Seminar 06
RAII and file streams

1. Resource Acquisition Is Initialization.

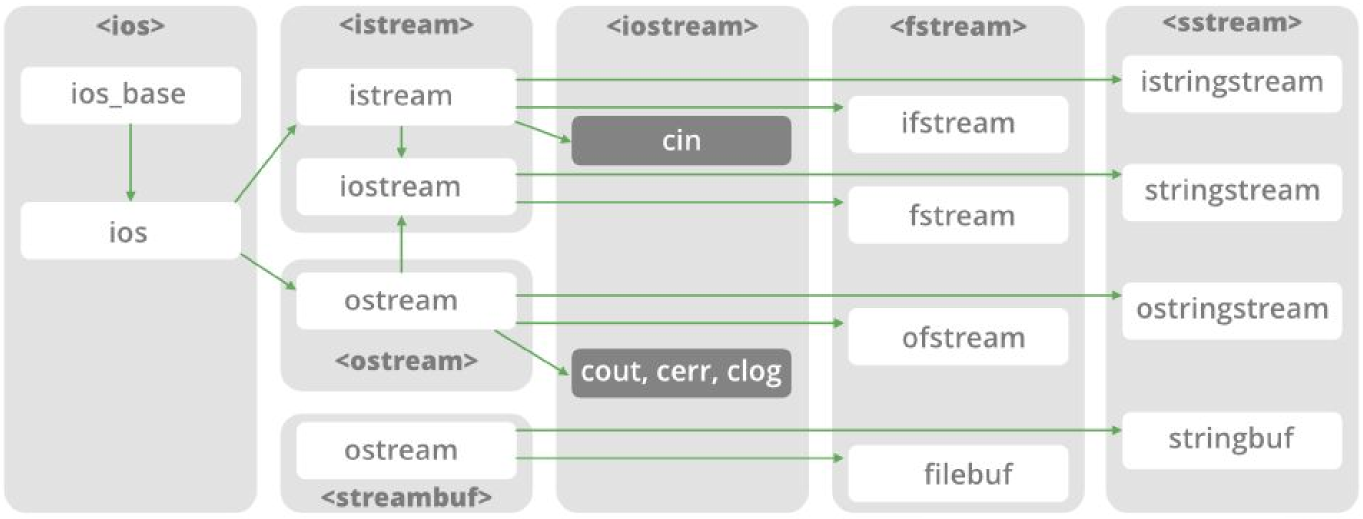
- From cppreference.com:
  *Resource Acquisition Is Initialization or **RAII**, is a C++ programming technique which binds the life cycle of a resource that must be acquired before use (allocated heap memory, thread of execution, open socket, open file, locked mutex, disk space, database connection - anything that exists in limited supply) to the lifetime of an object.*

  ***RAII** can be summarized as follows:*
  - ***encapsulate** each resource into a **class**, where*
    - *the constructor acquires the resource and establishes all class invariants or throws an exception if that cannot be done,*
    - *the destructor releases the resource and never throws exceptions;*
  - ***always** use the resource via an instance of a **RAII-class** that either*
    - *has automatic storage duration or temporary lifetime itself, or*
    - *has lifetime that is bounded by the lifetime of an automatic or temporary object*

2. File streams.

- Relative vs absolute path
  - Relative path - path from the current directory
  - Absolute path
  - *Note: Windows's paths use '\' and to write that in C++ we use '\\'*

- File streams, just like the IO streams - cout and cin, are streams for input and output of information, but to a certain **file**.



Stream classes hierarchy

- std::**ifstream**, std::**ofstream** and std::**fstream** are *classes* and we'll be creating objects from these classes to interact with files.
  Note: ***cin** is an object of type istream, and **cout**, **cerr** and **clog** - ostream*

- Text files
  - Used for storing text (usually .txt)
  - Text editors can view the information in the file
  - Easy IO with >>, << and getline *(just like cin and cout)*

- Binary files
  - Used for storing objects (usually .dat)
  - Text editors won't display the file correctly
  - For IO read and write methods must be used

- Steps when working with files
  1. Ask for a file to be opened.
  2. Check if the file has been opened successfully.
  3. Work with the file.
  4. Close the file as soon as we are done with it.

- File stream creation and methods
  - `std::[i/o]fstream <identifier>;`
  - `open(<file_path>, [flags])` - open file for reading or writing.
  - `close()` - close the file.
  - `eof()` - returns true when the end of the file is reached.
  - `bad()` - returns true if a reading or writing operation fails.
  - `is_open()` - returns true if the file is successfully opened.
  - `gcount()` - returns the number of bytes read thus far.
  - `ignore(<num>)` - skips <num> bytes from the file.
  - `peek()` - checks the next available character.
  - `tellg()` - returns the position of the **get pointer**.
  - `tellp()` - returns the position of the **put pointer**.
  - `seekg(<pos>, [way])` - changes the position of the **get pointer**.
  - `seekp(<pos>, [way])` - changes the position of the **put pointer**.
    `[way]` can be any of `ios_base::beg`, `ios_base::cur`, `ios_base::end`.
    Example: `seekp(5, ios_base::end)` means, move the put pointer 5 bytes from the end of the file towards the beginning.
  - `write(<place>, <size>)` - write <size> number of bytes in a binary file, reading the bytes from <place>.
  - `read(<place>, <size>)` - read <size> number of bytes from a binary file.
    `<place>` is an *address* of type `char*` (or `const char*` for write).
    `<size>` is the number of bytes to be written/read.

Notes:

When opening files for **writing**, by default *the content of the file is erased*.
When opening a **non-existing** file for reading, that file *will be created*.

- Flags when opening files
  - Additional options can be added when opening files that are defined in the second parameter of the open method (or in the constructor).
  - Flags:
    `std::ios::app` - when writing, **append** at the end of the file.
    `std::ios::trunc` - when a file has been opened for writing, erase (truncate) its content first before writing.
    `std::ios::binary` - open a file in binary mode.
    `std::ios::in` - open file for reading *(when an fstream object is used)*
    `std::ios::out` - open file for reading *(when an fstream object is used)*
  - *Note: Multiple flags can be added using the binary OR operator (|)*
    Example:
    ```
    std::fstream file("myFile.bin", std::ios::in | std::ios::binary);
    ```

- *Simple text file example:*
```
ofstream out("file.txt"); // Step 1
if (!out) { // Step 2
    cout << "Couldn't open file for writing!" << endl;
} else {
    out << "This text will go in the file" << endl; // Step 3
    out.close(); // Step 4
}
```

- *Simple binary file example:*
```
int num = 42;
char str[MAX_LEN];
strcpy(str, "Test string");

ofstream outBin("data.bin"); // Step 1
if (!outBin) { // Step 2
    cout << "Couldn't open file for writing!" << endl;
} else {
    // Writing primitive data types
    outBin.write((const char*) &num, sizeof(num));

    // Writing c strings involves writing its length first
    // and then writing the actual string
    int len = strlen(str);
    outBin.write((const char*) &len, sizeof(len));
    outBin.write((const char*) str, len + 1);

    outBin.close(); // Step 4
}
```

- More examples - in the practicum.