INGRESO Y SALIDA DE DATOS DESDE EL LENGUAJE C++

Elaborado por: Juan Miguel Guanira Erazo - PUCP

El lenguaje C++ pretende marcar una diferencia sustancial con el lenguaje C, en el sentido que los elementos que se empleen para realizar las operaciones de entrada y salida lo hagan de una forma más clara y simple, sin tener que indicar, de manera expresa, cómo se transformarán los caracteres que vienen de la entrada estándar, ni cómo se van a convertir los datos que se envíen a la salida estándar. En otras palabras ya no existirá una cadena de formato que decida cómo se realizarán estas transformaciones sin importar las variables que se empleen en la operación, ni su cantidad.

Por otro lado, a diferencia del lenguaje C, las operaciones de entrada y salida ya no se hacen a través funciones sino que se hacen por medio de *"objetos"* definidos por *"clases"* que han sido diseñadas de manera apropiada para este fin.

BIBLOTECAS DE FUNCIONES Y ESPACIOS DE NOMBRES

Para poder emplear los elementos que permitan la entrada y salida de datos del medio estándar de entrada y salida usaremos primordialmente la biblioteca iostream. Sin embargo, para poder manipular los datos, sobre todo para la salida de datos, se podrá emplear la biblioteca iomanip, como veremos más adelante.

Por otro lado, debemos tener presente que para el lenguaje C++ se han venido desarrollando una infinidad de bibliotecas de funciones y clases, lo que ha traído como consecuencia que en muchos casos se empleen en un mismo programa dos a más bibliotecas que definan las mismas clases u objetos, lo que trae como consecuencia una interrupción en el proceso de compilación. Para solucionar este problema, el lenguaje ha implementado un mecanismo por medio del cual los elementos definidos en una biblioteca se pueden colocar dentro de un ámbito, identificado con un nombre, ese ámbito se conoce con el nombre de "espacio de nombres" (la implementación de un espacio de nombres se desarrollará en el capítulo de funciones, aquí solo lo emplearemos). Para el caso de las bibliotecas iostream e iomanip, el espacio de nombres que se ha definido en ambos casos se denomina std que viene del inglés standard.

OBJETO *cout*

El objeto cout permite enviar al medio estándar de salida el resultado de expresiones que se le proporcionan, de un modo similar a printf. Sin embargo antes de mostrar su sintaxis, debemos aclarar ciertos puntos.

Como se indicó líneas arriba, las operaciones de entrada y salida en C++ no se hacen a través de funciones sino por medio de objetos, en este sentido cout es un objeto definido (también se dice "instanciado") de la clase ostream que se encuentran en la biblioteca iostream.

Debido a los espacios de nombres, tenemos tres formas de emplear el objeto cout (y cualquier otro de esta biblioteca o similar), a continuación las describimos:

1. Acompañando el objeto del espacio de nombres en cada invocación, esto es:

```
#include <iostream>
…
std::cout…   // El espacio de nombres std debe aparecer en cada invocación al objeto,
               separándolos con el operador de ámbito (::).
…
std::cout…
```

2. Empleando la cláusula using al inicio del módulo y después de la orden de pre procesador #include <iostream>, de la siguiente manera:

```
#include <iostream>
using std::cout;
…
cout      // Aquí ya no es necesario emplear el espacio de nombres.
          // Sin embargo cualquier otro elemento de iostream si lo requerirá
```

3. Utilizando la cláusula using acompañándolo de cláusula namespace, como se indica a continuación:

```
#include <iostream>
using namespace std;        // Con esto ningún elemento de iostream requerirá del espacio de
                               nombres
…
cout…
```

   Esta última forma de emplear el objeto cout es la que usaremos en adelante, debido a que una vez colocada la cláusula ya no se requerirá emplear el espacio de nombres en ningún elemento de las bibliotecas estándar de C++.

