University of Zagreb

Faculty of Electrical Engineering and Computing

Department of Applied Computing

**Instructions for using the character interface in Windows**

Zagreb, 2023.

Table of Contents

[1 Introduction 1](#_Toc52278100)

[2 Launching the character interface 1](#_Toc52278101)

[3 Execution of programs and commands 2](#_Toc52278102)

[4 Work with files and folders 3](#_Toc52278103)

[4.1 Windows file system 3](#_Toc52278104)

[4.2 View the contents of the folder (*dir*) 6](#_Toc52278105)

[4.3 Change working folder (*chdir, cd*) 6](#_Toc52278106)

[4.4 Creating a folder (*mkdir, md*) 7](#_Toc52278107)

[4.5 Copy files (*copy*) 7](#_Toc52278108)

[4.6 Promjena imena i premještanje datoteka i mape (*move*) 7](#_Toc52278109)

[4.7 Print the contents of the file to the screen (*type*) 7](#_Toc52278110)

[4.8 Delete the file (*del*) 7](#_Toc52278111)

[4.9 Delete a folder (*rmdir*) 8](#_Toc52278112)

[5 Redirect 8](#_Toc52278113)

[5.1 Redirect output to a file 8](#_Toc52278114)

[5.2 Redirect input data from a file 8](#_Toc52278115)

[5.3 Redirection of input and output data 9](#_Toc52278116)

[6 Text editors 9](#_Toc52278117)

[7 Exercise tasks 10](#_Toc52278118)

# Introduction

The Windows operating system, for communication with the user*,* in addition to the *graphical user* interface (GUI), allows the use of the text (character-based) userinterface (CLI - *command-line interface, command-line user interface, console user interface,* CUI*-character user interface*) in which the computer is controlled by typing a *command* in the *command line*. In this way of communication*,* the computer first prints a message (*prompt*) to the userthat it is ready to accept the command*,* after which the userenters the commandusing the keyboard. The character*-*based userinterface is very commonly used in science*,* engineering*,* and industry*,* and is supported by all of today's important operating systems*,* such as Unix*,* Linux*,* Windows*,* etc.

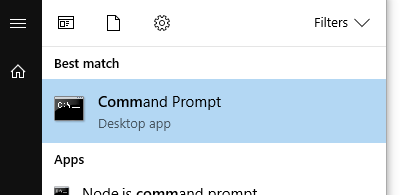
Within the course *Introduction to Programming*, the character interface will be used when working with files in which the source and executable codes of the C program are stored, and for translating and executing the C program.

# Launching the character interface

There are several options for launching a character interface (i.e., opening a character interface window). Here are some options:

Windows 10:

1. Start  ® Type *Command* ® the menu appears ® select *Command Prompt*:

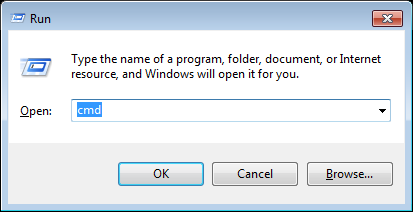


Older versions of Windows:

**a) Start**Slika gumba Start®  ***All Programs*** ®  ***Accessories***®  ***Command Prompt***

b) **Start**Slika gumba Start®  ***All Programs*** ®  ***Accessories*** ®  ***Run***

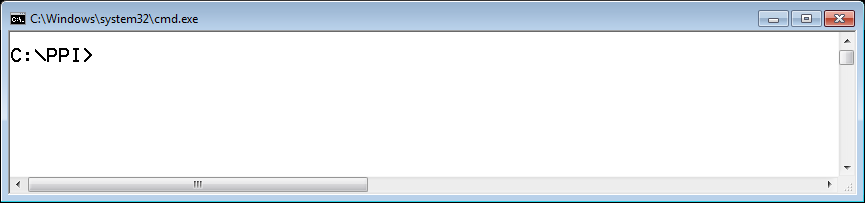
after which the window displayed below opens in which it is necessary to enter the **cmd** command in the box and press the **OK** button**:**



c) **Start**Slika gumba Start® type **cmd** in the search box, after which the window shown under b) will open

d) simultaneous pressing of the **Windows** and **R** keys (**Win + R**), after which the window shown under b) opens

In all versions of Windows, the command prompt window then opens:



# Execution of programs and commands

Starting a command or program is done by typing its name, arguments (e.g. file name) and options (parameters that modify the operation of the command, and are listed after the / sign), and by pressing the **Enter** key. Uppercase and lowercase letters are not distinguished in file names, commands and options. So, it doesn't matter if **HELP** or **help** is entered, for example.

