

```
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
```

```
#define MAXPAROLA 30
#define MAXRIGA 80
```

```
int main(int argc, char *argv[])
{
    int freq[MAXPAROLA]; /* vettore di contatori
delle frequenze delle lunghezze delle parole */
    char riga[MAXRIGA];
    int i, inizio, lunghezza;
    FILE *f;
```

```
for(i=0; i<MAXPAROLA; i++)
    freq[i]=0;
```

```
if(argc != 2)
```

```
{
    printf(stderr, "ERRORE, serve un parametro con il nome del file\n");
    exit(1);
}
```

```
f = fopen(argv[1], "r");
if(f==NULL)
```

```
{
    printf(stderr, "ERRORE, impossibile aprire il file %s\n", argv[1]);
    exit(1);
}
```

```
while( fgets( riga, MAXRIGA, f ) != NULL )
```



# System and Device Programming

## Review Session

Stefano Quer

Department of Control and Computer Engineering

Politecnico di Torino

## License Information

This work is licensed under the license



### Attribution-NonCommercial-NoDerivatives 4.0 International

This license requires that reusers give credit to the creator. It allows reusers to copy and distribute the material in any medium or format in unadapted form and for noncommercial purposes only.

① **BY:** Credit must be given to you, the creator.

② **NC:** Only noncommercial use of your work is permitted.

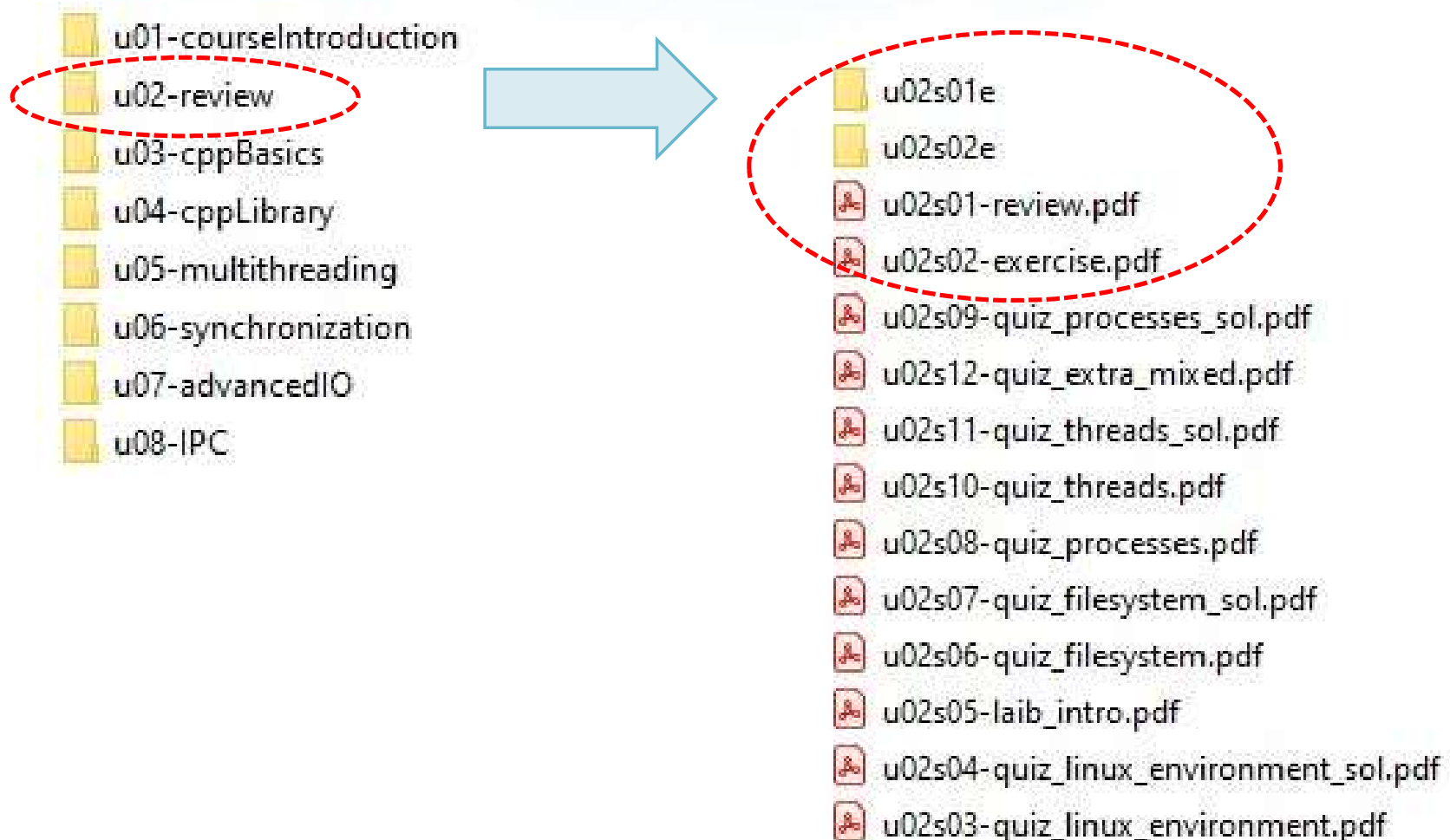
Noncommercial means not primarily intended for or directed towards commercial advantage or monetary compensation.

