## UE20CS152:

**PROBLEM SOLVING WITH C LABORATORY (0-0-2-0-1)**

**OBJECTIVE:** In this week, we will discuss in detail about the Shell input/output redirections and GCC compiler commands required for C program executions.

## Introduction:

There are always three default files ,

1. stdin (the keyboard)
2. stdout (the screen)
3. stderr (error messages output to the screen).

These, and any other open files, can be redirected.

Redirection simply means capturing output from a file, command, program, script, or even code block within a script and sending it as input to another file, command, program, or script.

## Exercise I: Redirect Output From The Windows Command Line To A Text File

The most useful ways to log and troubleshoot the behavior of commands or batch jobs that you run on Windows is to redirect output to a file.

When you type a command in the Windows console ([command prompt](https://helpdeskgeek.com/windows-10/how-to-customize-the-command-prompt-in-windows-10/)), the output from that command goes to two separate streams.

* **STDOUT:** Standard Out is where any standard responses from commands go. For example the standard response for the DIR command is a list of files inside a directory.
* **STDERR:** Standard Error is where any error messages go if there’s a problem with the command. For example if there aren’t any files in the directory, the DIR command will output “File Not Found” to the Standard Error stream.

We can redirect output to a file in Windows for both of these output streams

## Task 1: Redirect Standard Output Write to New File

Open the command prompt and type:

dir 1.txt

By default the output always gets displayed on terminal,since terminal is the default output file .But the standard output can also be redirected to a file .

There are two ways you can redirect standard output of a command to a file. The first is to send the command output write to a new file every time you run the command.

## dir 1.txt > clab.txt

The > character tells the console to output **STDOUT** to the file with the name we have provided.

When we [run this command](https://helpdeskgeek.com/help-desk/21-cmd-commands-all-windows-users-should-know/), we will notice that there isn’t any response in the command window except the error that the file doesn’t exist.

This is because the standard output for the command was redirected to a file called clab.txt. The file now exists in the same directory where we ran the command. The standard error output still displays as it normally does.

We can view the standard output that went to the file by typing “clab.txt” in the command window. This will open the text file in your default text file viewer. For most of them, this is usually [Notepad.exe**.**](https://helpdeskgeek.com/how-to/run-notepad-as-administrator-to-avoid-access-is-denied/)

The next time we run the same command, the previous output file will be deleted. A new output file will be recreated with the latest command’s output.

# Task 2: Redirect Standard Output Writes to the Same File

If we don’t want to overwrite the same file then we use another option >> rather than > to redirect to an output file.

## Example: hostname>>clab.txt

In the above example **the command hostname displays the hostname**. In this case, instead of overwriting the output file, this command appends the new output to the existing output file.

Every time we run a command and append the output to a file, it’ll write the new standard output to the end of the existing file**.**

# Task 3: Redirect Standard Error To a File

The same way as we redirect standard output writes to a file, we can also output the standard error stream to a file.

To do this, we will need to add 2> to the end of the command, followed by the output error file you want to create.

## Example: dir hello > clab.txt 2>output.err

This sends the standard output stream to clab.txt, and the standard error stream to output.err. The result is that, no output stream gets displayed in the console window. We can see the error messages by typing **output.err**. This will open the file in your default text file viewer.

We can see, any error messages from the command are output to the error file. Just as with the standard output, we can use **>>** instead to append the error to errors from previously run commands.

## Task 4: Redirect All Output: Writes to a Same File

All of the approaches above result in multiple files. One file is for the standard output stream and the other is for the standard error stream.

If you want to include both of these outputs to the same file, you can do that too. To do this, you just need to redirect all output to the same file using the following command.

## dir test.exe 1> myoutput.txt 2>&1

Here’s how this command works:

* The standard output is directed to the output file identified by output number 1.
* The standard error output identified by the number 2 is redirected to the output file identified by number 1.

This will append the error output to the end of the standard output.

## Task 5: Input Redirection-Taking input from file

Just as the output of a command can be redirected to a file, so can the input of a command be redirected from a file. As the greater-than Character > is used for output redirection, the less-than character < is used to redirect the input of a command.

The commands that normally take their input from the standard input can have their input redirected from a file in this manner.

