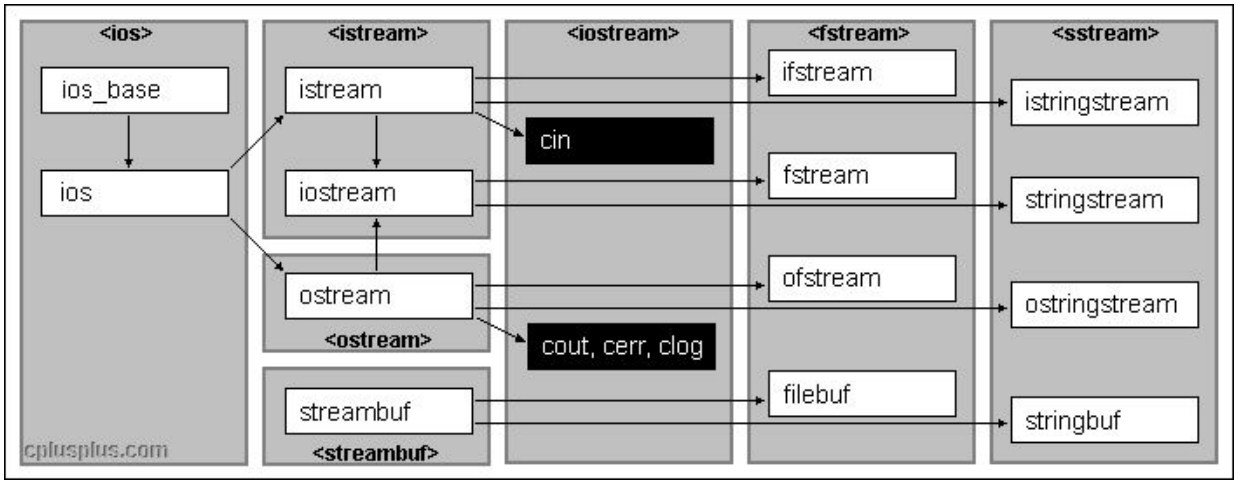
Lenguaje de Programación C++

Lectura y Escritura de archivos

Streams

C++ provee una abstracción de ***streams*** que permite trabajar con los dispositivos de entrada y salida.

La podríamos ver como una abstracción que acepta secuencia de bits, una interfaz independiente a los dispositivos, podemos transmitir datos a través de ellos sin preocuparnos de los dispositivos conectados.



Headers de Streams

Algunos de los más comunes:

<iostream> Nos da la salida y entrada de streams con formato.

- cin: entrada estándar (teclado)
- cout: salida estándar (consola)
- clog: stream de log (consola)
- cerr: stream de error estándar (consola)

<fstream> Nos da la salida y entrada de streams con formato a archivos

<iomanip> provee facilidades para la manipular el formateo de la salida.

Manipulando streams

Es posible controlar el formato de salida con precisión, utilizando los manipuladores de E / S y para eso, debemos incluir el encabezado iomanip,
#incluye <iomanip>

adjustfield	floatfield	right	skipws
basefield	hex	scientific	unitbuf
boolalpha	internal	showbase	uppercase
dec	left	showpoint	
fixed	oct	showpos	

Algunos ejemplos

```
std::cout.width(10);              // member function
std::cout << std::setw(10); // manipulator
```

```
cout.width(8);
cout.fill('0');
cout << hex << 77;
```

Prints out 0000004d

```
cout.width(8);
cout.fill('0');
cout << hex << 77;
```

Prints out 0000004d

Archivos

ifstream – para las funciones de input file

ofstream – para las funciones de output file

fstream – para ambas funciones input and output

Al igual que en C
tenemos modificadores para
usar al momento de la apertura

ios::binary – abre el archivo a modo binario
ios::ate – abre el archivo al final
ios::app – abre el archivo en modo append
ios::out – abre el archivo en modo escritura
ios::in – abre el archivo en modo lectura
ios::trunc – trunca el contenido del archivo

los modos se concatenan con el operadorr (‘|’)

Ejemplo de apertura para lectura

```
#include<iostream>
#include<fstream>
using namespace std;

int main(){
        ifstream i_file1("../nombre.txt", ios::in);

        ifstream i_file2("../nombre.txt", ios::in | ios::binary);

        ifstream i_file3("../nombre.txt");

        ifstream i_file4;

        i_file4.open("../nombre.txt");

        return 0;

}
```

Algunas formas posible de apertura.
Hay que tener en cuenta que es conveniente usar istream cuando realmente sea necesario

Comentario:

Siempre que se abre un archivo se debe controlar que se lo haya hecho en forma correcta

Chequeando errores y cierre

```
if (i_file.is_open()){
     /*codigo*/
}else{
     cerr << "Ouch!"
}
```

```
if (in_file){
     /*codigo*/
}else{
     cerr << "Ouch!"
}
```

```
if (!i_file){
     cerr << "Ouch!"
}
```

```
i_file.close();
```

Lectura y Escritura

- Archivos de Texto: para streams regulares, usamos los operadores << y >>.
- Archivos binarios:
  Usamos get, put y peek, read y write

write – escribe arreglos de bytes (char*) en el archivo.
read – lee un arreglo de bytes (char*) del archivo

get –toma un caracter del archivo.
put – escribe un caracter en el archivo

Ejemplo de lectura por renglones

```
string fila;

while(!i_file.eof()){
    getline(i_file, fila);
    cout << fila << endl;
}
```

Ejemplo de lectura simple

```
int num { };
double total { };
string nombre { };

i_file >> num;
i_file >> total >> nombre;
```

Escribiendo en un archivo

La apertura es similar a la que se hace para la lectura exclusiva. Teniendo en cuenta que los modificadores serían otros, como append, en caso de no usar este, el archivo comienza a escribir desde el principio pisando lo que se tenía

```
#include<iostream>
#include<fstream>
using namespace std;

int main(){
    ofstream i_file1{"../nombre.txt", ios::out, ios:app};

    ofstream i_file2{"../nombre.txt", ios::out | ios::binary};

    ofstream i_file3{"../nombre.txt"};

    ofstream i_file4;

    i_file4.open("../nombre.txt", ios::out);

    return 0;

}
```

Al igual que en lo dicho anteriormente lo correcto es validar que no haya error en la apertura.

Escritura usando el operador de flujo

```
int num {100};
double total {254.15};
string nombre {"Pablo"};

o_file << num << "\n"
         << total << "\n"
         << nombre << endl;
```

Como vemos, la escritura y la lectura de un archivo de texto es relativamente simple con el uso de los operadores de flujo.

Escribiendo estructuras

Es posible escribir bloques de datos en un archivo.

Es muy importante no escribir ningún puntero en un archivo, su ubicación podría, y lo más probable es que cambie en la próxima ejecución.

Debemos escribir las estructuras en forma binaria, convirtiéndolas como un puntero de caracteres. Esto nos permite escribir también a objetos de tipo clases.

Sintaxis de la escritura binaria

```
read( (char *) & ob, sizeof(ob));

write( (char *) & ob, sizeof(ob));
```

Banderas de estados

Un archivo tiene 4 posibles estados: (deberían tener que chequearse al usar archivos)

- good() – Retorna verdadero si ninguna de las otras banderas fueron seteadas.
- eof() – Fin del archivo
- fail() – Retorna true en los mismos casos que bad(), pero también cuando se ha encontrado un error de lectura como por ejemplo se encontró un caracter cuando se buscaba un float.
- bad() – el stream ha fallado al realizar una determinada tarea. El flujo puede esta corrupto.