POLITECNICO DI TORINO

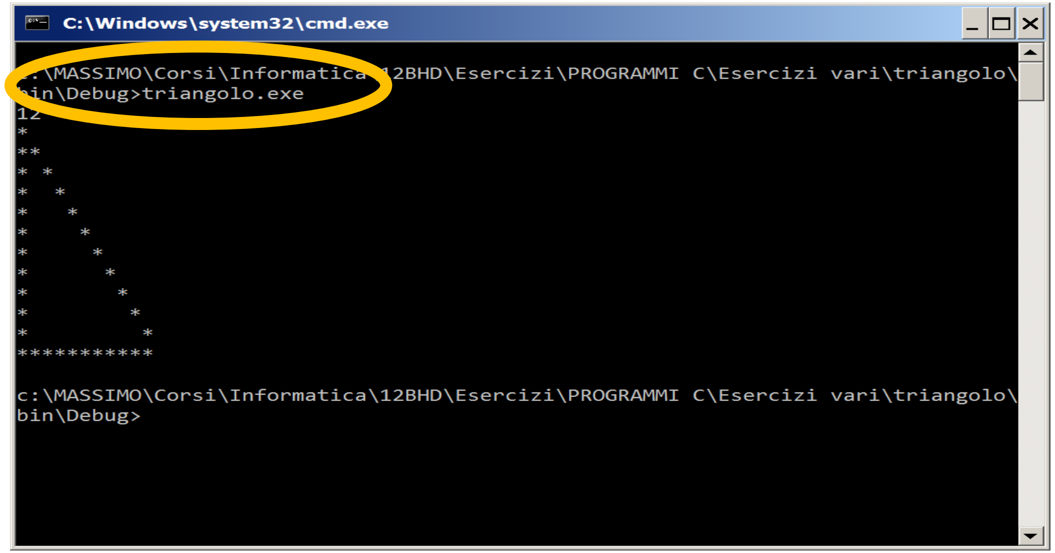
Dipartimento di Automatica e Informatica

Passing arguments to main

COMMAND LINE ARGUMENTS

“Command” user-interface

- The standard paradigm to execute a program is now the “double-click”
- There is an alternate modality, where the program is executed by calling the executable from a textual interface (COMMAND LINE)



```
c:\MASSIMO\Corsi\Informatica\12BHD\Esercizi\PROGRAMMI C\Esercizi vari\triangolo\
bin\Debug>triangolo.exe
12
*
**
* *
*  *
*   *
*    *
*     *
*      *
*       *
*        *
*         *
***********

c:\MASSIMO\Corsi\Informatica\12BHD\Esercizi\PROGRAMMI C\Esercizi vari\triangolo\
bin\Debug>
```

Arguments from the **command line**

- In C, it is possible to pass arguments to a program from the command line
  - Example:

```
C:\> myprog <arg1> <arg2> ... <argN>
```

where myprog is the executable file of a program

- Common in many "interactive" applications
  - Example: Windows (*prompt command*)

```
C:\home>  copy  file1.txt dest.txt
```

  - Example: Os-x (terminal)

```
$ cp file1.txt dest.txt
```

How to activate the terminal in CLion

- CLion has a terminal emulator embedded into it
- You can enable it by View -> Tool Windows -> Terminal
- See instructions in:
  - *https://www.jetbrains.com/help/clion/terminal-emulator.html*
- From the "terminal" window it is possible to run the executable with a textual command, and to specify additional arguments after the name of the executable

You need to verify the current directory, and the one where the executable is…

Arguments in Clion WITHOUT using Terminal

- As an alternative to using the terminal (which does not allow debugging)
  - It is possible to define arguments to the main even in IDE modality (with or without debug)
  - Menu "Run -> Edit Configurations"
  - You need to add the arguments in the window "Program arguments:"
- See full instructions from:
  - *https://www.jetbrains.com/help/clion/run-debug-configuration.html#envvars-progargs*
  - *section "Add program arguments"*

Passing arguments to `main()`

When you execute a program by specifying arguments in the command line, these arguments are **automatically** seen by the program, and received as parameters by the **`main()`** function.

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

- **`argc`**: Number of arguments that are specified in the command line
  - There is always at least an implicit one (name of the program)
- **`argv`**: **Array of strings**
  - **`argv[0]`** = first argument (it is always the name of the program)
  - **`argv[i]`** = generic i-th argument
  - **`argv[argc-1]`** = last argument

Examples

```
C:\progr>square
```

Number of arguments= 1 (argc=1)

```
C:\progr>square 5
```

Number of arguments = 2 (argc=2)

argv[1] contains "5"

```
C:\progr>square 5 K
```

Number of arguments = 3 (argc=3)

argv[1] contains "5"

argv[2] contains "K"

Arguments to the main

The main function has always two formal parameters:

- `argv` (array of strings): `argv[0]` is always present, it contains the name of the executable file of the program
- `argc` (integer): dimension of argv (→ total number of the arguments that appear in the command line)