③ **ND:** No derivatives or adaptations of your work are permitted.

To view a copy of the license, visit:

<https://creativecommons.org/licenses/by-nc-nd/4.0/?ref=chooser-v1>

# Overview



# Standards

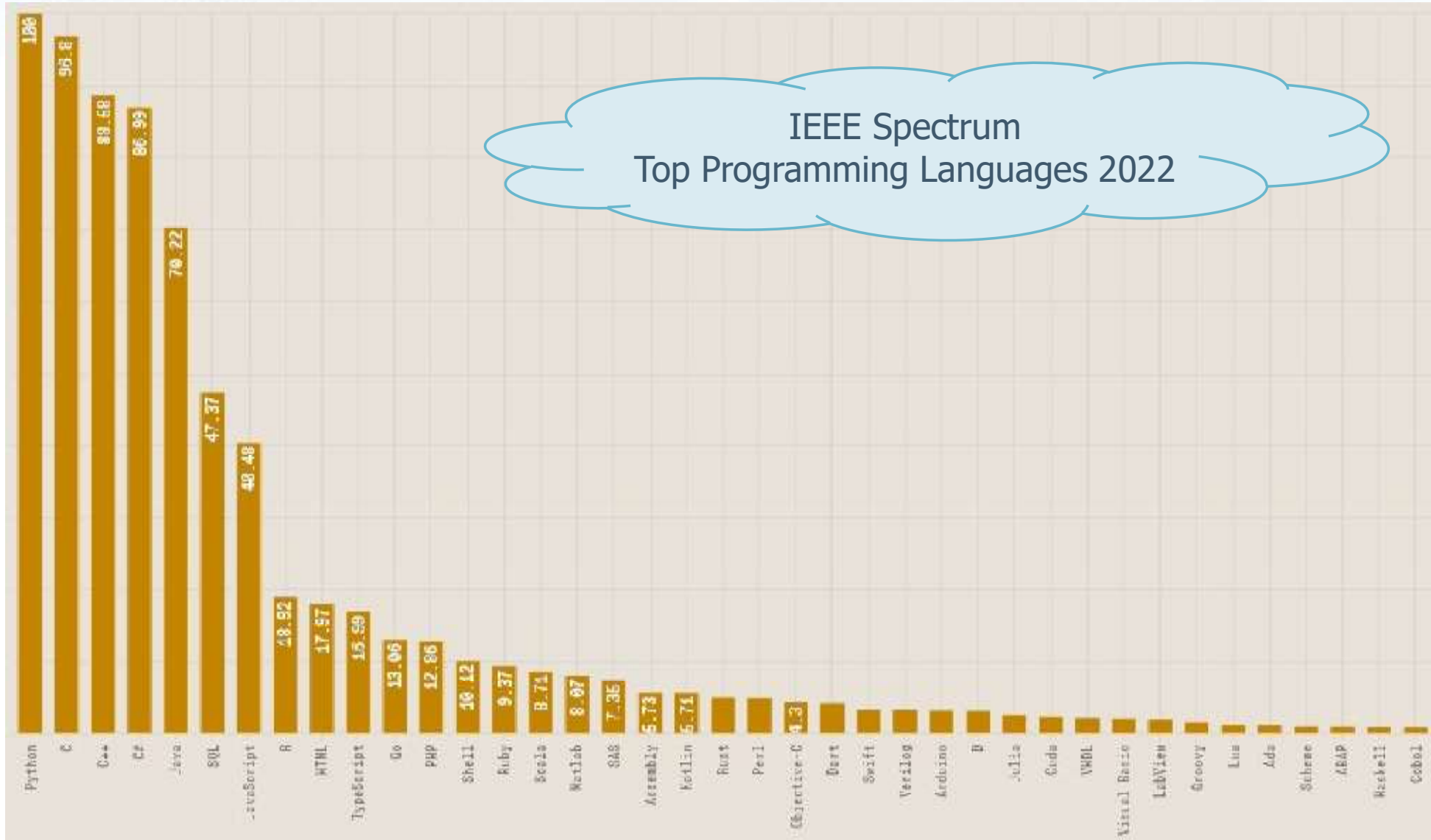
## ❖ Standards studied in this course

For C and POSIX, refer to the BS course  
**Operating Sytems** (English) and **Sistemi Operativi** (Italian)

Standard	Comments
C	1972: UNIX migrates form assembler to C language. Standard C language versions : ANSI C (1989), ISO C or C90 (1990), C94 (1994), C99 (1999), C11 (2011), C17 (2018), C23 (2023)
POSIX	POSIX = Portable Operating System Interface Family of standards proposed to promote UNIX systems portability. Defines the services a UNIX system must satisfy to be "POSIX compliant". Includes the ISO C standard.
C++	High-level general-purpose programming language. Created by Bjarne Stroustrup in 1985. Expanded significantly over time. Last ISO version in 2020.