The command is entered after the prompt, which appears when the interface is ready to accept the command from the user. The prompt usually ends with a character **>**.

The following are examples of using individual commands in their basic form. A list of common commands can be obtained by typing the **help** command. To display additional information about each command, it is necessary to enter **help *command\_name*.**

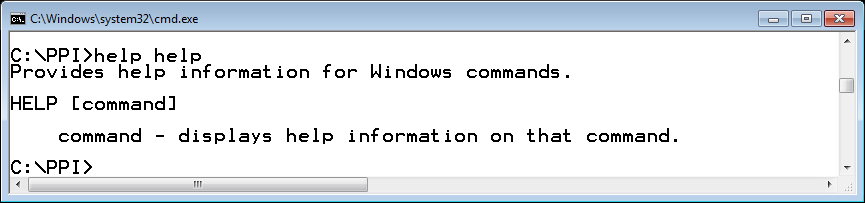
The running program can be interrupted by pressing the **Ctrl** and **C** (**Ctrl+C**) keys at the same time.

Program execution can be paused by pressing the key **Pause** or a key combination **Ctrl+S**. To resume running a program stopped in this way, press any key, for example **Enter**.

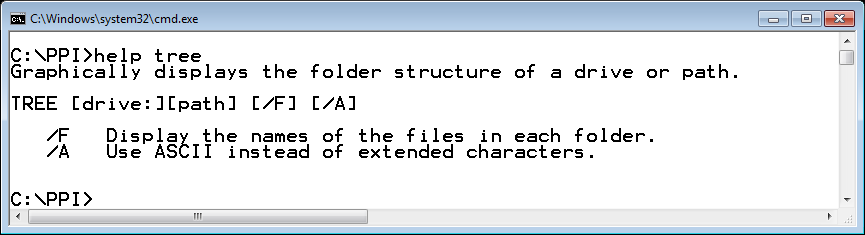
The **exit** command terminates the work with the character interface and closes the corresponding window.

Example:

**help help** (obtaining instructions on the operation of the program **help**)



**help tree** (obtaining instructions on the operation of the **tree** program; in addition to a brief description of the command, a description of the allowed options is given: **/F** and **/A,** so it is possible, for example, to issue the command: **tree /f**)



# Work with files and folders

## Windows file system

Data stored in external (permanent) memory(eg disk, CD, memory stick, ...) are logically organized into files. A file is a named set of data that makes up a logical whole. Files are logically grouped into folders (directories, indexes), which can contain any number of files and other folders (subfolders). The folders are organized hierarchically, in a tree-like structure. A folder is actually a file that contains a list and information about the characteristics of other files and/or folders.

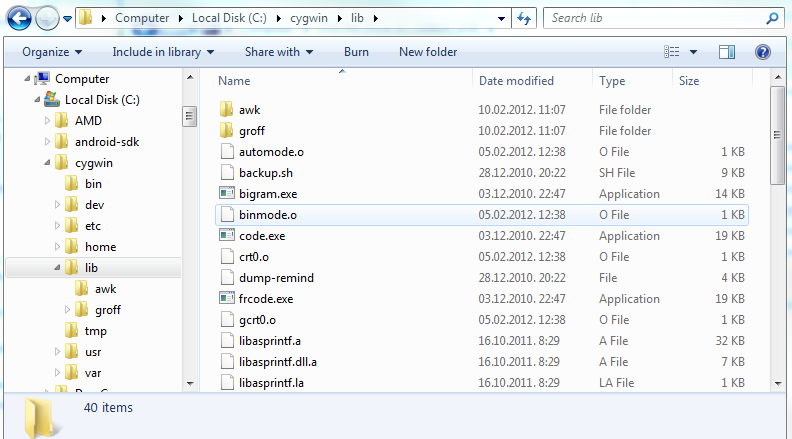
A computer file system is a part of the operating system that serves to organize, manage, and access files. File management is one of the basic tasks of the operating system. The operating system, among other things, keeps track of the physical location of files on storage media and offers an interface for opening existing, creating new files, reading from a file, writing to a file, moving, renaming, and deleting files and folders.

