

# ECE2800J

Programming and Introductory Data Structures

## **Passing Arguments to Program**

### **Learning Objectives:**

Know how to write more general programs that can take arguments

# Passing Arguments to Program

## Introduction

- So far, we have considered programs that take no arguments
  - You run your program like: `./program`
- However, programs can take arguments.
- For example, many Linux commands are programs and they take arguments!
  - `diff file1 file2`
  - `rm file`
  - ...

# Passing Arguments to Program

## Introduction

```
diff file1 file2
```

- The first word, `diff`, is the **name** of the program to run.
- The second and third words are **arguments** to the `diff` program.
- These arguments are passed to `diff` for its consideration, like arguments are passed to functions.
- The operating system collects arguments and passes them to the program it executes.

# Passing Arguments to Program

- Arguments are passed to the program through `main()` function.
- We need to change the argument list of `main()`:
  - Old: `int main()`
  - New: `int main(int argc, char *argv[])`

# Passing Arguments to Program

```
int main(int argc, char *argv[])
```

- Each argument is just a sequence of characters.
- All the arguments (including program name) form an array of C-strings.
- `int argc`: the number of strings in the array
  - E.g., `diff file1 file2`: `argc = 3`
  - The name `argc` is by convention and it stands for “argument count”.

# Passing Arguments to Program

```
int main(int argc, char *argv[])
```

- `argv` stores the array of C-strings.
  - Remember, a C-string is itself an array of `char` and it can be thought of as a pointer to `char`.
  - Thus, an array of C-strings can be thought of as an array of pointers to `char`.
  - Thus, `argv` is an array of pointers to `char`: `char *argv[]`
  - The name `argv` is again by convention and it is short for “argument vector” or “argument values”.

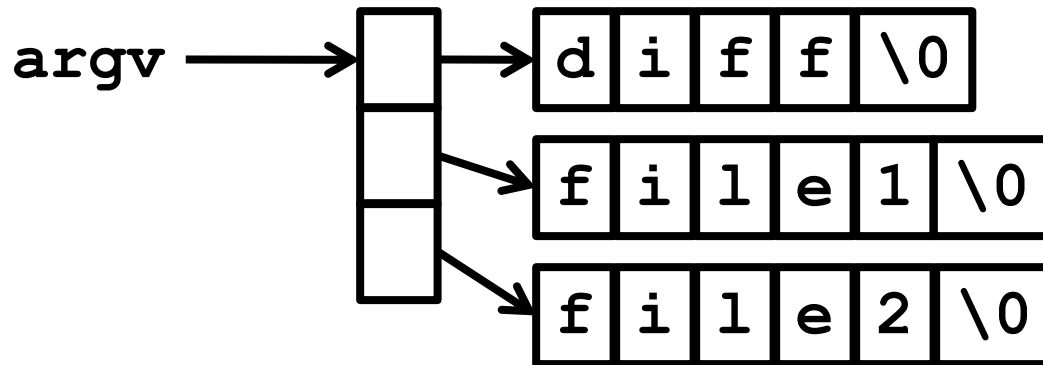
# Passing Arguments to Program

argv

```
diff file1 file2
```

```
char *argv[]
```

- Pictorially, this would look like the following in memory:



**Note:** `argv[0]` is the first string you type to issue the program. It includes the name of the program being executed and optional path (like `./`).

# Passing Arguments to Program

## Example

- Suppose we wanted to write a program that is given a list of integers as its arguments, and prints out the sum of that list.
- Before we can write this program we need a way to convert from C-strings to integers.
- We use predefined “standard library” function called `atoi()`.
- Its specification is

```
int atoi(const char *s);  
// EFFECTS: parses s as a number and  
//          returns its int value
```

- Needs `#include <cstdlib>`



# Passing Arguments to Program

## Example

- The problem we are examining can be solved as:

```
int main (int argc, char *argv[])
{
    int sum = 0;
    for (int i = 1; i < argc; i++) {
        sum += atoi(argv[i]);
    }
    cout << "sum is " << sum;
    return 0;
}
```

# Passing Arguments to Program

## Example

```
int main (int argc, char *argv[]) {  
    int sum = 0;  
    for (int i = 1; i < argc; i++) {  
        sum += atoi(argv[i]);  
    }  
    cout << "sum is " << sum;  
    return 0;  
}
```

- Finally, we save it to `sumIt.cpp`, compile, and run it:

```
$ g++ -o sumIt sumIt.cpp
```

```
$ ./sumIt 3 10 11 12 19
```



For the previous command, select all the correct answers

```
$ ./sumIt 3 10 11 12 19
```

- **A.** argc equals 5.
- **B.** argv contains exactly “3”, “10”, “11”, “12”, “19”.
- **C.** argv[0] equals “sumIt”.
- **D.** The command outputs “sum is 55”.

```
int main (int argc, char *argv[]) {  
    int sum = 0;  
    for (int i = 1; i < argc; i++) {  
        sum += atoi(argv[i]);  
    }  
    cout << "sum is " << sum;  
    return 0;  
}
```



# References

- Command-Line Arguments
  - Absolute C++, 4<sup>th</sup> Edition, Page 373