## For example, the sort command can take the input from the keyboard(Standard Input) itself as shown in the figure below.

The same sort command can be executed by taking the input from a file (Input redirection) by typing the following command:

Example:sort < arrange.txt

**Ubuntu version**

## Exercise I: Redirect Output From The Ubuntu Terminal To A Text File

Most Unix system commands take input from your terminal and send the resulting output back to your terminal. A command normally reads its input from the standard input, which happens to be your terminal by default. Similarly, a command normally writes its output to standard output, which is again your terminal by default.

## Output Redirection

The output from a command normally intended for standard output can be easily diverted to a file instead. This capability is known as **output redirection.**

If the notation > operator(or character) file is appended to any command that normally writes its output to standard output, the output of that command will be written to file instead of your terminal.

**Example1:** Check the following ls command which redirects the complete output of the command in the Clab\_files.txt

$ ls > Clab\_files.txt

## Notice that no output appears at the terminal. This is because the output has been redirected from the default standard output device (i.e, terminal) into the specified file(i,e in this case Clab\_files.txt). You can check the users file for the complete content:-$ cat Clab\_files.txt

**Example2:**If a command has its output redirected to a file and the file already contains some data, that data will be lost **i.e, Content is Overwritten.**

Consider the following example −

$ ls -l > Clab\_files.txt

$ cat Clab\_files.txt

**Example3**:To append the output to an existing file.

If you redirect the output of a command to an existing file,then you can use >> operator (or character)to append the output in an existing file as follows −

$touch Info.txt

$cat Info.txt

$cat > Info.txt Apple

Orange Mango

## [Press Ctrl+D]

$cat Info.txt Apple Orange Mango

$cat >> Info.txt Rose

Lily Jasmine

## [Press Ctrl+D]

$cat Info.txt Apple Orange Mango Rose

Lily Jasmine

## Input Redirection

Just as the output of a command can be redirected to a file, so can the input of a command be redirected from a file. As the greater-than Character > is used for output redirection, the less-than character < is used to redirect the input of a command.

The commands that normally take their input from the standard input can have their input redirected from a file in this manner.

**For example**, to count the number of lines in the file, we can execute the command as follows −

$touch C\_Info.txt

$cat C\_Info.txt

$wc < C\_Info.txt

$wc C\_Info.txt //Also try without redirection operator.

Upon execution, you will receive the following output. You can count the number of lines,word and byte count in the file by redirecting the standard input of the wc command from the file C\_Info.txt.

**Note:** There is a difference in the output produced by the two forms of the wc command. In the second case, the name of the file C\_Info.txt is listed with the count, in the first case, it is not.

In the first case,it only knows that it is reading its input from standard input so it does not display file name. In the second case, wc knows that it is reading its input from the file C\_Info.txt.

## Discard the output

Sometimes you will need to execute a command, but you don't want the output to be displayed on the Terminal or Screen. In such cases, you can discard the output by redirecting it to the file /dev/null

$ command > /dev/null

Here command is the name of the command you want to execute. The file /dev/null is a special file that automatically discards all its input.

To discard both output of a command and its error output, use standard redirection to redirect STDERR to STDOUT −

$ command > /dev/null 2>&1 Here 2 represents STDERR and 1 represents STDOUT.

**Note:** In Unix System,the file descriptor 0 is normally standard input (STDIN- keyboard), 1 is standard output (STDOUT-monitor/display), and 2 is standard error output (STDERR-monitor/display).

## Example for ERROR REDIRECTION using ‘2>’ OPERATOR

Assume you have wrriten your first C program it consist of couple of errors. Normally when you compile your C program, your errors will be displayed on the same terminal.

Suppose we want our program errors to be redirected to the specified file.Here is the example. First C program written in **first\_pgm.c**

## $gedit first\_pgm.c

int main()

{

printf("This is first program in c") return 0;

}

## Compilation:

**$ gcc first\_pgm.c**

first\_pgm.c: In function ‘main’:

first\_pgm.c:3:2: warning: implicit declaration of function ‘print’ [-Wimplicit- function-declaration]

printf("This is first program in c")

^

first\_pgm.c:4:2: error: expected ‘;’ before ‘return’ return 0;

^

Now, if we want these **errors to be redirected**,then are redirected to errorlog.txt file using 2>

$ gcc first\_pgm.c 2 > errorlog.txt

$ cat errorlog.txt

**Exercise II: GCC Complier Commands**

C program execution consist of 2 stages

1. **Compilation:** During compliation it checks all syntax and semantic errors. If in case of any errors it displays all errors on the terimnal. If no errors ,then it generates executable file called a.out
2. **Execution:** During execution a.out file takes relevant input from the user and gives the appropriate output.