A file is specified (identified) by name and *absolute path* to the folder in which it is entered. The file/folder name consists of a main part and a suffix, or extension, separated by a period. The extension is not a mandatory part of the file/folder name, but is usually listed next to the file name to suggest its type and purpose. For example, it is common to assign a *txt* extension to a text-only file, a *c* extension to a file that contains source code written in the C programming language, and an *exe* extension to a file that contains an executable program. In addition to the name, the file has other properties such as size, creation time, change time, etc.

A file system can include multiple external storage units (usually hard drives). In Windows, they can be divided into several logical units (partitions), usually denoted by the letters *C :, D :,* etc. In the following text, the term *disk* refers to a logical unit.

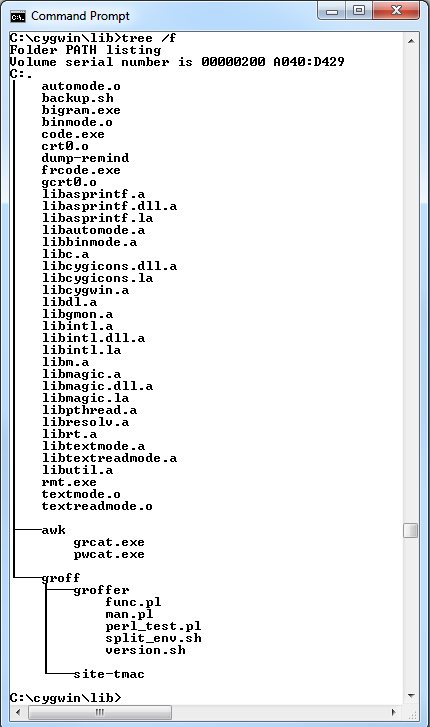
Each disk contains one *root* folder marked with **\**. As the number of base folders depends on the number of disks, in order to know which base folder it is, the disk label is indicated in front of the character *\*, eg *C*:, *D*:, etc. In some operating systems (eg Unix), there is only one base folder, so no disk label is specified.

The figure shows the tree structure of the folders on disk **C** and, on the right, the contents (subfolders and files) of the folder **C:\cygwin\lib**, in the graphical interface of the Windows operating system (using Windows Explorer):



The **tree** command allows the folder structure to be displayed in the character interface. If the command is given without options, only subfolder names are printed. By specifying the **/ f** option, files will also be displayed (picture to the right).

The operating system monitors the user's currentposition within the file system while using the character user interface. This currentposition is called the *work/current* disk and the *work/current* folder. Working disk and working folder information is an integral part of the prompt. In the image to the right, the prompt represents the text **C:\cygwin\lib >**, which means that the working disk is **C**, and the workingfolder is **lib**, which is a subfolder of the **cygwin** folder located in the main folder.



The file/folder can be accessed by specifying the path to that file/folder, i.e. by specifying the name of the folders you need to go through to get to that file/folder. Subfolders are separated by the character **\**. The path can be absolute or relative.

The absolute path starts with the disk label and the base folder, after which all folders are listed up to the folder where the file is located**.** For example, if the file **list.txt** is in a folder **\users\ana** on disk C, the absolute path to it is C:\users\ana\list.txt.

*Relative path* depends on the working folder. You can also use special tags when specifying the relative path to the file:

**.** indicates the working folder

**..** indicates a folder superior to the working folder

For example, if a working folder is C:\users\hrvoje (the absolute path to that map), mark **.** represents the mark for the folder C:\users\hrvoje, mark **..** represents the mark for the folder C:\users, and ***..\..*** is the mark for the root folder C:\. Relative to the above folder, the relative path to the file whose name it is **list.txt**, and is located in the folder C:\users\ana is **..*\*ana*\*popis.txt**.

Most commands accept the name of one or more files or folders as an argument. If the name of a folder or disk is omitted when specifying the name, it is assumed that it is a working folder and a disk.

