

ENTRADA/SALIDA Y PASAJE DE PARÁMETROS

Algoritmos y Estructuras de Datos I

26 de Marzo de 2018

MENÚ DEL DÍA

- ▶ Entrada y salida desde consola
- ▶ Entrada y salida desde archivos
- ▶ Pasaje de parámetros por valor y por referencia

ENTRADA/SALIDA

ENTRADA Y SALIDA DESDE CONSOLA

- ▶ En los sistemas UNIX la consola (teclado y monitor) permitía interactuar con una aplicación
- ▶ cout: console out. Imprime por pantalla un dato
- ▶ cin: console in: Lee un dato del teclado

El operador << inserta el flujo de datos en la salida estandar (la pantalla).

```
1  #include <iostream>
2
3  int main() {
4
5      std::cout << "Hola Mundo" << std::endl;
6
7      return 0;
8  }
```

OPERADOR DE EXTRACCIÓN >>

El operador >> extrae el flujo de datos de la entrada estandar (teclado).

```
1 #include <iostream>
2
3 int main() {
4
5     char letra;
6     std::cin >> letra;
7
8     return 0;
9 }
```

¿QUÉ HACE ESTE PROGRAMA?

```
1 #include <iostream>
2
3 int main() {
4     std::cout << "Ingrese un valor entero" << std::endl;
5     int valor = 0;
6     std::cin >> valor;
7     std::cout << "El valor ingresado fue " << valor << std::endl;
8     return 0;
9 }
```

STREAM

cin y cout son streams (*flujos*)

- ▶ Hay dos tipos principales de streams:
 - ▶ input stream: flujo de datos que representa una fuente de entrada (Ej: cin, ifstream).
 - ▶ output stream: flujo de datos que representa un destino de salida (Ej: cout, ofstream)

E/S CON ARCHIVOS

- ▶ Usamos:
 - ▶ ofstream: para escribir (write/salida)
 - ▶ ifstream: para leer (read/entrada)
- ▶ Escribir texto en un **archivo de texto plano** en C++ es similar a escribir texto por consola. Usamos:
 - ▶ operador de inserción (>>) para guardar texto en un archivo.
 - ▶ operador de extracción (<<) para extraer texto de un archivo.

ESCRIBIR VALORES EN UN ARCHIVO

- ▶ Para escribir archivos tenemos que seguir el siguiente protocolo:
 - ▶ Declarar un ofstream
 - ▶ Abrir el archivo (open)
 - ▶ Escribir (1 o más veces) (usar <<)
 - ▶ Cerrar el archivo (close)

EJEMPLO

```
1 #include <iostream>
2 #include <fstream>
3
4
5 int main() {
6     int a = 5;
7     ofstream fout;
8     fout.open("archivo.txt"); // abre el archivo para escritura
9     fout << "Hola, archivo!" << endl;
10    fout << "Ahora, un entero: " << a << endl;
11    fout << "Tambien una expresion: " << (a+2) << endl;
12    fout.close(); // cerramos el archivo
13    return 0;
14 }
```

¿Qué pasa al ejecutar este programa?

ESCRIBIR VALORES DE DISTINTOS TIPOS DE DATOS

- ▶ Hasta ahora escribimos únicamente enteros (`int`)
- ▶ También podemos escribir valores `bool`, `float`, `char`, etc.

EJEMPLO

```
1 #include <iostream>
2 #include <fstream>
3
4
5 int main() {
6     char c = 'x';
7     float f = 1.5;
8     bool b = true;
9     ofstream fout;
10    fout.open("archivo.txt"); // abre archivo
11    fout << c << endl; // escribe x (char)
12    fout << f << endl; // escribe 1.5 (float)
13    fout << b << endl; // escribe 1 (bool)
14    fout.close(); // cierra archivo
15    return 0;
16 }
```

EJERCICIOS GUÍA

ENTRADA/SALIDA

Resolver los ejercicios 11 y 12 de la guía.

