Introdução à Programação 2020/2021

António J. R. Neves
Daniel Corujo

Departamento de Electrónica, Telecomunicações e Informática
Universidade de Aveiro

an@ua.pt / dcorujo@ua.pt

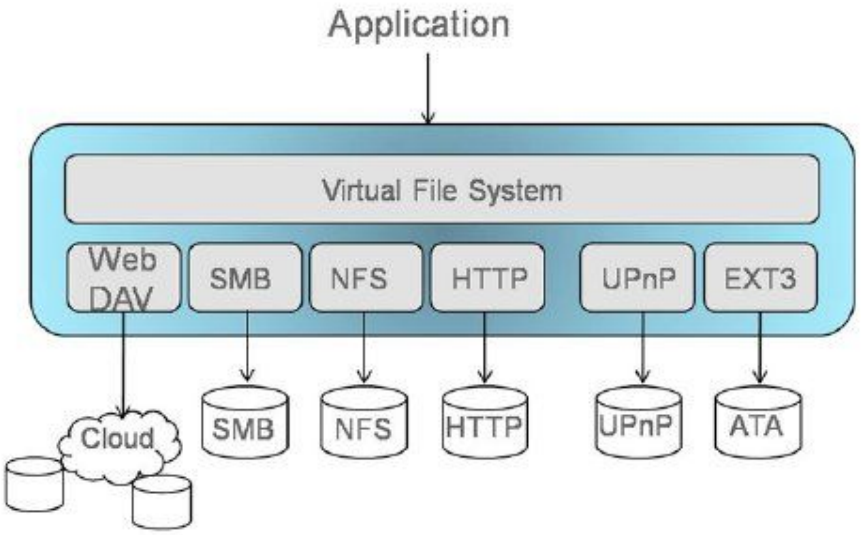
http://elearning.ua.pt/

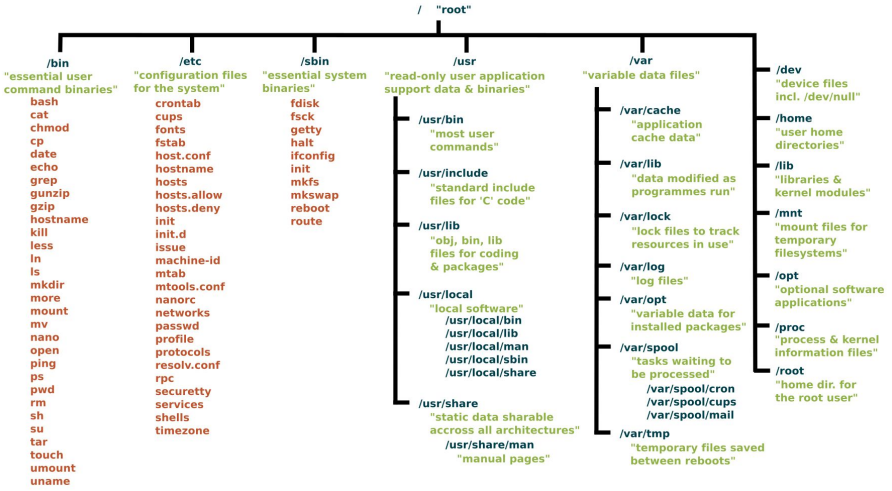
Topics

deti universidade de aveiro
departamento de eletrónica, telecomunicações e informática