In some commands it is not required to write the full name of the file or folder, but it is also possible to use special characters (so-called metacharacters, *wildcards*):

\* replaces an undetermined string, any combination of characters

? replaces only one, any character.

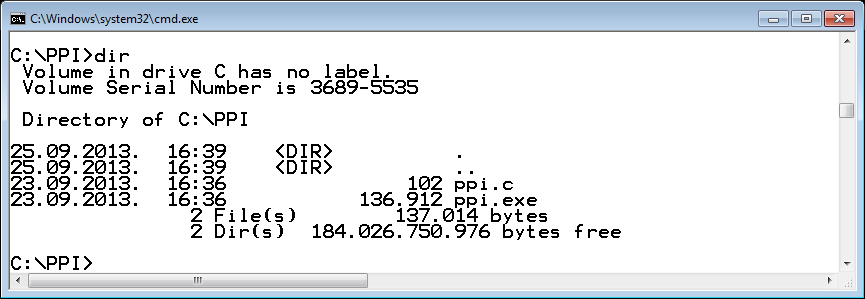
For example, the string ?.txt specifies all files that have a one-character name and extension txt.

## View the contents of the folder (*dir*)

The **dir** command prints the contents of the folder. Names that are labeled in the printout with <DIR> are folders and the rest are files.

Example:

dir (view all files and folders in the working folder)



dir ipro (view the *ipro* folder located in the working folder)

dir c:\tmp (view the *tmp* folder on disk C :, specifying the absolute path)

dir \tmp (view the *tmp* folder on the working disk, specifying the absolute path)

dir c:\t\*.\* (view all files whose name begins with the letter *t* and have some extension)

dir ?.txt (view all files whose name has one letter and have a suffix *txt*)

## Change working folder (*chdir, cd*)

The **cd** (*change directory*) command is used to changethe working folder. By executing the **cd** command, the specified folder is declared a working folder. The folder whose name is listed with the command must exist. If the folder name after the **cd** command is omitted, the command prints the absolute path to the working folder. It is allowed to use the absolute and relative path to the folder.

**REMARK:**

The **cd** command cannot change the working disk. Changing the working disk is done by specifying the name of the disk and the colon.

Example:

**cd \**users (the \*users* folder on the working disk becomes the working folder)

cd .. (the folder parent to the working folder becomes the working folder)

cd ..**\ana** (*ana* folder, which is located in the parent of the working folder, becomes the working folder)

cd users (the *users* folder, located in the working folder, becomes the working folder)

**cd** (print the absolute path to the working folder)

D: (disk *D* becomes the working disk)

## Creating a folder (*mkdir, md*)

The **mkdir** command creates a new folder, the name of which must be specified. By creating a folder, that folder does not become the working folder. To make this folder the working folder, you need to use the **cd** command.

Example:

**mkdir c:\tmp** (creating a *tmp* folder in the root folder on disk *C*)

**mkdir tmp** (creating a *tmp* folder in the working folder)

## Copy files (*copy*)

The **copy** command copies a file (or files) to another file within the same or another folder. The **copy** command in a simpler form has two arguments: the first argument represents the name of the file whose contents are being copied (source), and the second argument is the name of the copy. In another form, multiple files can be passed to the copycommand as a source, and the last argument is the name of the folder to which all the listed files are copied.

Example:

copy ipro.c ipro1.c (copy the *ipro.*c file to the *ipro1.*c file in the working folder)

copy ipro.c \tmp (copy the *ipro.c* file from the working folder to the \*tmp* folder on the working disk)

copy C:\\*.\* . (copy all files from the root folder to the working folder)

## Promjena imena i premještanje datoteka i mape (*move*)

The **move** command changes the name of one file or folder, or moves a folder or file to another folder. If multiple file names are specified as the source or special characters are used, the name of the folder (similar to the **copy** command) to which the source files will be moved should be specified for the target name.

Example:

**move ana hrvoje** (changing the name of the folder *ana* to *hrvoje*)

**move ipro.c ipro.save** (rename file *ipro*.c to *ipro*.*save*)

move ipro\* tmp (move files whose name starts with *ipro* and have any extension to the *tmp* folder)

## Print the contents of the file to the screen (*type*)

The **type** command displays the contents of a file (usually files with the *txt* extension) on the screen. Attempting to print the contents of a file that contains more than just text (such as files with an *exe* extension) results in an incomprehensible string.

