Programación Sobre Redes

L2: Referencias, arreglos, vectors, y streams (parte II)

Nicolás Mastropasqua

April 16, 2020

Instituto Industrial Luis A. Huergo

Contenidos

1. Repaso
2. Streams
3. Entrada/Salida con archivos

Repaso

Vectores y arreglos

Arreglos

- Los arreglos son soportados nativamente por C

Vectores y arreglos

Arreglos

- Los arreglos son soportados nativamente por C
- Tienen tamaño fijo. No se pueden redimensionar.

Vectores y arreglos

Arreglos

- Los arreglos son soportados nativamente por C
- Tienen tamaño fijo. No se pueden redimensionar.

Vectores y arreglos

Arreglos

- Los arreglos son soportados nativamente por C
- Tienen tamaño fijo. No se pueden redimensionar.

Vectores

Vectores y arreglos

Arreglos

- Los arreglos son soportados nativamente por C
- Tienen tamaño fijo. No se pueden redimensionar.

Vectores

- Sólamente disponibles en C++, utilizando la biblioteca adecuada

Vectores y arreglos

Arreglos

- Los arreglos son soportados nativamente por C
- Tienen tamaño fijo. No se pueden redimensionar.

Vectores

- Sólamente disponibles en C++, utilizando la biblioteca adecuada
- Implementan un arreglo dinámico

Vectores y arreglos

Arreglos

- Los arreglos son soportados nativamente por C
- Tienen tamaño fijo. No se pueden redimensionar.

Vectores

- Sólamente disponibles en C++, utilizando la biblioteca adecuada
- Implementan un arreglo dinámico
- Ojo! Por defecto, los vectors se pasan **por copia**

Streams

Streams

Streams

Un objeto que representa un flujo de datos. Esto puedes ser una fuente de entrada o un destino de salida de los datos del programa.

Tipos de streams

Repasando

- Un input stream es aquel que se utiliza para leer datos desde una fuente.

Tipos de streams

Repasando

- Un input stream es aquel que se utiliza para leer datos desde una fuente.
- Un output stream es aquel que se utiliza para escribir datos hacia un destino.

Tipos de streams

Repasando

- Un input stream es aquel que se utiliza para leer datos desde una fuente.
- Un output stream es aquel que se utiliza para escribir datos hacia un destino.
- cin es un objeto de la clase istream para leer desde la entrada estandar, es decir, los datos son ingresados a través del teclado.

Tipos de streams

Repasando

- Un input stream es aquel que se utiliza para leer datos desde una fuente.
- Un output stream es aquel que se utiliza para escribir datos hacia un destino.
- cin es un objeto de la clase istream para leer desde la entrada estandar, es decir, los datos son ingresados a través del teclado.
- cout es un objeto de la clase ostream para escribir en la salida estándar, es decir, los datos se ven en la pantalla.

Tipos de streams

Repasando

- Un input stream es aquel que se utiliza para leer datos desde una fuente.
- Un output stream es aquel que se utiliza para escribir datos hacia un destino.
- cin es un objeto de la clase istream para leer desde la entrada estandar, es decir, los datos son ingresados a través del teclado.
- cout es un objeto de la clase ostream para escribir en la salida estándar, es decir, los datos se ven en la pantalla.

Utilizando output stream

El operador « inserta los datos después de él en el stream que lo precede. Se llama operador de inserción.

Utilizando output stream

El operador « inserta los datos después de él en el stream que lo precede. Se llama operador de inserción.

Ejemplo

```
int edad = 19;
string x = "pepe";
cout << 120;
cout << x;
cout << "Tengo " << edad << " actualmente.";
```

Utilizando output stream

El operador « inserta los datos después de él en el stream que lo precede. Se llama operador de inserción.

Ejemplo

```
int edad = 19;
string x = "pepe";
cout << 120;
cout << x;
cout << "Tengo " << edad << " actualmente.";
```

Que veremos en pantalla?

Utilizando output stream

El operador « inserta los datos después de él en el stream que lo precede. Se llama operador de inserción.