- Files
- Streams
- ifstream / ofstream

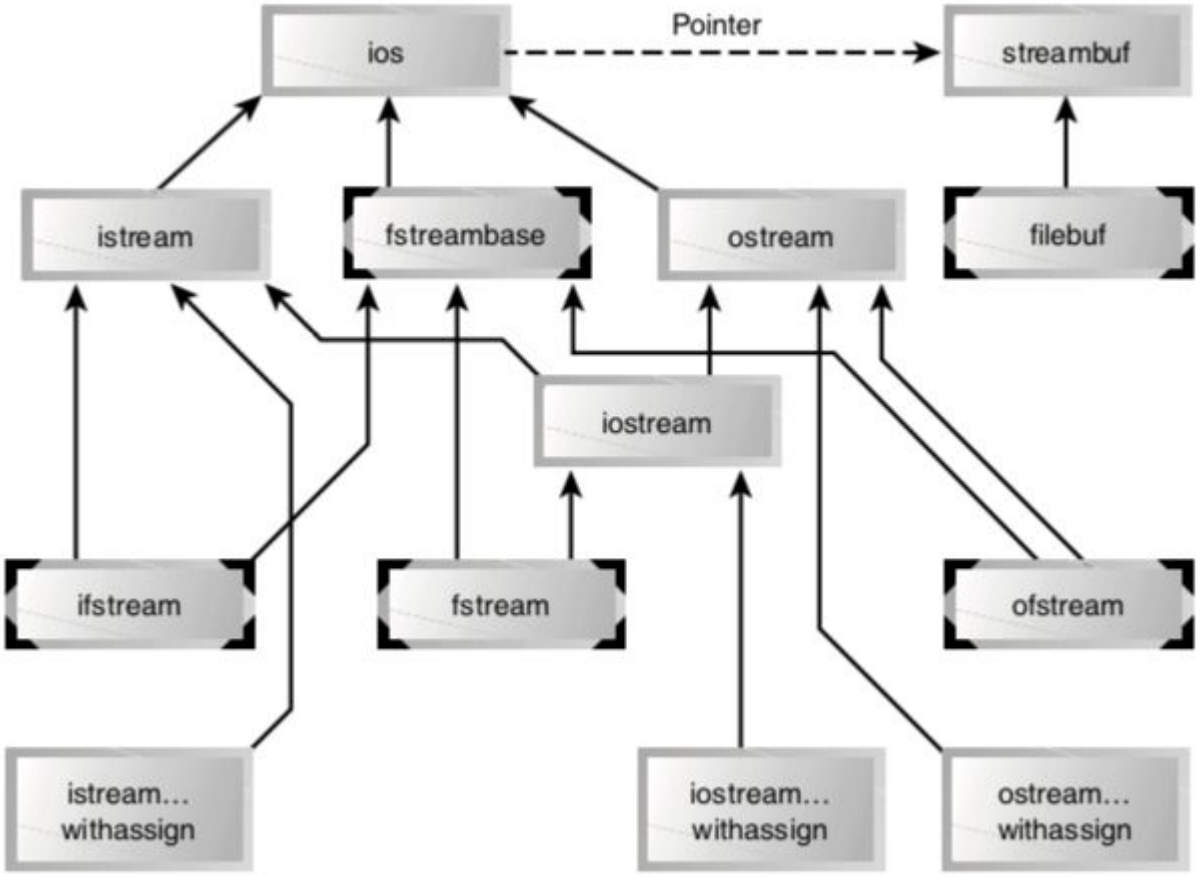
- Programs must be able to write data to files or to physical output devices such as displays or printers, and to read in data from files or input devices such as a keyboard
- From the point of view of a C/C++ program, all kinds of files and devices for input and output are uniformly represented as logical data streams (communication channels)
- Streams can be either text or binary streams





deti universidade de aveiro
departamento de eletrónica, telecomunicações e informática

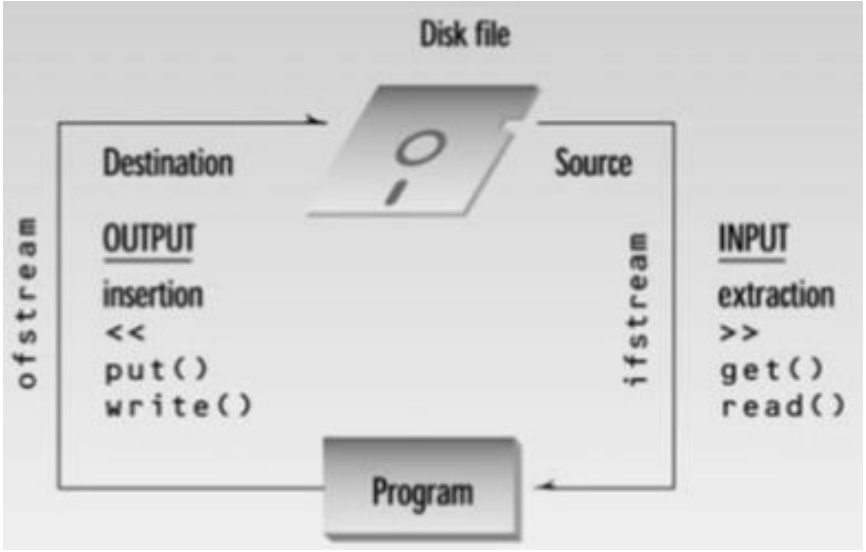
- The stream classes are arranged in a rather complex hierarchy
- The extraction operator `>>` is a member of the `istream` class, and the insertion operator `<<` is a member of the `ostream` class.