# Programming languages

IEEE Spectrum  
Top Programming Languages 2022



## Review: Problem solving

- ❖ Problem solving in C language
  - C language basics
  - Dynamic memory allocation
  - Data structure and Abstract Data Types (ADT)
  - Main algorithms
    - Sorting
    - Searching
    - Merging
    - Symbol tables (tree, hash, etc)
    - Heaps
    - Graphs

Refer to BS course of  
**Algorithms and Data  
Structures**



# Review: Operating Systems

## ❖ Operating system (specifically, UNIX-like) theoretical aspects

### ➤ Filesystem

- Basic structure, IO

### ➤ Processes

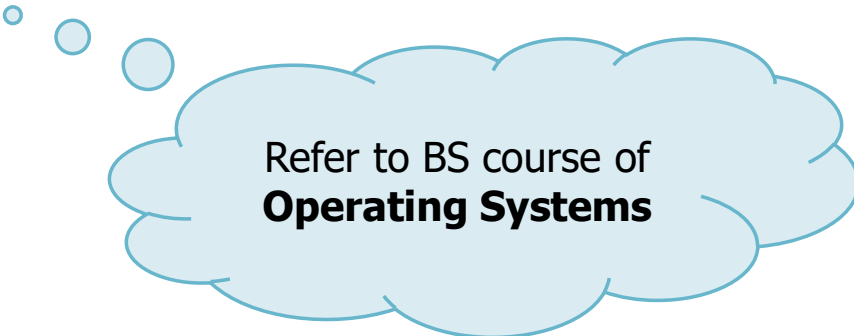
- Signals, pipes

### ➤ Threads

- Multithreading

### ➤ Synchronization

- Semaphores, mutexes



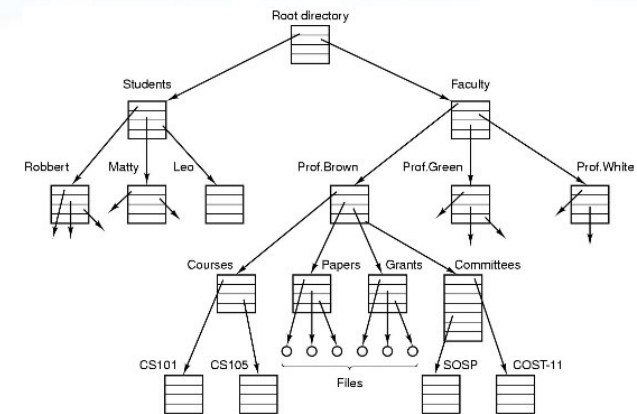
Refer to BS course of  
**Operating Systems**

# Review: Operating Systems

## ❖ The filesystem

### ➤ Basics filesystem notions

- Files and directories



### ➤ UNIX I/O

UNIX I/O system call	Action
open	To open a file (UNIX primitives for fopen)
read	Read a file (ASCII or BINAY format)
write	Write a file (ASCII or BINAY format)
lseek	Random access to a file
close	To close a file (UNIX primitives for fclose)



# Review: Operating Systems

## ❖ Processes and process manipulation

Process System call	Action
getpid, getppid, etc.	Process identification
fork	Process creation (cloning)
wait	Manage child termination
waitpid	Manage a specific child termination (with non blocking features)
exec (execl and execv)	Process substitution
system	To call a shell command
signal	Install a signal handler
kill	Send a signal
pause, sleep, alarm	Put a process in pause, into a sleep state, set an alarm (countdown)
pipe (read & write)	Create a pipe and use it for Inter-Process Communication (IPC) between two processes

# Review: Operating Systems

## ❖ Multithreading and thread manipulation

### ➤ The POSIX standard

- Pthreads (POSIX Threads)

Revised in section 05

Multithreading system calls	Action
pthread_equal	To check the equivalence of two thread identifiers
pthread_self	To get back a thread identifier
pthread_create	To create a new thread
pthread_exit	To exit a thread
pthread_join	To wait for a thread
pthread_cancel	To cancel an existing thread
pthread_detach	To make a thread as detached (i.e., not joinable)

# Review: Operating Systems

- ❖ Typical synchronization problems
  - Producer and consumer, readers and writers, etc.
- ❖ Synchronization with semaphores



Revised in section 06

Synchronization system call	Action
sem_init	Initialize a semaphore
sem_wait	Wait on a semaphore
sem_trywait	Non-blocking function to wait on a semaphore
sem_post	Signal (unblock) a semaphore
sem_getvalue	Get the value of a semaphore back (prone to race-condition)
sem_destroy	Destroy (free) and existing semaphore

# Review: Operating Systems

## ❖ Synchronization with mutexes

- Mutex stands for **MUT**ual **EX**clusion
- Binary (and faster) semaphore



Revised in section 06

Synchronization system call	Action
pthread_mutex_init	Create and initialize a mutex (binary semaphore)
pthread_mutex_lock	Wait on a mutex
pthread_mutex_trylock	Non-blocking version to wait on a mutex
pthread_mutex_unlock	Signal (unlock) a mutex
pthread_mutex_destroy	Destroy (free) and existing mutex