Example:

type ipro.c (printing the contents of the *ipro*.*c* file to the screen)

## Delete the file (*del*)

The **del** command can delete one or more files (the use of special characters \* and ? Is allowed). If the folder name is specified as a command argument, all files in that folder will be deleted.

Example:

del ipro.exe (delete the *ipro.exe* file)

del x.\* (delete all files named *x* and with any extension in the working folder)

del \tmp (delete all files in the *tmp* folder)

del . (delete all files in the working folder)

## Delete a folder (*rmdir*)

The **rmdir** command deletes the specified folder. If no additional options are specified, the folder can only be deleted if it is empty, i.e. if there are no files or folders in it. Using the **/s** option, it is also possible to delete a folder that is not empty.

Example:

rmdir C:\tmp (delete an empty *tmp* folder in the source folder)

rmdir /s exercise (deleting the *exercise* folder located in the working folder, the folder will be deleted even if it is not empty)

# Redirect

When executing commands in the character interface, it is assumed that the input data is accepted from the keyboard and the output is printed in a window. The keyboard represents the standard input and the window the standard output. However, sometimes it is necessary to redirect the output data to a file, or to download the input data from the file instead of the keyboard. This is accomplished by using redirect characters:

> redirects the output to a file; if the file already exists, its previous contents are deleted

< retrieves input data from a file

>> redirects the output to a file; if the file already exists, the data is added to the end of the file

## Redirect output to a file

By command

dir C:\tmp > dir\_tmp.txt

the result of the ***dir C:\tmp*** command, which would appear on the screen by default, will be written to the ***dir****\_****tmp****.****txt***file in the working folder*.* If the file did not exist before the command was executed, it will be created by the command*.* If the file existed, its old contents will be lost.

By command

dir . >> dir\_tmp.txt

the result of the command dir . will be added to the end of the file *dir\_tmp.txt*.

## Redirect input data from a file

Redirecting input data is especially useful when testing a program that requires a lot of input data, because instead of constantly typing it using the keyboard, it is possible to use data that has been pre-prepared and stored in a file.

By command

calculate.exe < date.txt

the standard input is redirected from the *data.txt* file, so the program *calculate.exe* will load the necessary datafrom that file instead of from the keyboard*.* If the program prints the results, they will be printed on the computer screen.

## Redirection of input and output data

Command

calculate.exe < data.txt > results.txt

the input to the *calculate.exe* program is redirected from the *data.txt* file, and the output is redirected to the *results.*txt file.

# Text editors

Text editors are programs intended for writing text. There are two groups of text editors: basic text editors that do not have special formatting characters (so-called control codes) and advanced text editors that, in addition to letters and common characters, insert special characters that are used to format the final text (margin, type and font sizes, etc.). Only basic text editors are used to write the program.

The following editors (in addition to Visual Studio Code) are allowed in lab: **Notepad** and **Notepad++.**

Notepad can be run in two ways:

* by specifying only the command **notepad**,
* by specifying the command **notepad** followed by the name of the file to be edited (**notepad** *file\_name*).

The same applies to the Notepad ++ editor, but assuming that the folder where Notepad ++ is installed is added to the system variable path (like for *gcc*, see *Instructions For Writing Programs*):

* by specifying only the command **notepad++**,
* by specifying the command **notepad++** followed by the name of the file to be edited (**notepad++** *file\_name*).

If the editor is started without a file name, the file name in which you want to save the typed text will be specified later*.* However, as *txt* is the default extension for the file name, it often happens that the file gets an additional *txt* extension, and care should be taken with the folder in which it will be located*.* For example, if a file needs to be named *ipro.c*, it is possible that it should actually be named *ipro.c.txt.* Therefore, it is recommended to start the editor by specifying the file name (with the possibility of specifying the absolute or relative path):