Por otro lado, para poder realizar la operación de salida de datos por medio del objeto cout, el lenguaje C++ ha sobrecargado el operador de desplazamiento de bits **<<** de modo que la operación se hará ahora a través de este operador que en adelante será nombrado como "operador de inserción de flujo". En el capítulo de funciones se desarrollará el tema de sobrecarga de operadores.

Modo de uso: (suponiendo que se ha colocado la cláusula using namespace std)

```
cout << expresión;
```

Esta operación permite enviar a la salida estándar un flujo de caracteres provenientes de la conversión del resultado de la expresión en una cadena de caracteres.

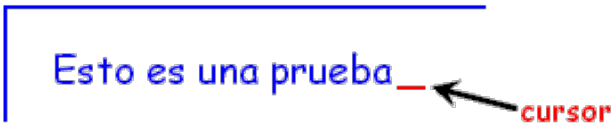
Una vez ejecutada la operación el sistema devuelve una referencia al objeto cout, por lo tanto esta operación se podrá concatenar con otras equivalentes de la manera siguiente:

```
cout << expresión1 << expresión2 << expresión3…;
```

Ejemplo:

```
cout << "Esto es una prueba";
```

Al ejecutar la orden se obtiene en la pantalla (medio estándar de salida):

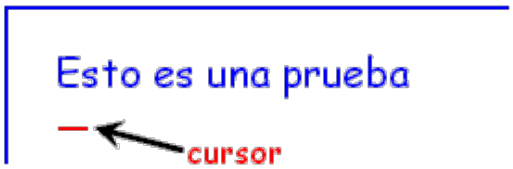


Como se puede observar la expresión es evaluada y enviada al medio estándar de salida, esto lo hace sin agregarle alguna otra información, como pueden ser espacios o cambios de línea.

Si se desea enviar los caracteres de cambio de línea se puede emplear el caracter '\n' como se hace en el leguaje C, sin embargo el lenguaje C++ proporciona una herramienta que hace lo mismo, esta se denomina endl, entonces podríamos ejecutar la orden de la siguiente manera:

```
cout << "Esto es una prueba" << endl;
```

Al ejecutar la orden se obtiene en la pantalla:



Por otro lado está el problema de los espacios entre resultados y la tabulación de datos. Los siguientes ejemplos muestran estos casos:

Empecemos con el siguiente código:

```
int a = 23, b = 765, c = 9;
cout << a << b << c << endl;
```

Al ejecutar la orden se obtiene en la pantalla:

Como se observa, los datos se pegan, situación que no se puede aceptar. Para intentar solucionar este problema se podría emplear cadenas con espacios o tabuladores ('\t'), sin embargo esto da muy malos resultados sobre todo cuando los datos son muy disparejos en tamaño.

Pero antes de ver cómo solucionar este problema debemos entender primero que el operador **<<** se aplica una sola vez con el objeto **cout**, pero como esta operación, luego de enviar el resultado al medio de salida, devuelve una referencia al objeto cout, permite concatenarla de modo que parezca que estamos ejecutando una sola instrucción. En otras palabras si escribimos la siguiente instrucción:

```
cout << a << b << c << endl;
```

El compilador va a interpretar esa línea como si hubiéramos escrito:

```
cout << a;
cout  << b;
cout << c;
cout  << endl;
```

Regresando al tema anterior, el lenguaje C++ nos proporciona herramientas que permite solucionar problema del formato. Estas herramientas se dan en dos categorías: mediante ***"funciones miembro"*** del objeto cout y mediante funciones que proporciona la biblioteca iomanip.