Example: program that takes the names of two files as arguments

```
int main (int argc, char *argv[]) {
   FILE *fp1, *fp2;
   if (argc!=3) {
       printf("ERROR: the program was not executed with the required arguments\n");
       return 1;
   }
   fp1 = fopen(argv[1],"r");
   fp2 = fopen(argv[2],"r");
   …
```

How to use `argc` and `argv`

- Loop that processes one argument at a time

```
for (i=1; i<argc; i++) {
   /* process argv[i] as a string */
}
```

- NB:
  - No matter the nature of the information you want to pass to the main, argv[i] is **always a string** (that is, if you want to pass the number 5 as argument, it will be received as the string “5”, not as an integer)
  - In case we need a numerical value, we need a way to convert strings into numerical values

Conversion of numerical arguments

In C there are specific functions (defined in **`<stdlib.h>`**) to convert a string to a numerical value

- **`int atoi(char s[]); /* converts s to integer */`**
- **`double atof(char s[]); /* converts s to real */`**

```
// Examples of use of atoi/atof (just to understand…)
int x = atoi("2");                // x=2
double z = atof("2.35e-2");   // z=0.0235
```

NB: **`atoi/atof`** assume that the string s contains a value that can be correctly interpreted as an integer/real number, respectively. **In case of error, they return 0** to the caller. It is suggestable to check the result of the conversion

Examples with atoi/atof

```
// Example with command line arguments
// Suppose the program is executed from command line as follows:
// sum 5.4 -0.15e2
int main (int argc, char *argv[]) {
  float a, b, sum;
  if (argc != 3) {
       printf(“The execution should be %s <number1> <number2>!”,argv[0]);
       return 1;
  }
  a = atof(argv[1]);
  b = atof(argv[2]);
  sum = a+b;
  printf(”The program %s computes %f +%f = %f\n", argv[0], a, b, sum);
  return 0;
}
```

Example 1

- Write a program that receives two integers N and D **from the command line**, and prints on the screen all the numbers that are less than or equal to N and divisible by D

Example of execution (Windows)

```
C:\> myprogram 10 2
```

Output on the screen:

```
2 4 6 8 10
```

Solution

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
  int N, D, i;
  if (argc != 3) {
     printf(”Execution error: the number of arguments is not valid\n”);
     printf(”Please execute as: %s <int> <int>\n”, argv[0]);
     return 1;
  }
  N = atoi(argv[1]);
  D = atoi(argv[2]);

  for (i=1;i<=N;i++) {
    if (i%D == 0) {
      printf(”%d ”,i);
    }
  }
  return 0;
}
```

Which kind of arguments should we pass to the main?

- In theory, anything…
- In the practice, typical arguments are:
  - **Filenames**
    - Ex: the name of the input file and/or output file are passed as arguments, instead of being introduced by keyboard
  - **Options of the program,** that specify the type of operation or “modality” that we want to execute
    - These “options” (aka ***flag*** o ***switch***) are conventionally specified as –<***character***>, to distinguish them from the other arguments
    - Example

$$\texttt{C:\textbackslash> myprog }\underbrace{\texttt{-x -u}}_{\textit{options}}\ \underbrace{\texttt{file.txt}}_{\textit{additional argument}}$$

Example 2

- Write a program `m2m` that reads a text from a file and rewrites it on a second file, after converting all uppercase letters to lowercase or vice versa, depending on the flags specified on the command line:

```
-l, -L          lowercase conversion
-u, -U          uppercase conversion
```

  The flag **–h** (or **–H**) allows to print a help on the screen, with the execution instructions

- Possible ways to execute the program from command line:

```
m2m –l input.txt output.txt
m2m –L input.txt output.txt
```

The program reads input.txt and copies the text into output.txt, after converting it to lowercase

```
m2m –u input.txt output.txt
m2m –U input.txt output.txt
```

The program reads input.txt and copies the text into output.txt, after converting it to uppercase

```
m2m -h
m2m -H
```

The program prints a help on the screen

Solution

```
 #include <stdio.h>
void convertToUpper(char file1[], char file2[]); // function that reads text from file1 and copies it to file2, coverted to uppercase
void convertToUpper(char file1[], char file2[]); // function that reads text from file1 and copies it to file2, coverted to lowercase
int main(int argc, char *argv[]) {
    … …
    switch (argv[1][1]) {   // the second character (position 1) of the string argv[1] is the switch selector
       case ‘l’: case ‘L’:
       convertToLower(argv[2],argv[3]);
       break;
     case ‘u’: case ‘U’:
       convertToUpper(argv[2],argv[3]);
       break;
     case ‘h’: case ‘H’:
       printf(“Usage: m2m -[lLuU] <namefile_input> <namefile_output>\n m2m –[hH] for help\n”);
       break;
     default:
       printf(“Usage error! m2m –[hH] for help\n”);
     }
    return 0;
}
… // you can implement the functions convertToUpper and convertToLower on your own
```