ESCRIBIR AL FINAL DE UN ARCHIVO EXISTENTE

- ▶ ¿Qué hace la operación `ofstream.open("archivo.txt")` si `archivo.txt` ya existe?
 - ▶ Si no existe, crea el archivo
 - ▶ Si existe, sobrescribe todo su contenido (borra lo que había antes)
- ▶ ¿Cómo podemos hacer para que el contenido anterior sea respetado?
 - ▶ Para escribir al final del archivo hay que abrirlo en modo `append`
 - ▶ Para abrir un archivo en modo append, hay que usar `ofstream.open("archivo.txt", ios_base::app)`

LEER DATOS DESDE UN ARCHIVO

- ▶ Para leer declaramos un `ifstream` y usamos el operador `>>`.
- ▶ Es importante conocer de antemano el orden y tipo de datos de los elementos a leer del archivo.
- ▶ **Aclaración:** En esta materia, vamos a suponer que el archivo contiene sólo una línea de valores separados por espacios o saltos de línea (enter).
- ▶ Debemos seguir el siguiente protocolo:
 - ▶ Declarar un `ifstream`
 - ▶ Abrir el archivo (`open`)
 - ▶ Leer (1 o más veces) (usando `>>`)
 - ▶ Cerrar el archivo (`close`)
- ▶ Ejemplo: Leer un archivo `entrada.txt` que contiene dos enteros separados por un espacio en blanco (ejemplo: "15 20")

DEMO: LEER DATOS DESDE UN ARCHIVO

```
1  #include <iostream>
2  #include <fstream>
3
4
5  int main() {
6      int a =0;
7      int b =0;
8      ifstream fin;
9      fin.open("entrada.txt"); // abre el archivo para lectura
10     fin >> a; // lee el 15
11     fin >> b; // lee el 20
12     fin.close(); // cierra el archivo
13     return 0;
14 }
```

EJERCICIO 13 GUÍA

EJERCICIO 13

Resolver el ejercicio 13 de la guía.

FUNCIÓN END-OF-FILE (EOF)

- ▶ Además de open y close tenemos la función eof().
- ▶ La función eof() retorna true si ya no hay más contenido del archivo para leer.
- ▶ Usaremos eof() sólo cuando abrimos un archivo para lectura.
- ▶ Ejemplo:
 - ▶ Leer de un archivo una lista de enteros y calcular la suma de sus elementos.
 - ▶ No sabemos de antemano cuantos enteros hay almacenados en el archivo.

USANDO EOF()

```
1  #include <iostream>
2  #include <fstream>
3
4  int main() {
5      int suma = 0;
6      ifstream fin;
7      fin.open("archivo.txt");
8
9      while( !fin.eof() ) {
10         int a = 0;
11         fin >> a;
12         suma += a;
13     }
14
15     fin.close();
16     std::cout << "La suma de la lista es: " << suma << endl;
17     return 0;
18 }
```

MANEJO DE ERRORES

- ▶ Hasta ahora tenemos las funciones `open`, `close` y `eof` para operar con archivos.
- ▶ ¿Qué pasa cuando queremos abrir un archivo para lectura que no existe?
- ▶ ¿Qué pasa cuando no tenemos permisos para leer un archivo?
- ▶ ¿Qué pasa cuando no tenemos permisos para sobrescribir un archivo?
- ▶ Para todos esos casos, se puede consultar a la función `fail()`
- ▶ La función `fail()` retorna `true` si hubo una falla al intentar ejecutar una operación (por ejemplo: `open`, `close`)

DEMO:

¿QUÉ HACE EL SIGUIENTE PROGRAMA?

```
1  #include <iostream>
2  #include <fstream>
3
4  int main() {
5      int a =0;
6      int b =0;
7      ifstream fin;
8      fin.open("pepito.txt");
9      if (fin.fail()) { // true si hubo error al abrir
10         std::cout << "Error" << endl;
11     } else {
12         std::cout << "Abierto" << endl;
13     }
14     fin.close();
15     return 0;
16 }
```

RESUMEN: E/S EN C++

- ▶ `cout`: stream para escritura en la entrada estándar
- ▶ `cin`: stream para lectura de la entrada estándar
- ▶ `ifstream`: stream de lectura de archivos
- ▶ `ofstream`: stream para escritura de archivos
- ▶ `open()`: abre un archivo para escritura o lectura dependiendo del tipo de stream
- ▶ `close()`: cierra un archivo
- ▶ `<<` escribe un valor en el stream
- ▶ `>>` lee un valor del stream
- ▶ `eof()`: retorna `true` si la lectura del archivo llegó al final
- ▶ `fail()`: retorna `true` si la última operación falló

PASAJE DE ARGUMENTOS: COPIA VS REFERENCIA

PASAJE DE ARGUMENTOS

¿Qué pasa cuando ejecuto el siguiente programa?

```
1  #include <iostream>
2
3
4  void cambiarValor(int x) {
5      x = 15;
6  }
7
8  int main() {
9      int y = 10;
10     cambiarValor(y);
11     std::cout << "El valor de y es " << y << endl; // que valor se imprime?
12     return 0;
13 }
```

- ▶ Hasta ahora los argumentos entre funciones se pasaron siempre **por copia**.