En el primer caso se emplea: cout.width(n), donde ***cout.width*** representa el llamado a ejecución de la función miembro (o también denominado método) ***width*** que pertenece a (es miembro de) ***cout***. n es un valor entero que indica la cantidad mínima de caracteres a emplear en la salida del valor (es equivalente al uso de %nd en el Lenguaje C). El ejemplo siguiente muestra el efecto de esta función:

```
int  a  =  2351,   b  = 765;
cout << a;
cout.width(10);
cout << a << endl;
cout << b << endl;
```

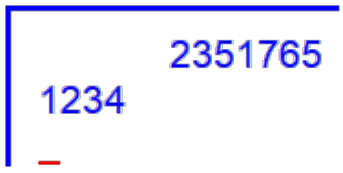
Al ejecutar el código se puede observar:

Aquí podemos ver algunas cosas muy significativas, la primera es que la instrucción de formato y la instrucción que envía el valor de salida van por separado. La segunda y más importante es que el formato se aplica solo a la operación de salida inmediata, al resto no se aplica por lo que deberá anteponerlo en cada instrucción de salida.

Observe este otro ejemplo:

```
int  a  =  2351,   b  =  765,   b  =  1234;
cout.width(10);
cout << a << b << endl;
cout << c << endl;
```

Al ejecutar el código obtenemos:

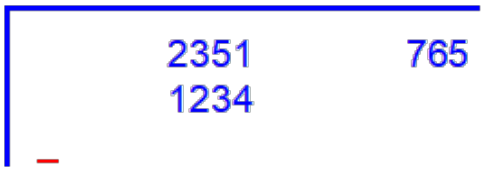


Observe que el único valor afectado por el formato es el de la variable "a".

Luego, para conseguir que se aplique a todas las variables se deberá hacer lo siguiente:

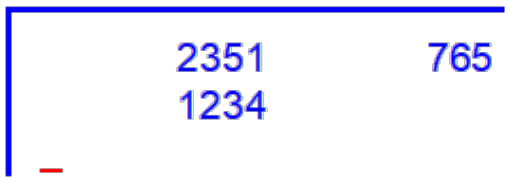
```
int a = 2351, b = 765, b = 1234;
cout.width(10);
cout << a ;              // Sin el cambio de línea: endl
cout.width(10);
cout << b << endl;
cout.width(10);
cout << c << endl;
```

Con lo que obtendremos lo siguiente:



En el segundo caso emplearemos una función definida en la biblioteca iomanip. Se trata de setw(n) (setw viene del inglés "set width"). Esta función, al igual que ***cout.width(n)***, define la mínima cantidad de caracteres en a que se representará el valor, sin embargo la forma de emplearla es mucho más sencilla y práctica, como veremos a continuación:

```
int a = 2351, b = 765, b = 1234;
cout << setw(10) << a << setw(10) << b << endl;
cout << setw(10) << c << endl;
```



El resultado es igual al del caso anterior pero del código podemos observar que con esta forma se puede concatenar en una sola instrucción, sin embargo, como en el primer caso el formato solo se aplica a la expresión contigua.

Es bueno indicar que ***setw*** forma parte del espacio de nombres ***std*** de la biblioteca ***iomanip***, por lo que al igual que con ***cout*** se requiere incluir este espacio de nombres explícitamente en el programa en cualquiera de sus formas.

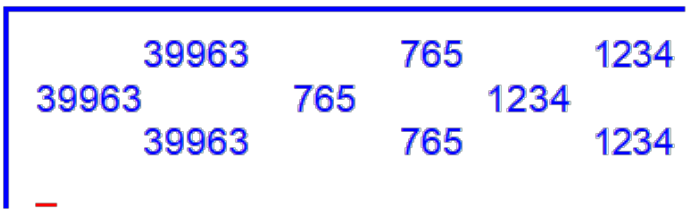
Banderas de formato

Las banderas de formato (***format flags***) son constantes que permiten definir ciertos atributos que influenciarán en la forma cómo aparecerán los datos en el medio de salida.