Ejemplo

```
int edad = 19;
string x = "pepe";
cout << 120;
cout << x;
cout << "Tengo " << edad << " actualmente.";
```

Que veremos en pantalla?
120pepeTengo 19 actualmente.

Utilizando input stream

El operador » extrae datos del stream que lo precede, y lo guarda en la variable escrita del lado derecho. Se llama operador de extraccion.

Ejemplo

```
int edad;
cout << "Ingrese su edad " << endl;
cin >> edad;
cout << "Ingrese su nombre " << endl;
string nombre;
cin >> nombre;
```

Utilizando input stream

El operador » extrae datos del stream que lo precede, y lo guarda en la variable escrita del lado derecho. Se llama operador de extraccion.

Ejemplo

```
int edad;
cout << "Ingrese su edad " << endl;
cin >> edad;
cout << "Ingrese su nombre " << endl;
string nombre;
cin >> nombre;
```

Que pasa en este caso?

Entrada/Salida con archivos

Usando archivos

No necesariamente tenemos que usar la entrada y salida estándar, sino que podemos utilizar streams mas generales. Podemos definir un ifstream (hereda de istream) o un ofstream(hereda de ostream) que funcione de manera análoga a los estándar, pero escribiendo o leyendo de donde lo decidamos.

Tipos

- ifstream: Lee de un archivo.
- ofstream: Escribe en un archivo.

Mas info: http://www.cplusplus.com/doc/tutorial/files/

Usando archivos

Ejemplos

```
ifstream archivoDeEntrada("miArchivo.in");
ofstream archivoDeSalida("miArchivo.out");
int z;
archivoDeEntrada >> z;
archivoDeSalida << "Escribiendo a un archivo!";
```

La lectura y escritura es igual a los casos anteriores, solo que antes hay que declarar desde donde vamos a leer o a donde escribiremos.

Consejos

- Al abrir un stream hay que checkear si esta operacion fue realizada correctamente y manejarlo adecuadamente (con .is open())
- Al terminar de utilizarlo hay que cerrarlo haciendo .close() (de todas maneras se cerrara automaticamente cuando termine el scope).
- Es importante cerrarlo ya que podría pasar que otro proceso no puedo accederlo porque aún esta abierto. Además el sistema libera los recursos utilizados (por ejemplo, buffers de memoria)

Consejos

Ejemplo

```
...
ofstream miArchivo;
miArchivo.open("example.txt");
if (miArchivo.is_open())
{
   miArchivo << "Una linea." << endl;
   miArchivo << "Otra linea." << endl;
   miArchivo.close();
}
...
```

Problema

- Hasta ahora vimos ejemplos donde siempre sabíamos cuanto teníamos que leer. Que sucede si queremos crear un ciclo que lea hasta que se terminen de leer todos los datos de dicho stream?

Problema

- Hasta ahora vimos ejemplos donde siempre sabíamos cuanto teníamos que leer. Que sucede si queremos crear un ciclo que lea hasta que se terminen de leer todos los datos de dicho stream?
- Luego de leer desde un input stream, la expresion inputStream » x evalua a false si se termino de leer todo el archivo, o a true si esto no sucedió.

Problema

Ejemplo

```
ifstream miArchivo;
miArchivo.open("example.txt");
int var;
while( inputStream >> var )
{
   ...
}
```

Para saber si terminamos de leer un archivo también podemos utilizar inputStream.eof(), que devuelve true si llego al final del archivo y false en caso contrario.

Funciones útiles

- Si nuestra entrada es Alice-Bob y queremos leer un string que solamente contenga Alice, podemos hacer lo siguiente: getline(inputStream,stringDondeGuardo,'-');

Funciones útiles

- Si nuestra entrada es Alice-Bob y queremos leer un string que solamente contenga Alice, podemos hacer lo siguiente: getline(inputStream,stringDondeGuardo,'-');
- Esto indica que se debe leer de un stream (y guardarlo en el string dado) hasta que se encuentre un char guion. All se detendra de leer, y descartara el guion ledo. Lo pueden utilizar con otros caracteres, como por ejemplo la coma, el punto y coma o espacio.

¿Dudas?