- The `cout` object, representing the standard output stream, which is usually directed to the video display, is a predefined object of the `ostream_withassign` class, which is derived from the `ostream` class
- Similarly, `cin` is an object of the `istream_withassign` class, which is derived from `istream`
- The classes used specifically for disk file I/O are related to the `fstream` class
- The `ios` class is the base class for the hierarchy - it contains many constants and member functions common to input and output operations of all kinds
- The three most important features of `ios` are the formatting flags, the error-status flags, and the file operation mode

deti universidade de aveiro
departamento de eletrónica, telecomunicações e informática

- The `istream` and `ostream` classes are derived from `ios` and are dedicated work with files (to input and output, respectively)
- We can have formatted I/O or binary I/O
  - In formatted I/O, numbers are stored on disk as a series of characters
  - In binary I/O data are stored as they are in the computer's memory
- `istream` class contains such functions as `get()`, `getline()`, `read()`, and the overloaded extraction (>>) operators
- `ostream` contains `put()` and `write()`, and the overloaded insertion (<<) operators



```
char ch = 'x';
int j = 77;
double d = 6.02;
string str1 = "Kafka";
string str2 = "Proust";
ofstream outfile("fdata.txt");
outfile << ch
        << j
        << ' '
        << d
        << str1
        << ' '
        << str2;
outfile.close();
cout << "File written\n";
```

```
char ch;
int j;
double d;
string str1;
string str2;
ifstream infile("fdata.txt");
infile >> ch
       >> j
       >> d
       >> str1
       >> str2;

cout << ch << ...
```

- The technique of our last examples won't work with strings containing embedded blanks
- To handle such strings, you need to write a specific delimiter character after each string, and use the `getline()` function, rather than the extraction operator
- In the next example each line finishes with a newline

```
ofstream ofs("names.txt");
ofs << "Ana Maria" << endl
    << "Joana Fonseca" << endl
    << "Elena Maria" << endl
    << "Maria Joana" << endl;
ofs.close();
```

```
ifstream ifs("names.txt");
while(!ifs.eof()){ //(ifs.good())
 string s;
 getline(ifs, s);
 cout << s << endl;
}
ifs.close();
```

- We need to use two new functions: `write()`, a member of `ofstream`; and `read()`, a member of `ifstream`
- These functions think about data in terms of bytes (type `char`)
- They don't care how the data is formatted, they simply transfer a buffer full of bytes from and to a disk file
- The parameters to `write()` and `read()` are the address of the data buffer and its length
- The address must be cast, using `reinterpret_cast`, to type `char*`, and the length is the length in bytes (characters), not the number of data items in the buffer
- When open the file, we must use the `ios::binary` argument in the second parameter

```
// Initialize an array with some ints
const int ARR_SIZE = 6;
int out_arr[ARR_SIZE];
for(int i = 0 ; i < ARR_SIZE ; i++)
 out_arr[i] = 10 * i;

// Write the data on file
ofstream ofsb("data.bin", ios::binary);
ofsb.write(reinterpret_cast<char*>(out_arr), ARR_SIZE * sizeof(int));
ofsb.close();

// Read the data from the file
int in_arr[ARR_SIZE];
ifstream ifsb("data.bin", ios::binary);
ifsb.read(reinterpret_cast<char *>(in_arr), ARR_SIZE * sizeof(int));
ifsb.close();

for (auto num: in_arr)
 cout << num << endl;
```

- Exceptions provide a convenient, uniform way to handle errors that occur in **runtime**
- They are caused by a wide variety of exceptional circumstance, such as running out of memory, not being able to open a file, trying to initialize an object to an impossible value, or using an out-of-bounds index to a vector
- The exception mechanism uses three new C++ keywords: `throw` (later in the 2nd semester), `catch`, and `try`

```
// Example: trying to read from a file that does not exist
try {
   std::ifstream ifs("filename");     // filename does not exist
   ifs.exceptions(std::ifstream::failbit);
   // ...
   ifs.close();
} catch (std::exception& e) {
   std::cout << "Could not read from file! Found exception: "
             << e.what() << std::endl;
}
```