Follow the commands given the below to execute any c Program as:-

**Note** more info refer to help on gcc command:-

$gcc –help

**Options in gcc compiler**

1. **-E** Stop after the preprocessing stage; do not run the compiler proper. The output is in the form of preprocessed source code, which is sent to the standard output. Input files that don't require preprocessing are ignored.

**Example:** $ gcc -E first\_pgm.c

1. **-c** Compile or assemble the source files, but do not link. The linking stage simply is not done.The ultimate output is in the form of an object file for each source file. Example: $ gcc -c first\_pgm.c

In the above case object file or executable wont be generated for the further execution.Only compilation will be done.

1. **-o** Place output in file. This applies to whatever sort of output is being produced, whether it be an executable file, an object file, an assembler file or preprocessed C code.

## Example:

gcc first\_pgm.c -o output

**Note :** Some of the commands discussed details:-

1. pgm >> file

Output of pgm is appended to file

1. n > file

Output from stream with descriptor n redirected to file

1. n >> file

Output from stream with descriptor n appended to file

1. n >& m

Merges output from stream n with stream m

1. n <& m

Merges input from stream n with stream m

1. << tag

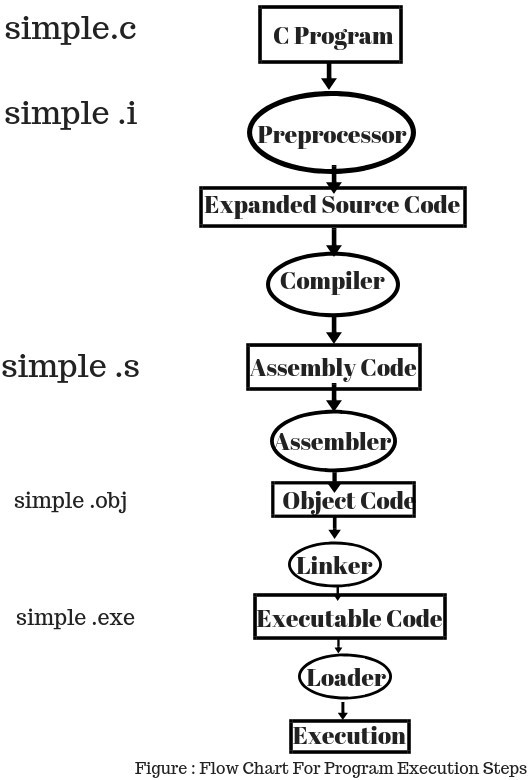
Standard input comes from here through next tag at the start of line

1. Takes output from one program, or process, and sends it to another.
2. List of Commands to revisit:-

man, pwd,date,cal,ls,ls -l,mkdir,rmdir,cd,rm,echo, cat,touch, gedit,wc etc.

,

**C program Execution steps:**



**C Source code**- C program written.

**Preprocessor-** The preprocessor files are attached with our code. We have use different header files like stdio.h, math.h etc. These files are attached with the C Source code and the final C Source generates. (‘#include’, ‘#define’ These are Preprocessor Directives.) .The preprocessor generates an expanded source code.

**Compiler −** After generating the preprocessed source code, it moves to the compiler and the compiler generates the assembly level code after compiling the whole program.

**Assembler** − The assembler takes assembly level language from compiler and generates the Object code, this code is quite similar to machine code (set of binary digits).

**Linker** − Linker is another important part of the compilation process. It takes the object code and link it with other library files, these library files are not the part of code, but it helps to execute the total program. After linking the Linker generates the executable code which is ready to execute.

**Loader** − The executable code is sent to loader which loads it into memory and then it is executed. After execution, output is sent to console.