Manejo de archivos

Manejo de archivos

Principalmente se hace uso de la librería fstream.h

Los objetos ifstream, ostream y fstream son usados para operaciones de lectura, escritura y lectura/escritura en archivos, respectivamente.

```
ofstream archivo;  // objeto de la clase ofstream, se va a escribir

archivo.open("datos.txt"); // Se utiliza el método open
```

Manejo de archivos

El archivo se puede abrir sin la necesidad de utilizar el método open, utilizando la ruta del archivo en el constructor:

```
ofstream archivo("datos.txt");  // constructora de ofstream
```

Se puede especificar el modo de apertura del archivo incluyendo un parámetro adicional en el constructor:

member constant	stands for	access
`in`	**in**put	File open for reading: the *internal stream buffer* supports input operations.
`out`	**out**put	File open for writing: the *internal stream buffer* supports output operations.
`binary`	**binary**	Operations are performed in binary mode rather than text.
`ate`	**at e**nd	The *output position* starts at the end of the file.
`app`	**app**end	All output operations happen at the end of the file, appending to its existing contents.
`trunc`	**trunc**ate	Any contents that existed in the file before it is open are discarded.

These flags can be combined with the bitwise OR operator (|).

class	default mode parameter
`ofstream`	ios::out
`ifstream`	ios::in
`fstream`	ios::in | ios::out

Manejo de archivos

fstream archivo("Pruebas.txt", ios::app); // Agrega contenido al archivo

fstream archivo("Pruebas.txt", ios::ate); // El puntero de escritura se inicializa al final de archivo.

fstream archivo("Pruebas.txt", ios::in); // Operaciones de lectura

fstream archivo("Pruebas.txt", ios::out); // Operaciones de escritura

fstream archivo("Pruebas.txt", ios::binary); // Operaciones binarias.

fstream archivo("Pruebas.txt", ios::trunc); // Se borra lo que haya en el archivo.

Manejo de archivos

Se puede utilizar la función miembro fail para detectar un error al abrir el archivo

```
if(archivo.fail())
    cerr << "Error al abrir el archivo Pruebas.txt" << endl;
```

La bandera EOF se activa cuando se ha llegado al final del archivo.

```
archivo.eof()
```

Manejo de archivos: Funciones que nos permiten verficar procesos en los archivos

- good Produce un 1 si la operación previa fué exitosa.
- eof Produce un 1 si se encuentra el final del archivo.
- fail Produce un 1 si ocurre un error.
- bad Produce un 1 si se realiza una operación inválida.

```
archivo.getline(linea, sizeof(linea));

    if(origen.archivo()) // si lectura ok
```

Manejo de archivos que no son texto

Por ultimo, para realizar copias, escrituras, o lecturas de archivos que no son texto, se debe utilizar las operaciones binarias (ios::binary), el siguiente programa hace la copia del Archiv04.exe a Copia.exe., en este caso se utilizan los métodos read y write para copiar el archivo.

```
ifstream origen("archivo.exe", ios::binary);

ofstream destino("Copia.exe", ios::binary);

origen.read(linea, sizeof(linea));

destino.write(linea, sizeof(linea));
```

Manejo de archivos

Para mover los apuntadores de archivo a posiciones específicas se utilizan dos funciones: seekg( ) coloca el apuntador de escritura de archivo en un lugar específico, y seekp( ) mueve el apuntador de lectura a una posición específica en el archivo

seekp(desplazamiento, posicion)

seekg(desplazamiento, posicion)

Posibles valores de posición:

ios::beg	offset counted from the beginning of the stream
ios::cur	offset counted from the current position
ios::end	offset counted from the end of the stream

Manejo de archivos

Para la posición se pueden especificar las siguientes banderas.

ios::beg Desde el principio del archivo

ios::cur Desde la posición actual del apuntador

ios::end Desde el fin del archivo

letras.seekg(7, ios::beg); // obtiene la letra en la posición 7 del archivo

letra=letras.get();

Manejo de excepciones

Excepciones

Una excepción es un problema que se presenta durante la ejecución de un programa.

Proporcionan una manera de transferir el control de una parte de un programa a otro.

Se basan en tres palabras claves: **try, catch, throw**.
El programador encierra dentro de un bloque **try** el código que podría generar error que produciría una excepción. El **try** va seguido por uno o más bloques **catch**, dónde cada bloque **catch** contiene un **manipulador de excepciones**.
Throw: emite una excepción cuando un problema aparece. Va dentro del try.