El ejemplo siguiente muestra alguno de ellos:

```
int a = 39963, b = 765, b = 1234;
cout << setw(10) << a << setw(10) << b << setw(10) << c << endl;
cout << left << setw(10) << a << setw(10) << b << setw(10) << c << endl;
cout << right << setw(10) << a << setw(10) << b << setw(10) << c << endl;
```

La salida de este programa será:

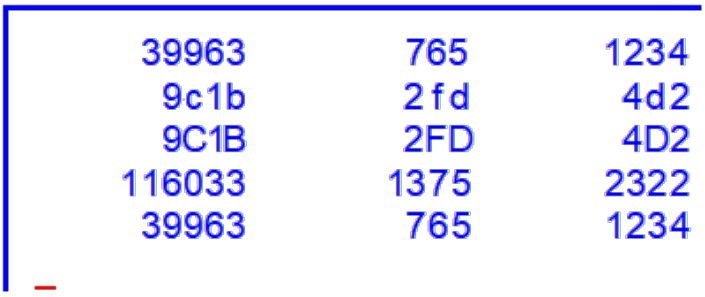


Observe que el formato se mantiene en todos los casos, sin embargo el uso de los elementos "left" y "right" harán que los resultados se alineen a la izquierda o a la derecha respectivamente. También se puede ver que una vez que se ejecuta el elemento, la alineación se mantenga hasta que se use de otro elemento lo cambie.

Un segundo ejemplo de estas banderas podemos verlas a continuación:

```
int a = 39963, b = 765, b = 1234;
cout << setw(10) << a << setw(10) << b << c << endl;
cout << hex << setw(10) << a << setw(10) << b << setw(10) << c << endl;
cout << uppercase << setw(10) << a << setw(10) << b << setw(10) << c << endl;
cout << oct << setw(10) << a << setw(10) << b << setw(10) << c << endl;
cout << dec << setw(10) << a << setw(10) << b << setw(10) << c << endl;
cout << nouppercase;
```

Al ejecutar el programa se verá lo siguiente:



Las banderas, "***hex***", "***oct***" y "***dec***" controlan la base (16, 8 y 10) en la que aparecerán los valores en la salida. Las banderas "***uppercase***", y "***nouppercase***" determinan la forma en que se mostrarán las letras en el formato hexadecimal, mayúsculas o minúsculas respectivamente. Observe también que los formatos se mantienen hasta que se levante otra bandera.

Finalmente mostraremos cómo poder rellenar los espacios colocados en los formatos con otro caracter diferente:

```
int a = 39963, b = 765, b = 1234;
cout << setw(10) << a << setw(10) << b << setw(10) << c << endl;
cout.fill('0');
cout << setw(10) << a << setw(10) << b << setw(10) << c << endl;
cout.fill(' ');

cout << setw(10) << a << setw(10) << b << setw(10) << c << endl;
cout << setfill('X');
cout << setw(10) << a << setw(10) << b << setw(10) << c << endl;
cout << setfill(' ');
```

La salida será como sigue:

```
    39963          765        1234
0000039963000000765000001234
    39963          765        1234
XXXXX39963XXXXXXX765XXXXXX1234
_
```

Formato en valores de punto flotante

Por defecto, los valores de punto flotante aparecerán con un formato pre establecido. El ejemplo siguiente muestra este detalle:

```
double f = 3.1415926535;
double g = 0.0000012345;

cout << f << endl;
cout << -f << endl;
cout << g << endl;
cout << -g << endl;
```

La salida será:

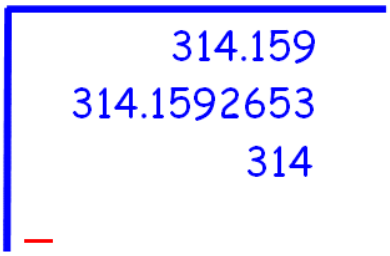
```
3.14159
-3.14159
1.2345e-006
-1.2345e-006
_
```

Para dar format a valores de punto flotante, C++ requiere del uso de dos herramientas, una que define la precisión en la que aparecerá el número (***cout.presicion(n)*** o ***setpresicion(n)***) y la otra que interpretará el significado de la precisión (***fixed***).

