



SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

CMPT 125 - Introduction to Computing Science and Programming II - Fall 2022

- Passing arguments to main()
- Redirecting stdin and stdout
- Reading letters from input and calculating the frequencies.

Passing arguments to main()

Until now :

Our **main()** functions were not receiving any arguments.

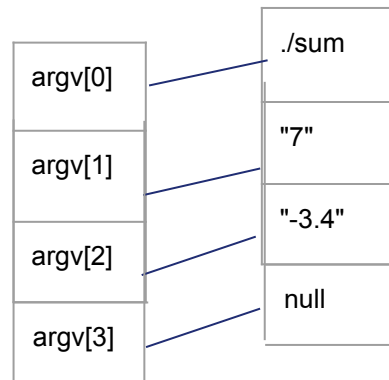
But **main()** similar to other functions can receive arguments, just there is some notes to follow:

- **main()** function can receive two arguments.
- To do that, the main() function should be defined as **int main(int argc, char* argv[])**
- **argv**: Array of strings. The array has size argc+1
- **argv[0]** is the name of the exe file
- **argv[1]** is the first argument
- **argv[0]** is the second argument
- ...
- **argv[argc-1]** is the last argument
- **argv[argc]** is the null
- let say our **main()** function is in **test.c**, we can pass arguments to it using :

```
gcc test.c -o test
```

```
./test argv1 argv2 ... argvn
```

ex: `./sum 7 -3.4`



Lest clear thing up with an example:

Write a function that receives arguments to main and prints the arguments

```
#include <stdio.h>

int main(int argc, char* argv[]) {
    printf("Hello world\n");

    printf("argc = %d\n", argc);
    for (size_t i = 0; i < argc; i++)
        printf("argv[%d] = %s\n", i, argv[i]);

    printf("argv[%d] = %s\n", argc, argv[argc]);
    printf("argv[%d] = %s\n", argc+1, argv[argc+1]);
    return 0;
}
```

Lest clear thing up with an example:

Write a function that receives two integer as main function arguments and return their sum.

```
#include <stdio.h>
#include <stdlib.h> // contains atoi() function

int main(int argc, char* argv[]) {
    if (argc!=3) {
        printf("wrong number of arguments %d", argc);
        return 1;
    }

    int x = atoi(argv[1]); // atoi() gets a string containing an integer and returns an int
    int y = atoi(argv[2]);
    printf("%d + %d = %d", x, y, x+y);
    return 0;
}
```

Exercise 1:

Now write a function the receives three number and return the biggest one.

you can use `>` and `<` to direct your stdin and stdout.

for example:

`./hello > myfile.txt` will redirect your `printf` to `myfile.txt`

so if you have a program that do `printf("Hello World")`, instead of seeing output on terminal you can find it in `myfile.txt`

similarly `./read_numbers < numbers.txt` will use `numbers.txt` to read the input from the file instead of reading from the user

Exercise 2:

Compile a program hello that prints "hello world".

Run: `./hello > hi.txt`

See what hi.txt contains

Modify the file hi.txt

Run again: `./hello > hi.txt`

See what hi.txt contains now

Run again this time use `>>` (double right):

`./hello >> hi.txt`

See what hi.txt contains now

Try again. Conclude about append vs overwriting the target file

Exercise 3:

Write the following program **sum.c**:

```
#include <stdio.h>
```

```
int main() {  
    int x;  
    int y;  
    printf ("Enter x:\n");  
    scanf("%d", &x);  
    printf ("Enter y:\n");  
    scanf("%d", &y);  
    printf("%d + %d = %d", x, y, x+y);  
    return 0;  
}
```

Create a file **nums.txt** that contains several numbers one in each line. For example:

```
8  
5  
12
```

- 1) Compile the program into an executable called **sum**
- 2) Run `./sum`
Enter some numbers when needed.
- 3) Run the program again, this time
`./sum < nums.txt`
This will read the input from the file

Exercise 5:

Write a program called **frequencies** that calculates the frequency of letter occurrences in text.

1. The program gets one argument, and it is a string given as `argv[1]`.
2. Print the string to check that you got the correct string
3. use `> my_output.txt` to create the file that will contain the frequency of each letter in the obtained argument
4. Letters that occur zero times should not appear in the output.
5. Characters other than lower and upper case letters should be ignored.
6. Lower and upper case instances count as the same letter, e.g. 'a' and 'A' are both reported for the letter 'a' on the output.
7. The frequencies reported should sum up to the total length of the string.

In your program declare an array of length 26, where each entry counts the number of appearances of a letter.

Each letter has a numerical ASCII value. Use the `ascii` value to compute the corresponding index in the array.

8. You should not implement this function by writing 26 "if" statements (1 for each letter).

For example: if you run

`./frequencies HihElloAAAA > my_output.txt`

then the output file will look something like this:

```
A 4
E 1
H 2
H 1
L 2
O 1
```