# Review: Linux in practice

## ❖ Select a Linux version

26.02.2023  
www.distrowatch.com

Kernel	Ubuntu Version
3.2	2012 12.04 LTS Precise Pangolin
3.13	2014 14.04 LTS Trusty Tahr
4.4	2016 16.04 LTS Xenial Xerus
4.15	2018 18.04 LTS Bionic Beaver
5.4	2020 20.04 LTS Focal Fossa
5.15	2022 22.04 LTS Jammy Jellyfish

Last 12 months		
1	<a href="#">MX Linux</a>	2758-
2	<a href="#">EndeavourOS</a>	2374-
3	<a href="#">Mint</a>	2117-
4	<a href="#">Manjaro</a>	1490-
5	<a href="#">Pop!_OS</a>	1238-
6	<a href="#">Ubuntu</a>	1182▼
7	<a href="#">Fedora</a>	1180▲
8	<a href="#">Debian</a>	1043▲
9	<a href="#">Garuda</a>	801▼
10	<a href="#">Lite</a>	763▲
11	<a href="#">openSUSE</a>	742▲
12	<a href="#">Zorin</a>	730-
13	<a href="#">elementary</a>	626-
14	<a href="#">KDE neon</a>	610-
15	<a href="#">antiX</a>	497-
16	<a href="#">PCLinuxOS</a>	442-
17	<a href="#">Kali</a>	432-
18	<a href="#">AlmaLinux</a>	425▲
19	<a href="#">ArcoLinux</a>	401-
20	<a href="#">SparkyLinux</a>	385-
21	<a href="#">FreeBSD</a>	379▲
22	<a href="#">EasyOS</a>	376▲
23	<a href="#">Puppy</a>	371-
24	<a href="#">Slackware</a>	355▼
25	<a href="#">Solus</a>	331-
26	<a href="#">Linuxfx</a>	329-
27	<a href="#">Alpine</a>	328-
28	<a href="#">Kubuntu</a>	327-
29	<a href="#">Peppermint</a>	316-
30	<a href="#">Q4OS</a>	315-