Por defecto ***cout.presicion(n)*** y ***setpresicion(n)***, define la cantidad de cifras o dígitos que se mostrarán en el medio de salida, ***entiéndase que esto se refiere al número de dígitos, no al número de cifras decimales***. El siguiente ejemplo muestra este detalle:

```
double f = 314.15926535;

cout << setw(14) << f << endl;          // Que es equivalente a:
cout.precision(10);                     // cout << setprecision(10) << setw(14) << f << endl;
cout << setw(14) << f << endl;          //
cout.precision(3);                      // cout << setprecision(3) << setw(14) << f << endl;
cout << setw(14) << f << endl;          //
```



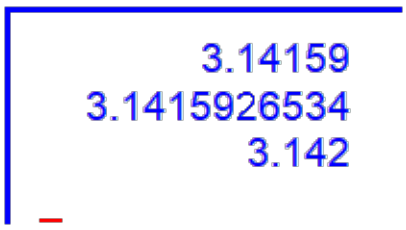
La figura anterior confirma lo explicado, ***cout.presicion(n)*** y ***setpresicion(n)*** no definen el número de decimales sino el número de dígitos que aparecerán en la salida.

Por eso se requiere otra herramienta, en este caso ***fixed***. Al activar esta bandera, el valor de precisión se interpretará como el número de decimales.

El siguiente programa muestra este efecto:

```
double f = 3.1415926535;
                                    // Que es equivalente a:
cout << fixed;
cout.precision(5);                  // cout << setprecision(5) << setw(14) << f << endl;
cout << setw(14)
cout.precision(10);                 // cout << setprecision(10) << setw(14) << f << endl;
cout << setw(14) << f << endl;      //
cout.precision(3);                  // cout << setprecision(3) << setw(14) << f << endl;
cout << setw(14) << f << endl;      //
cout.unsetf(ios::fixed);            // Se desactiva la orden fixed
```

La ejecución mostrará lo siguiente:



OBJETO *cin*

El objeto cin, al igual que la función scanf, permite leer, uno por uno, los caracteres de un flujo de caracteres que ingresa del medio estándar de entrada. Los caracteres son convertidos de acuerdo al tipo de variable que acompaña al objeto.

Modo de uso: (suponiendo que se ha colocado la cláusula using namespace std)

```
cin >> expresión;
```

Como se observa, aquí también se ha sobrecargado un operador, en este caso el operador de bits **>>**, que en adelante será nombrado como "operador de extracción de

flujo". De igual manera, como en cout, la operación devolverá una referencia al objeto cin, por lo que la operación se podrá concatenar de la siguiente manera:

cin >> expresión1 >> expresión2 >> expresión3...;

Ejemplo:

```
int a;
double f;
cin >> a >> f;
```

Al ejecutar la orden, el sistema detendrá el programa para que el usuario pueda colocar los datos en el buffer de entrada, luego de presionar la tecla **ENTER [↵]** el programa convertirá los caracteres del flujo de entrada en la representación binaria correspondiente al tipo de dato de la variable y lo asignará a ella. Los caracteres ingresados deberán corresponder con el tipo de la variable de lo contrario el proceso se detendrá, asignando a la variable lo que se haya podido convertir hasta ese momento. El proceso termina satisfactoriamente cuando se encuentre un separador: espacio en blanco, cambio de línea o tabulador.

Como hemos indicado, el objeto cin devuelve una referencia al mismo objeto, por esta razón no podemos contar con un valor de respuesta que permita saber si la operación se realizará con éxito o no.

ENTRADA Y SALIDA DE CARACTERES

La entrada y salida de caracteres desde C++ se puede realizar, de igual manera que con los números, con los objetos cin y cout empleando una variable una variable de tipo char, sin embargo existen algunos métodos que pueden ser muy útiles dado el caso. A continuación describiremos algunos de éstos:

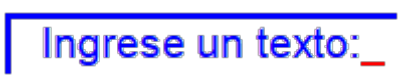
Método *cin.get()*:

El método toma un caracter del buffer de entrada y lo entrega al programa. El siguiente ejemplo muestra su efecto:

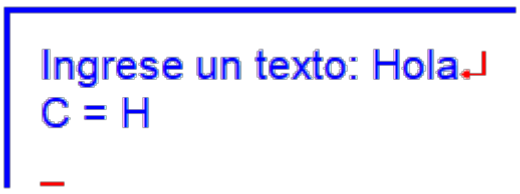
```
char  c = 'A';

cout << "Ingrese un texto: ";
c = cin.get();
cout << "C = " << c << endl;
```

Al ejecutarlo, primero se verá el mensaje en la ventana y el programa se detendrá, como se ve a continuación:



Luego de ingresar el texto y presionar la tecla ENTER [↵] se verá lo siguiente:

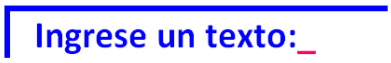


Existe una gran diferencia entre leer un carácter con el objeto cin que con el método get. La diferencia consiste en que con cin los separadores se ignoran si se encuentran antes del dato a leer, mientras que con get se lee el carácter inmediato sea cual fuera. El siguiente ejemplo muestra su efecto:

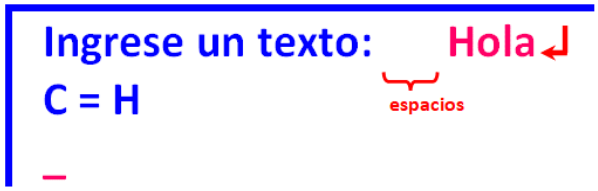
Con cin		Con get
`char c = 'A';`		`char c = 'A';`

`cout << "Ingrese un texto: ";`
`cin >> c;`
`cout << "C = " << c << endl;`
	Vs.	`cout << "Ingrese un texto: ";`
		`c = cin.get();`
		`cout << "C = " << c << endl;`

Al ejecutarlo la versión con cin, primero se verá el mensaje en la ventana y el programa se detendrá, como se ve a continuación:

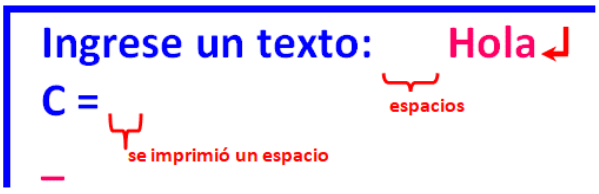


Luego de ingresar el texto colocando espacios antes de la palabra y presionar la tecla ENTER [↵] se verá lo siguiente:



Observe que a pesar de los espacios se lee la letra H.

Al ejecutarlo la versión con get, primero se verá el mensaje en la ventana, luego de ingresar el texto colocando espacios antes de la palabra y presionar la tecla ENTER [↵] se verá lo siguiente:



El método get leyó el primer espacio en blanco.

Método *cout.put(c)*:

El método toma el caracter contenido en la variable "**c**" y lo envía al medio de salida. El siguiente ejemplo muestra su uso:

```
char  c =  'A';

cout << "Ingrese un texto: ";
c = cin.get();
cout << "C = "
cout.put(c);
cout << endl;
```

Al ejecutarlo se mostrará de manera idéntica al resultado del programa anterior.

Método *cin.unget()*:

El método envía al buffer de entrada el último caracter extraído por el método ***cin.get()***.

Los ejemplos siguientes mostrarán el efecto de este método:

```
// Ejemplo 1, sin usar cin.unget()
char  a,  b,  c;

cout << "Ingrese un texto: ";
a = cin.get();
b = cin.get();
c = cin.get();
cout << "A = " << a << endl;
cout << "B = " << b << endl;
cout << "C = " << c << endl;
```

Al ejecutarlo, se obtendrá lo siguiente:

```
Ingrese un texto: Hola↵
A = H
B = o
C = l
_
```

```
// Ejemplo 2, usando cin.unget()
char  a,  b,  c;

cout << "Ingrese un texto: ";
a = cin.get();
cin.unget();                    // ⇐
b = cin.get();
c = cin.get();
cout << "A = " << a << endl;
cout << "B = " << b << endl;
cout << "C = " << c << endl;
```

Al ejecutarlo, se obtendrá lo siguiente:

```
Ingrese un texto: Hola↵
A = H
B = H
C = o
_
```

Método cin.peek():

El método obtiene una copia del caracter del buffer de entrada, sin extraerlo y lo entrega al programa.

