or “--” in front of them, such as “-o”. These are enable options for the command that you wish to run
   3. *arg1* and *arg2* - these are values or files that we wish to run our command with. In the case of the cd command, we had one argument “u:fall2021/se185/lab01”, which was the directory that we wanted to navigate to
7. The next command we will cover is the **ls**, list directory, command. This command will print out the contents of the current directory to the command line. Try it now...notice it does not print anything out as, the lab1 folder should be empty.
8. Now go to the File Explorer and create a new text file in the U:fall2021/se185/lab01 directory. Use the **ls** command again. Now it should print the name of the text file you just created.

**Note:** The terminal will keep a record of your previously typed commands. You can access these previous commands by using the arrow keys to scroll through the list of previous commands. Also, you use tab to have the terminal attempt to autocomplete a word in a command.

## First Program:

You will modify, compile, and execute a simple C program. We will be using the **gcc compiler** on the command line. This will translate the source code (.c file) into a program which the computer will be able run (e.g. an .exe file in Windows).

1. Download and save **lab01-1.c** to your lab01 folder. Open this file in Notepad++ or another text editor/IDE of your choosing. Two other good options are Visual Studio (already installed on the computers) or CLion (which every student can get for free with their Iowa State email).
   1. Do NOT copy it from your web browser- you will run into errors later on when trying to compile your code.
2. Edit the **printf** statement on line 27 to replace **###YOUR MESSAGE HERE###** with a message of your choice. Now save your file with the keyboard shortcut “**ctrl+s**” or click on the blue floppy drive icon.
3. Great! Now you have written your first program. However, if we want to actually run the program, we have to translate it from the human readable C code into machine executable code. This is achieved through **compiling** the code using the **gcc compiler**.
4. In the Cygwin terminal, **make sure you are in your lab1 directory**. Compile the C program you just modified with the following command:

gcc lab01-1.c –o lab01-1

1. If compilation was successful, a new file **lab01-1.exe** should appear in your lab01 folder. The file was created by the **gcc** compiler, we defined the new file’s name with **-o lab01-1**, and the source code (.c file) to compile as **lab01-1.c**.

The **-o** flag defines the output file name, so if we type the command as

gcc lab01-1.c –o howdy

We would end up with a new file **howdy.exe**.Whatever argument you put after the “-o” will be your executable. Do NOT put your “.c” file after the “-o”, otherwise you lose your source code (your source code will be compiled into the “.c” file which will not work).

1. You can run the lab1 program with either of the following commands:

./lab01-1 *or*

./lab01-1.exe *or*

(whatever you may have named the output file)

1. You should see your message printed out on the terminal. Take a screenshot of this terminal output using the Snipping Tool. (“Start>Windows Accessories>Snipping Tool”).

**Note:** Code must be saved and recompiled every time you make a change to your code so the new version of your program runs.

## Input and Output Program:

Now that you have dipped your toes into programming by creating a simple program, we will create something slightly more complex. Investigate two programs, one which **creates output** and one that **reads input**.

1. Download the two source code files: **lab01-output.c** and **lab01-input.c**
2. Open both of the files in Notepad++ or the text editor/IDE of your choice.
3. Note that the **lab01-ouput.c** file contains two important lines of code on lines 19 and 21.
   1. On line 19, we have a variable declaration of variable number and we assign it a value of 0.
   2. On line 21, we have a **printf** statement, the important parts are inside the two parentheses. Notice that we have the name of our variable, **number**, after the comma. This means that we are using the variable as a part of our **printf** statement. The two characters inside the quotes are called a **format specifier**. This is needed when we want to print the contents of a variable.
4. Change the assignment of the variable from 0 to some other number and change the text **###YOUR NETID HERE###** on line 24 to your NetID. Compile and run the code. Note that the output should be what you changed the variable value to and your NetID.
5. In the **lab01-input.c** file we have several **printf** statements, **scanf** statements, and variable declarations.
   1. The **scanf** statement on line 28 allows us to read input into the program. In this case the input will be stored as an integer into the variable.   
      **Note:** when using **scanf**, you need to append an **&** before the name of the variable (for most variables). We will cover the reason for this in a future lecture.
   2. The **scanf** statement on line 31 works with the variable on line 21.
6. Compile and run **lab01-input.c**. Notice that the program appears to freeze after the first printf statement. This is because it is waiting for user input. Type a number into the terminal and hit enter. Now it will wait for you to type in your NetID. Do this as well. The program should print out a formatted message and end.

### Examples: *(if the values typed were 15 and se185)*

Running **lab01-output.exe**

|  |
| --- |
| **$** gcc lab01-output.c –o lab01-output **$** ./lab01-output  15  se185 |

Running **lab01-input.exe**

|  |
| --- |
| **$** gcc lab01-input.c -o lab01-input **$** ./lab01-input  Value before input: 0   Type a number: 15  Type your NetID: se185    Your input was 15 and your ISU email is se185@iastate.edu |

## UNIX Inter-file Communication:

1. One feature of UNIX that you will have to use in later labs is inter-file and command communication. Your TA should have provided a brief demo and explanation at the beginning of lab.
2. Redirection allows us to take the output of a command and insert it into a file. It also allows us to take the contents of a file and insert it into a command’s input. The two symbols we use are “<” and “>”.
   1. Example:  
      ./**lab01-output > mydata.txt**   
      This line takes the output of the program “lab01-output” and creates or overwrites it into a file called mydata.txt. What happens if you have multiple words in your printf on line 24?  
        
      **./lab01-input < mydata.txt**   
      This line takes the contents of mydata.txt and makes it the input of lab01-input. Try these two commands.
3. Piping allows us to take the output of one command/program and send it to the input of another program. This is different from redirection, as it deals with communication between programs, whereas redirection deals with communication between programs and files. The symbol for piping is “|”.
   1. Example:   
       ./**lab01-output | ./lab01-input**  
        
      This line takes the output of output and inserts it as the input of input. Try this out.
4. We can combine the usage of both redirection and piping to run pipe output between programs and save output.
   1. Example:  
       ./**lab01-output | ./lab01-input > NETID.txt**This example is similar to the example in the last section, except it saves the output to file named NETID.txt. Try running this command, replacing NETID with your NetID. When you run the command, no text will be output to the terminal. If you open the text file, you will see the terminal output.
   2. Take a screenshot of the empty terminal as well as the properly formatted text file to include in your lab report.

## PDFs:

For this lab and the following labs, you can generate a PDF for the lab reports using Microsoft Word. A formal lab report format is not required for Lab 01, but will be required for all other labs.

# Lab 01: Companion Homework

## Base Conversion

Learning binary and other numbering systems is an important skill for computer and software engineers. Write the following in decimal (base 10), binary (base 2), octal (base 8), and hexadecimal (base 16). Show your work ***by hand*** (don’t forget to scan your work and put it in your PDF). Scanners are available in certain labs on campus and the computer lab on the first floor of Parks Library. If you take a picture, be sure that it is easily readable. For clarification, you convert what is given into the three other versions.

1. Decimal
   1. 110
   2. 1010
   3. 4210
   4. 25510
2. Hexadecimal
   1. F16
   2. DF16
   3. 8116
3. Binary
   1. 100100112
   2. 1111112
4. Octal
   1. 228