## Review: Linux in practice

### ❖ Install a LINUX OS

- LXSS, Linux Windows Subsystem
- Hypervisor, e.g., Virtualbox



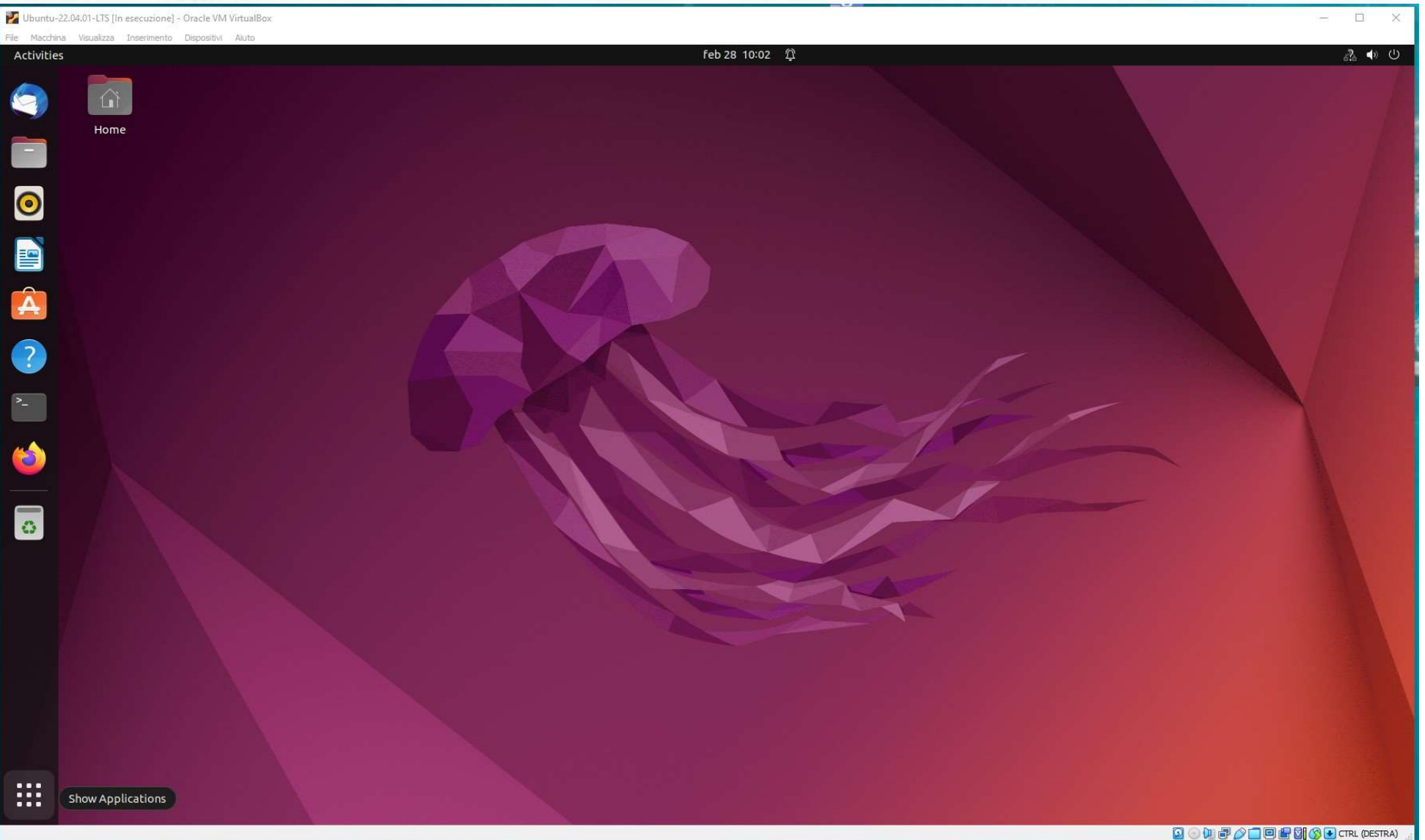
### ❖ Master Linux main tools

- Shell commands
  - For example, ls, cp, mv, ln, ...
- Programming apps
  - Editor, Linker, Compiler, Debugger, IDE

Refer to BS course of  
**Operating System**



## Ubuntu-20.04.01-LTS at work



## C++ Projects

- ❖ C++ inherits most of C's syntax
- ❖ It is almost always implemented as a compiled language
- ❖ A C++ project must be organized as a
  - Set of header files including the declarations
    - <name>.h
  - Set of C++ files including the header files and the implementations
    - <name>.cpp
  - If a strict separation is maintained, it is possible to create a **library**

# Example

Interface  
my\_class.h

To avoid symbol duplication  
Please, refer to "Algorithms and  
programming" for further details

Implementation  
my\_class.cpp

```
#ifndef MYCLASS  
#define MYCLASS
```

```
class MyClass {  
    char* ptr;  
public:  
    int doWork();  
};
```

```
#endif
```

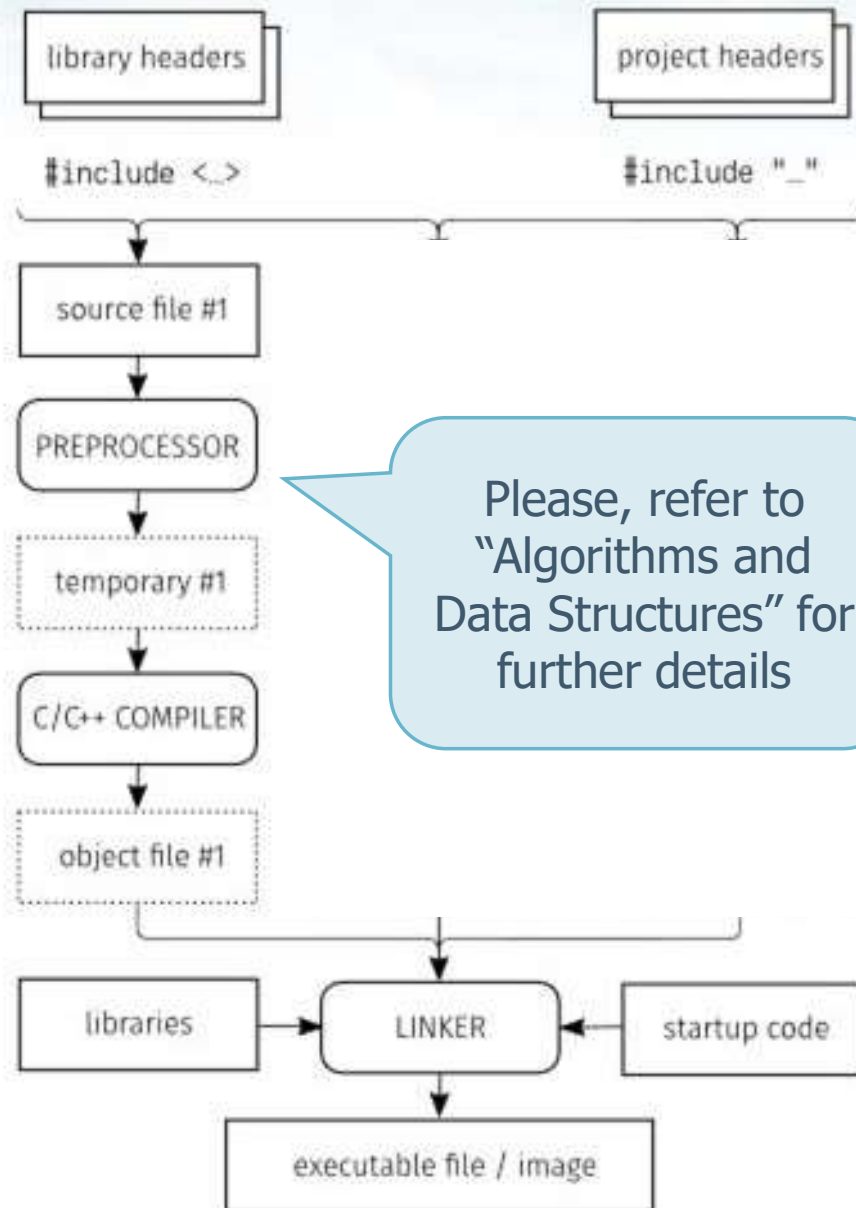
```
#include "my_class.h"
```

Include symbols

```
int MyClass::doWork() {  
    //codice  
    return 0;  
}
```