PASAJE DE ARGUMENTOS EN C++

Pasaje por valor (o por copia)

- ▶ Coloca en la posición de memoria del argumento de entrada el **valor** de la expresión usada en la invocación.
- ▶ Si la función modifica el valor, no se cambian las variables en el llamador.
- ▶ **Declaración** de la función: `int f(int b);`
- ▶ **Invocación** de la función: `f(x)`, o bien `f(x+5)` o bien `f(5)`.
- ▶ Es el modo *por defecto* de pasaje de argumentos en C++.

PASAJE DE ARGUMENTOS

¿Cómo hacemos para que cambiarValor realmente cambie el valor de y?

```
1  #include <iostream>
2
3
4  void cambiarValor(int x) {
5      x = 15;
6  }
7
8  int main() {
9      int y = 10;
10     cambiarValor(y);
11     std::cout << "El valor de y es " << y << endl; // que valor se imprime?
12     return 0;
13 }
```

PASAJE DE ARGUMENTOS POR REFERENCIA

Pasaje por referencia

- ▶ La función recibe una dirección de memoria donde encontrar el argumento.
- ▶ La función puede leer esa posición de memoria pero también puede escribirla.
- ▶ Todas las asignaciones hechas dentro del cuerpo de la función cambian el contenido de la memoria **del llamador**.
- ▶ La expresión con la que se realiza la invocación debe ser necesariamente una *variable*.
- ▶ **Declaración** de la función: `int f(int &b);`
- ▶ **Invocación** de la función: `f(x)`, **pero no** `f(x+5)` **ni** `f(5)`.

PASAJE DE ARGUMENTOS POR REFERENCIA

- Modificamos el pasaje de parámetros con & para indicar que es una referencia y no una copia:

```
1  #include <iostream>
2
3
4  void cambiarValor(int& x) { // pasaje por referencia
5      x = 15;
6  }
7
8  int main() {
9      int y = 10;
10     cambiarValor(y);
11     std::cout << "El valor de y es " << y << endl; // que valor se imprime?
12     return 0;
13 }
```

ALIASING

- El operador & permite indicar que usaremos la **referencia** en lugar de la **copia** de una variable.
- Decimos una variable es un **alias** de otra variable si ambas apuntan a la misma porción de la memoria.

DEMO: ALIASING

```
1  #include <iostream>
2
3
4  int cambiarValor(int& a, int& b) {
5      // cuanto vale a? cuanto vale b?
6      b = 3;
7      // cuanto vale a? cuanto vale b?
8      a = 4;
9      // cuanto vale a? cuanto vale b?
10     return 0;
11 }
12
13 int main() {
14     int c = 0;
15     cambiarValor(c,c);
16     return 0;
17 }
```

EJERCICIOS

EJERCICIO 17

Resolver el ejercicio 17 de la guía.

PARÁMETROS **in**, **out**, **inout**

- ▶ En nuestro lenguaje de especificación los parámetros de una función pueden ser **in**, **out** o **inout**.
 - ▶ **in**: parámetros de entrada
 - ▶ **out**: parámetros de salida
 - ▶ **inout**: parámetros de entrada y de salida
- ▶ ¿Que podemos usar en C++ para implementar cada uno de estos parámetros?
 - ▶ Para un parámetro **in**: un argumento que se pase por **copia**.
 - ▶ Para un parámetro **inout**: un argumento que se pase por **referencia**.
 - ▶ Para un parámetro **out**:
 - ▶ un argumento que se pase por **referencia**, o
 - ▶ el valor de retorno de la función

RESUMEN: PASAJE POR COPIA VS. PASAJE POR REFERENCIA

Pasaje por copia	Pasaje por referencia
Por defecto. No es necesario anotar el parámetro	Hay que anotar el parámetro usando &
Crea una copia del dato	Crea un alias al dato
Los cambios son locales a la función	Los cambios modifican el dato original
in, out	inout, out

EJERCICIOS

EJERCICIOS

Resolver hasta el ejercicio 20 de la guía.