Ejemplos

```
double division(int a, int b)
{
   if( b == 0 ) throw "Division by zero condition!";
   return (a/b);
}

try {
     z = division(x, y);
     cout << z << endl;
}catch (const char* msg) {
     cerr << msg << endl;
}
```

Ejemplos

```
try
{
    throw 20;
}
catch (int e)
{
    cout << "An exception occurred. Exception Nr. " << e << '\n';
}
```

Ejemplos

```
try
{
    throw 20;
}
catch (int e)
{
    cout << "An exception occurred. Exception Nr. " << e << '\n';
}
```

Ejemplos: Múltiples handlers

```
try {
  // código aquí
}
catch (int param) { cout << "int exception"; }
catch (char param) { cout << "char exception"; }
catch (...) { cout << "default exception"; }
```

Ejemplos: Múltiples handlers

```
try {
  // código aquí
}
catch (int param) { cout << "int exception"; }
catch (char param) { cout << "char exception"; }
catch (...) { cout << "default exception"; }
```

Ejemplos: Standard exceptions

C++ proporciona una librería que permite hacer uso de excepciones comunes que pueden ocurrir en un programa. Para utilizarla se debe incluir:

#include <exception>

Esta clase tiene una función virtual llamada what que devuelve una secuencia de caracteres terminada en un carácter nulo (de tipo char *) y que puede ser sobrescrita en clases derivadas para contener algún tipo de descripción de la excepción.

Ejemplos: Standard exceptions

Excepciones que se pueden encontrar:

bad_alloc thrown by new on allocation failure

bad_cast thrown by dynamic_cast when it fails in a dynamic cast

bad_function_call thrown by empty function objects

Ejemplos: Standard exceptions

```
try  {
        int* myarray= new int[1000];
}
catch (exception& e){
        cout << "Standard exception: " << e.what() << endl;
}
```

STL

Standard Template Library

Colección de estructuras de datos genéricas

Un vector (o tabla, o array) es la estructura de datos más básica: un contenedor que almacenan elementos del mismo tipo en n posiciones consecutivas de memoria.

Para que el programa use vectores, se debe importar la librería:

#include <vector>

Declarando e instanciando vectores

Todos los elementos de un vector son del mismo tipo, y este se debe especificar en el momento de crearlo: un vector<T> es un vector cuyos elementos son de tipo T.

El tamaño del vector también se especifica en el momento de su creación; de otro modo, el vector empieza vacío.

```
vector<int> v(3); //vector de 3 coordenadas enteras
vector<string> vst(10); //vector de 10 strings
vector<bool> w; //vector de booleanos, vacío

vector<int> v(10, 3); //vector de 10 enteros, inicializados en 3
```

Actuando como variables normales

Los vectores, al igual que todas los contenedores de la STL, tienen la muy útil propiedad de actuar como variables normales y corrientes: podemos asignarlos, compararlos, copiarlos, pasarlos como parámetro, hacer que una función los devuelva, etc.

```
vector<int> v(10, 3);
vector<int> w = v; //w es una copia de v
++v[0]; //v y w son ahora distintos
w.resize(0); //ahora w esta vacio
v = w; //ahora v tambien esta vacio
bool iguales = v==w; //iguales vale true
```

Cambiando su tamaño

Otra de las ventajas de los vectores sobre los arrays clásicos es que resultan muy prácticos para almacenar una lista de elementos el tamaño de la cual pueda ir aumentando. Para ello, se usa el método push_back del vector.

```
string palabra;
vector<string> vs;
while (cin >> palabra) {
	vs.push_back(palabra);
}
```

Los iteradores

La STL ofrece otro modo de recorrer los vectores distinto al método tradicional de usar un indice i. Son los llamados iteradores.

Esencialmente, un iterador a un vector<T> es una variable de tipo vector<T>::iterator que actúa como un apuntador a un elemento del vector. Si it es un iterador, las operaciones ++it y --it sirven para desplazar el iterador arriba y abajo dentro del vector, y la operacion *it devuelve el elemento apuntado por el iterador.

vector<int>::iterator it = v.begin();

Otras utilidades de los iteradores

Son necesarios para poder hacer una de las operaciones m´as importantes en un vector: ordenar sus elementos. La STL permite ordenar muy fácilmente los elementos de un vector<T>: basta con hacer una llamada a la función sort.

sort(v.begin(), v.end()); //ordena un vector de menor a mayor

Para usar la función sort, hay que haber hecho un #include <algorithm> al principio del código.

Matrices: Vectores de vectores

El tipo T del vector<T> puede ser prácticamente cualquier tipo de datos. En particular, podemos tener vectores de vectores, es decir, vectores donde el tipo T es otro vector. Por ejemplo, el tipo vector<vector<double> > representa matrices (tablas) de reales.

A diferencia de los arrays bidimensionales tradicionales del C++, no existe la obligación de que todas las filas tengan el mismo numero de elementos: cada vector fila es independiente de los demás, y la única restricción es que todos ellos deben contener elementos del mismo tipo.