**notepad** **ipro.c** (the editor opens the *ipro.c* file located in the working folder)

**notepad** d:\exercises\**p.c** (the editor opens the *p.c* file located in the folder \exercises on disk *D*)

# Exercise tasks

1. Do the following:
2. Declare the basic folder on disk *C* a working folder (in the continuation of the exercise, change the working disk and the folder only when requested in the task).
3. Create an *exercise* folder on disk *C* in the foot folder.
4. Within the *exercise* folder, without changing the working folder, create *personal* and *ipro* folders.
5. Declare the folder *C:\exercise\personal* theworking folder using the relative path, and create *txt* and *pictures* folders within it
6. Using the absolute path, declare the *pictures* folder a working folder, and then copy several picturesto that folder (you can search for them first using, for example, Windows Explorer). When copying images, state the absolute path to the folder where the picturesare located, and the relative path to the folder to which you want to copy *pictures*.
7. Declare the folder *C:\exercise\personal\txt* working folder, using a relative path and a single command.
8. Check if the file *not.here* exists in the working folder.
9. Use the text editor to enter some text in the *life.txt* file.
10. Copy the *life.txt* file to the *tmp.txt* and *tmp1.txt* files in the working folder.
11. Print the tree structure of the folder (no files) on disk *C* starting from the base folder. Pause and then resume printing that structure on the screen.
12. Print the tree structure of the folder, including the files, starting with the *exercise* folder using a relative path. Compare the resulting structure with the structure displayed in the graphical interface, using Windows Explorer.
13. Call the ***help*** program without arguments, and redirect the program output to the *help.txt* file.
14. Print the contents of the working folder.
15. Move the *help.txt* file to the parent folder.
16. By printing the contents of the working folder and then the parent folder, check that the copying was successful.
17. Without changing the working folder, print the contents of the *help.txt* file to the screen.
18. In the working folder, delete all files whose name begins with the letter *p*.
19. Declare the root folder as the working folder by specifying the relative path.
20. Delete folder *C:\exercise\personal\txt* (specify absolute path to folder).
21. Delete all files in the folder C: \exercise\personal\pictures (specify the relative path to the folder).
22. Delete the folder *personal*.
23. Write a program that will read two integers and print a larger number*.* Store the source code in the *test.*c file in the *ipro* folder created in Task 1*.* Use the Notepad or Notepad ++ editor to edit the program (specify the file name when starting the editor). Translate the program using the GCC compiler, and name the executable file *test.exe.* Run the program so that:
24. the numbers are read from the keyboard and the result is printed on the monitor screen
25. the numbers are read from the keyboard and the result is printed to the *output.txt* file
26. numbers are read from the *input.txt* file previously prepared using a text editor, and the result is printed to the end of the *output.txt* file, without deleting its previous contents

**Solutions**

1. cd c:\
2. mkdir exercise
3. mkdir exercise\personal

mkdir exercise\ipro

1. cd exercise\personal

mkdir txt

mkdir pictures

1. cd c:\exercise\personal\pictures

npr: copy c:\users\hrvoje\pictures\\*2013\*.jpg .

1. cd ..\txt
2. dir not.here
3. notepad life.txt (or notepad++ life.txt)
4. copy life.txt tmp.txt

copy life.txt tmp1.txt

1. tree \

temporary suspension of execution: Pause (or Ctrl+S)

resume execution: any key

1. tree /f ..\..
2. help > help.txt
3. dir
4. move help.txt ..
5. dir .

dir ..

1. type ..\help.txt
2. del p\*
3. cd ..\..\..
4. rmdir /s *C:\exercise\personal\txt*
5. del *C:\exercise\personal\pictures*
6. rmdir /s *C:\exercise\personal*
7. cd c:\exercise\ipro

notepad test.c (or notepad++ test.c)

Enter the following code in the *test.c* text file:

#include <stdio.h>

int main(void) {

int x,y;

printf("Enter two integers: ");

scanf("%d %d", &x, &y);

if (x>y) {

printf("%d\n", x);

} else {

printf("%d\n", y);

}

return 0;

}

gcc -std=c11 -pedantic-errors -Wall -o test.exe test.c

1. test.exe
2. test.exe > output.txt

type output.txt

1. notepad input.txt (or notepad++ input.txt)

Enter two integers in the character file *input.txt*, e*.*g.:

10 5

test.exe < input.txt >> output.txt

type output.txt