COMPUTER ENGINEERING LABORATORY

Luigi Rizzo

luigi.rizzo@unipd.it

October 2023-January 2024

UNIVERSITÀ DEGLI STUDI DI PADOVA

Exercises: loops, arrays

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Command-line Arguments

In environments that support C, there is a way to pass command-line arguments or parameters to a program when it begins executing.
Command-line arguments are a way to pass information to a C program from the command line when you run it. These arguments are commonly used for configuration, input data, or other parameters that affect how the program behaves. In C, command-line arguments are received as parameters to the main function.
When main is called, it is called with two arguments.
The first (conventionally called argc, for argument count) is the number of command-line arguments the program was invoked with; the second (argv, for argument vector) is a pointer to an array of character strings that contain the arguments, one per string.

Command-line Arguments

By convention, argv[0] is the name by which the program was invoked, so argc is at least 1.
If argc is 1, there are no command-line arguments after the program name.
Therefore command-line arguments are strings of characters that you provide when running a C program.
The main function has the following signature:

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

You can access and process command-line arguments by using argc to determine the number of arguments and argv to access the argument values

Command-line Arguments

For example, to access the second command-line argument:

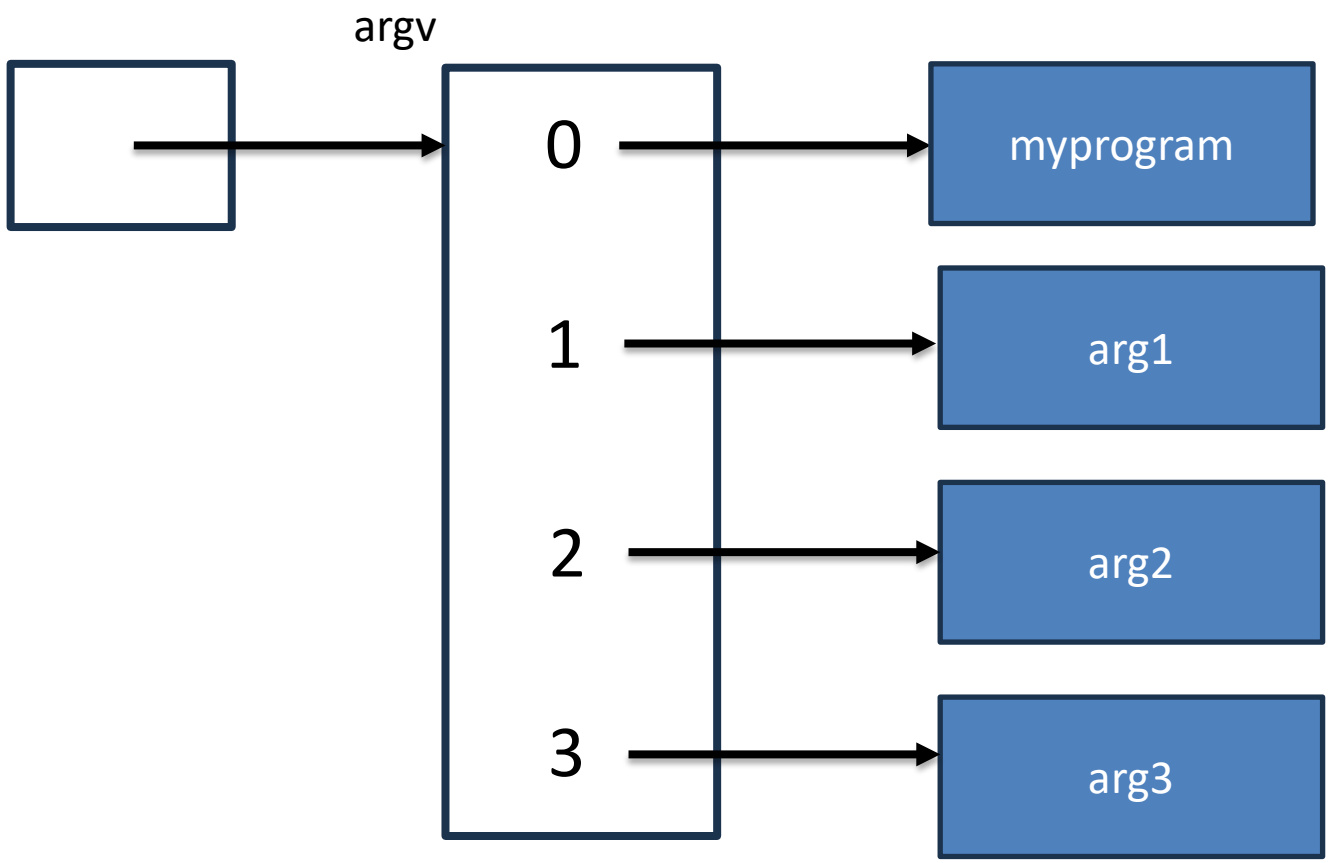
```
char *secondArg = argv[2];
```

To run a C program with command-line arguments, you typically execute it from the command line and provide the arguments after the program name:

```
$ ./myprogram arg1 arg2 arg3
```

In this example, myprogram is the name of the C program, and arg1, arg2, and arg3 are the command-line arguments.

Command-line Arguments




```
for (int i = 1; i < argc; i++)
    printf("%s\t", argv[i]);
```

Command-line Arguments

Command-line arguments are commonly used for various purposes, such as specifying input files, configuration options, debugging flags, and any information that you want to pass to the program when it's executed.
They are especially useful when writing scripts or programs that need to be parameterized without modifying the source code.

When working with command-line arguments, it's essential to perform error checking to ensure that the expected number of arguments is provided and to handle invalid or missing arguments gracefully.

Exercise 1

Write the first 30 elements of a series defined as follows: the first three elements are worth 1, the subsequent ones (i>=4) are worth the sum of the elements i-1 and i-3

Some suggestions:

- you could use an array of integers or you could use 4 int variables

Exercise 2

Write a program that, given a number N>0 (at most 10) of different integer values provided as command-line arguments, prints on the screen the maximum and minimum of the inserted sequence, the position in which this value was inserted and the sequence ordered in ascending mode.

Some suggestions:

- use the function atoi (simpler) or strtol (more complete and robust) to convert the string arguments in integer values

```
for (int i = 1; i < argc; i++)
    printf("%s\t", argv[i]);
```

Exercise 3

Given a string, transform it in a new string, in which every character is located OFFSET positions further in the alphabet

- The alphabet considered is:
  - ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
- The alphabet is cyclical: after the 'z' there is the 'A'
- For example, with OFFSET = 4
  - the character 'a' becomes 'e'
  - the letter 'X' becomes 'b'
  - the letter 'x' becomes 'B'
- The complexity of the program lies in the fact that in ASCII coding the sequences 'A'..'Z' and 'a'..'z' (the sequence of uppercase characters comes first) are not consecutive, but there is half another set of characters, so it is necessary to break the program into 2 ifs.