

CS1010

<http://www.comp.nus.edu.sg/~cs1010/>

Programming Methodology

UNIT 11

UNIX I/O Redirection



NUS
National University
of Singapore

School of
Computing

Unit 11: UNIX I/O Redirection

Objective:

- Learn how to use I/O redirection in UNIX to redirect input from a file and output to a file.

Unit 11: UNIX I/O Redirection

1. Introduction
2. Input Redirection
3. Output Redirection
4. Combining Input and Output Redirection

1. Introduction

- Recall in [Unit #4 Overview of C Programming](#), it is mentioned that the default standard input stream ([stdin](#)) is the keyboard, and the default standard output stream ([stdout](#)) is the monitor.
- In UNIX, you may run a program that normally reads input data interactively to read the input data from a file instead.
- Likewise, you may write the output of a program to a file instead of printing it on the screen.
- This is known as [input/output redirection](#).
- Note that this is an operating system (UNIX) feature and not a C feature.

2. UNIX Input Redirection (1/3)

- Some programs read a lot of input data (eg: programs involving arrays), which makes it very inconvenient for users to key in that large amount of data interactively.
- Instead, we may store the input data in a file, and let the program read the data from that file.
- We may do it in 2 ways:
 - Read the file using **file processing functions** (eg: **fopen()**, **fscanf()**, **fprintf()**) – these will be covered next time
 - **Redirect** the input from the file instead of from **stdin** – we will do this for the moment

2. UNIX Input Redirection (2/3)

Unit11_Example.c

```
#include <stdio.h>

int main(void) {
    int num, sum = 0;

    printf("Enter integers, terminate with ctrl-d:\n");
    while (scanf("%d", &num) == 1) {
        sum += num;
    }
    printf("Sum = %d\n", sum);
    return 0;
}
```

- Running the program interactively:

What does this mean?

```
$ a.out
Enter ... ctrl-d:
5
12
-7
0
23
← User enters ctrl-d here
Sum = 33
```

2. UNIX Input Redirection (3/3)

- Using an editor (eg: vim), create a text file to contain the input data. Let's call the file **numbers**.

File **numbers**

5
12
-7
0
23

- Use the UNIX input redirection operator **<** to redirect input from the file **numbers**

```
$ a.out < numbers  
Enter ... ctrl-d:  
Sum = 33
```

- (This is how CodeCrunch runs your program. It redirects input from some file to feed your program.)

3. UNIX Output Redirection (1/2)

- Instead of printing your output to the default `stdout` (monitor), you may redirect the output to a file as well.
- Use the UNIX output redirection operator `>`.

```
$ a.out > outfile
```

```
5
```

```
12
```

```
-7
```

```
0
```

```
23
```

```
← User enters ctrl-d here
```


3. UNIX Output Redirection (2/2)

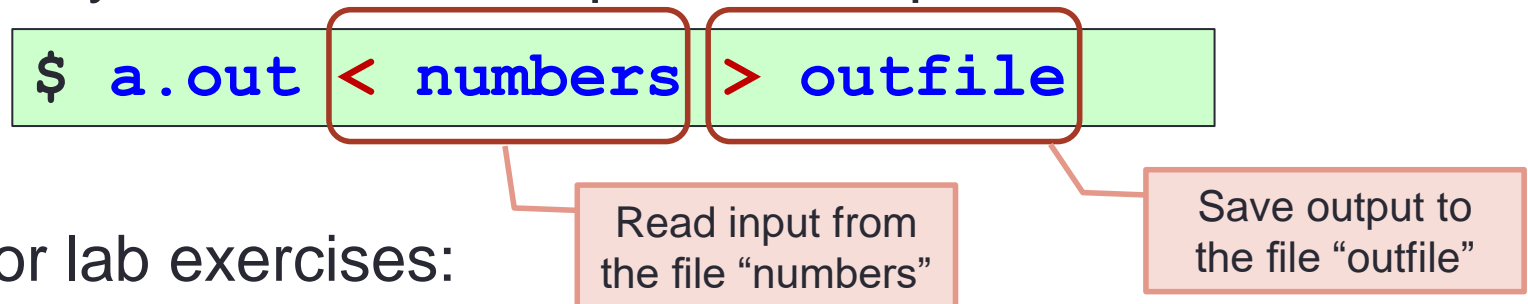
- The file `outfile` is created which captures all outputs of the program.

```
$ cat outfile
Enter integers, terminate with ctrl-d:
Sum = 33
```

- Output redirection `>` fails if the specified output file already exists
- If you want to append the output of a program to an existing file, you may use `>>`

4. Combining Input and Output Redirection

- You may combine both input and output redirection



- Tip for lab exercises:

- Using input redirection, you can download the given input files on the CS1010 website and run your program on these files.
- Using output redirection, you may now generate your own output file and compare it with the expected output file provided on the CS1010 website.
- Use the UNIX `diff` command to compare two files. Example:
`diff file1 file2`
- If the two files compared are identical, no output will be generated by the `diff` command.

Summary

- In this unit, you have learned about
 - Using UNIX input redirection **<** to redirect input from a file to a program
 - Using UNIX output redirection **>** to redirect output of a program to a file

End of File