Method (doWork) of a class (MyClass):  
Name of the class (MyClass) +  
Scope identifier (::)

# C++ Projects

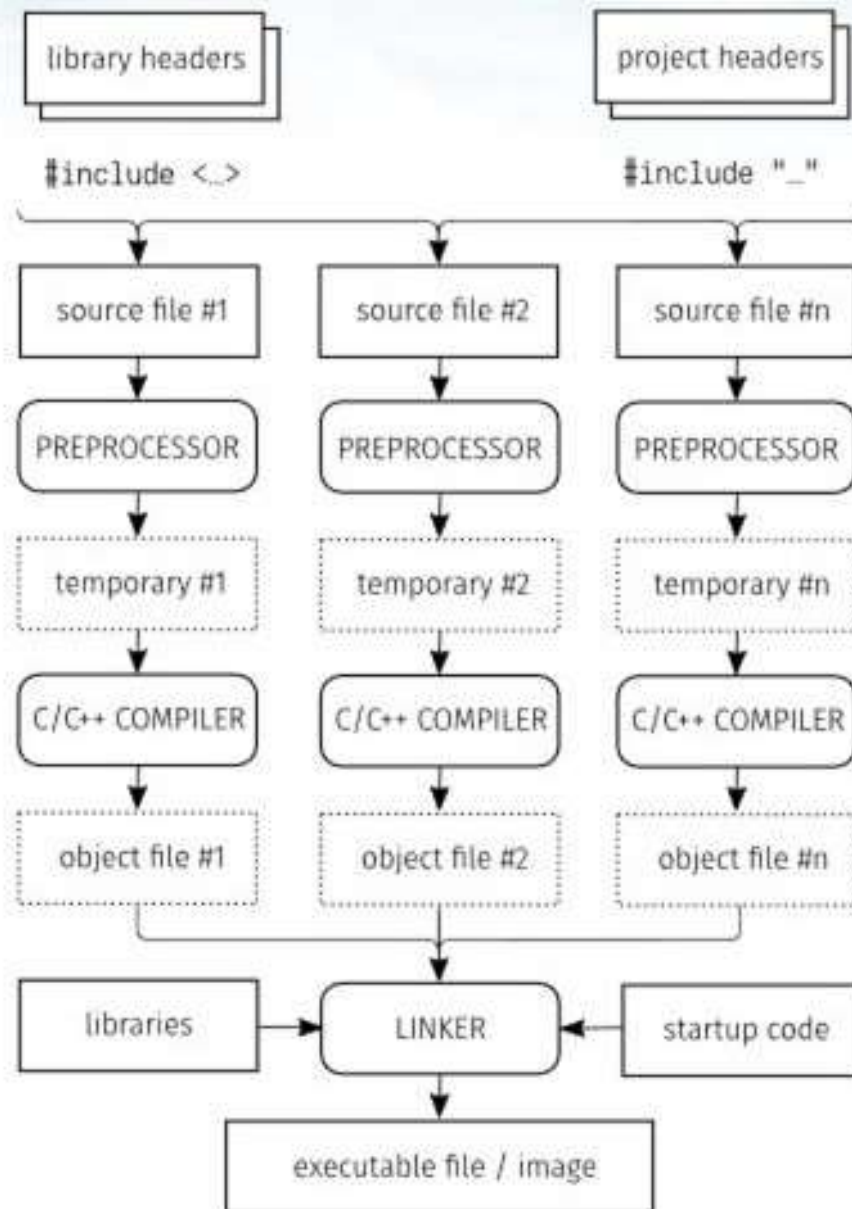


❖ Developing a program in C/C++ typically requires six phases

- Editor
- Pre-processor
- Compiler
- Linker
- Loader
- Execute

Single-file projects

# C++ Projects



❖ Developing a program in C/C++ typically requires six phases

- Editor
- Pre-processor
- Compiler
- Linker
- Loader
- Execute

Multi-file projects

## C++ Projects

- ❖ Many vendors provide C++
  - Compilers for different platforms
    - Free Software Foundation, LLVM, Microsoft, Intel, Embarcadero, Oracle, IBM XL C++,EDG, etc.
  - Debuggers
  - IDEs such as Visual Studio, C-Lion, Eclipse, etc.



# Programming in Linux

- ❖ Use separate tools to
  - Edit, compile, debug, etc.

<b>Editors</b>	<b>Comments</b>
<b>VIM</b>	VI Improved. Default editor UNIX/Linux
<b>Geany</b>	Editor for LINUX desktop
<b>Sublime</b>	Editor and development suite
<b>Brackets</b>	From ADOBE for Linux
<b>Gedit</b>	Default editor for desktops GNOME
<b>VS Code</b>	Microsoft, per Windows, UNIX/Linux, macOS.
<b>Nano</b>	As Pico but more powerful
<b>Emacs</b>	One of the oldest and pwerfull editors for Linux

# Programming in Linux

## ➤ GNU C Compiler

### ▪ Command line compilation

```
gcc -c file1.c  
gcc -c file2.c  
gcc -c main.c
```

g++  
is equivalent to  
gcc -xc++ -lstdc++ -shared-libgcc

```
gcc -o myexe file1.o file2.o main.o
```

```
gcc -Wall -g -o myexe file1.c file2.c main.c -lpthread
```

```
g++ -Wall -o myprgm -I. p1.cpp p2.cpp p3.cpp -lm
```

It is always possible  
to use a Makefile

# Programming in Linux

## ➤ GNU GDB Debugger

- Standalone or embedded use

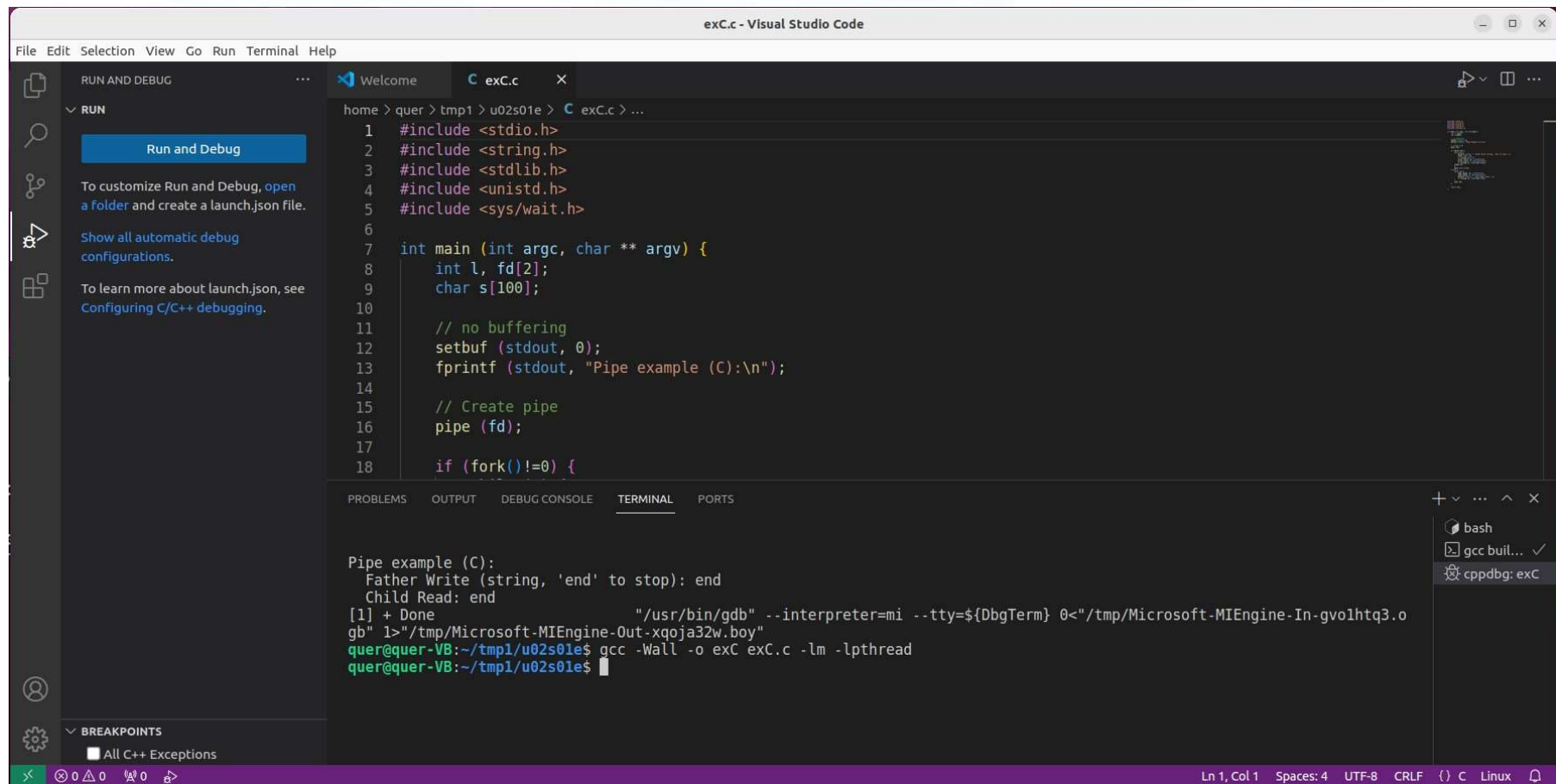
```
quer@quer-VB: ~/tmp1/u02s01e
quer@quer-VB:~/tmp1/u02s01e$ ls
exC  exCandCpp.cpp  exC.c  exCpp.cpp  Makefile
quer@quer-VB:~/tmp1/u02s01e$ make
gcc -Wall -g -o exC exC.c -lm -lpthread
g++ -Wall -g -o exCpp exCpp.cpp -lm -lpthread
g++ -Wall -g -o exCandCpp exCandCpp.cpp -lm -lpthread
quer@quer-VB:~/tmp1/u02s01e$ gdb
GNU gdb (Ubuntu 12.1-0ubuntu1-22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) file exC
Reading symbols from exC...
(gdb) l
1      #include <stdio.h>
2      #include <string.h>
3      #include <stdlib.h>
4      #include <unistd.h>
5      #include <sys/wait.h>
6
7      int main (int argc, char ** argv) {
8          int l, fd[2];
9          char s[100];
10
(gdb) b 12
Breakpoint 1 at 0x1314: file exC.c, line 12.
(gdb) r
```

## VS Code

- ❖ Visual Studio Code, commonly referred to as VS Code
  - Launch by Microsoft in 2015
  - Is a source-code editor developed for Windows, Linux, and macOS
  - Includes support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git
  - Can be used with a variety of programming languages, including C, C#, C++, Fortran, Go, Java, JavaScript, Node.js, Python, Rust, and Julia

# VS Code on Ubuntu at work



## ❖ CLion

- Distributed by JetBrains with several other IDEs
  - PyCharm (Python), GoLand (Go), RubyMine (Ruby), WebStorm (JavaScript), etc.
- Used for C/C++ (and others, e.g., CUDA)
- Cross-platform
  - Provides consistent experience on Windows, macOS, and Linux
- Adopts several GCC-based compilers
  - Clang, Clang-cl, Visual Studio C++ compiler, IAR compiler, custom-defined compiler





# Example 1

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main () {
    int l, fd[2];
    char s[100];
    setbuf (stdout, 0);
    pipe (fd);
    if (fork()!=0) {
        while (1) {
            fprintf (stdout, "Father Write: ");
            scanf ("%s", s);
            l = strlen (s) + 1;
            write (fd[1], &l, sizeof(int));
            write (fd[1], s, l*sizeof(char));
            if (strcmp (s, "end")==0) break;
            sleep (1);
        }
        wait ((int *) 0);
    } else {
```

```
        while (1) {
            read (fd[0], &l, sizeof(int));
            read (fd[0], s, l*sizeof(char));
            fprintf (stdout, "Child Read %s\n", s);
            if (strcmp (s, "end")==0) break;
        }
        exit (0);
    }
    return (0);
}
```



A C program

# Example 2

```
#include <string>
#include <vector>
#include <iostream>
```

```
using namespace std;
```

```
int main () {
    vector<string> v;
    string s2 = "one";
    v.push_back(s2);
    s2 = "two";
    v.push_back(s2);
    s2 = "three";
    v.push_back(s2);
    for (int i = 0; i < v.size(); ++i) {
        cout << "  " << v[i] << '\n';
    }
    return (0);
}
```

Based on the explanations you requested for the slides you sent, here's a quick summary of the key concepts:

Slide 1: System and Device Programming

Focuses on concepts related to programming that interact with the operating system and computer hardware.  
Slide 2: Semaphores and Threads

Semaphores: Synchronization tools that control access to shared resources between threads. They act like flags to ensure only one thread can access a resource at a time. (Think of traffic lights for threads accessing a printer)  
Threads: Units of execution within a process. They share the memory and resources of the process they belong to. (Think of workers in a factory using shared tools and materials)  
Slide 3: Mutexes

A specific type of semaphore used for mutual exclusion. Ensures only one thread can access a shared resource at a time, preventing race conditions. (Think of a lock on a bathroom stall)  
Slide 4: C++ Project Structure

Emphasizes separating code into header files (.h) and C++ source files (.cpp).  
Header files: Contain declarations for functions, classes, and variables (like blueprints).  
Source files: Contain the actual implementation of the declarations (like building the structure based on the blueprints).  
This separation promotes better code maintainability and reusability.  
Slide 5: C++ Interface and Implementation

Provides an example of how header and source files work together in C++.  
The header file declares a class with member variables and functions (interface).  
The source file defines the implementation details for the member functions (bringing the interface to life).  
Slide 6: C++ Program Development Phases

Illustrates the six phases involved in developing a C++ program:  
Preprocessor: Processes source code (includes header files, removes comments).  
Compiler: Translates preprocessed code into machine code.  
Assembler (if applicable): Converts assembly language to machine code (for specific processor).  
Linker: Combines object files and libraries into an executable file.  
Loader: Prepares the executable for execution in memory.  
Execution: The program starts running machine code instructions.  
Slide 7: Inter-Process Communication with Pipes

A C++ program

## Example 3

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

C Libraries

A C++ program  
with some C libraries

```
#include <unistd.h>
#include <sys/wait.h>
```

POSIX Libraries

```
#include <string>
#include <vector>
#include <iostream>
```

C++ Libraries

```
using namespace std;
```

```
int main () {
    ... as example 1 ...
    ... as example 2 ...
}
```

C Code

C++ Code