En el ejemplo siguiente se podrá ver el efecto de este método:

```
char  a,  b,  c  d;

cout << "Ingrese un texto: ";
d = cin.peek();

a  =  in.get();
b  =  cin.get();
c  =  cin.get();
cout << "A = " << a << endl;
cout << "B = " << b << endl;
cout << "C = " << c << endl;
cout << "D = " << d << endl;
```

La ejecución de este código mostrará lo siguiente:

```
Ingrese·un·texto:·Hola↵
A·=·H
B·=·o
C·=·l
D·=·H
_
```

Método cin.putback(c):

El método es similar a cin.unget() con la diferencia que el usuario puede enviar al buffer de entrada el caracter que desee.

En el ejemplo siguiente se podrá ver su efecto:

```
char  a,  b,  c  d;
cout << "Ingrese un texto: ";

a  =  cin.get();
cin.putback('M');

b  =  cin.get();
c  =  cin.get();
d  =  cin.get();
cout << "A = " << a << endl;
cout << "B = " << b << endl;
cout << "C = " << c << endl;
cout << "D = " << d << endl;
```

La ejecución de este código mostrará lo siguiente:

```
Ingrese un texto: Hola↵
A = H
B = M
C = o
D = l
_
```

Función ws:

La función permite extraer del buffer de entrada todos los separadores (espacios en blanco, tabuladores y cambios de línea) que vaya encontrando hasta encontrar un caracter diferente.

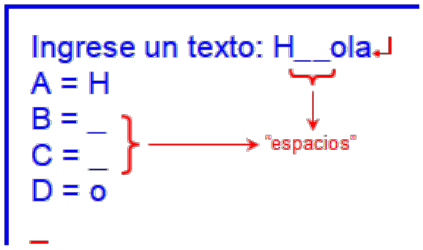
Observe los siguientes códigos:

```
// Ejemplo 1, sin usar ws
char  a,  b,  c  d;

cout << "Ingrese un texto: ";

a  =  cin.get();
b  =  cin.get();
c  =  cin.get();
d = cin.get();
cout << "A = " << a << endl;
cout << "B = " << b << endl;
cout << "C = " << c << endl;
cout << "D = " << d << endl;
```

El resultado será el siguiente:




```
// Ejemplo 2, usando ws
char  a,  b,  c  d;

cout << "Ingrese un texto: ";

a  =  cin.get();
cin << ws;
b  =  cin.get();
c  =  cin.get();
d  =  cin.get();
cout << "A = " << a << endl;
cout << "B = " << b << endl;
cout << "C = " << c << endl;
cout << "D = " << d << endl;
```

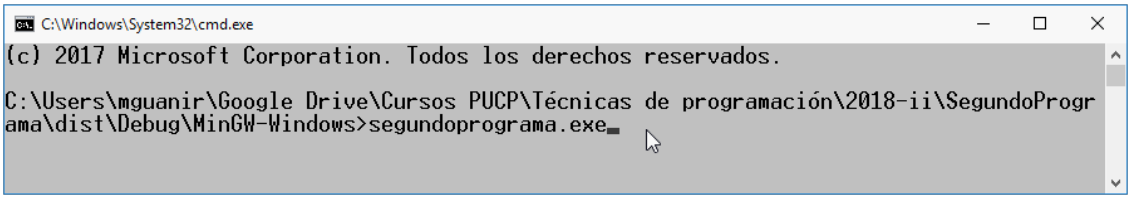
El resultado será el siguiente:

```
Ingrese un texto: H__ola↵
A = H
B = o
C = l
D = a
_
```

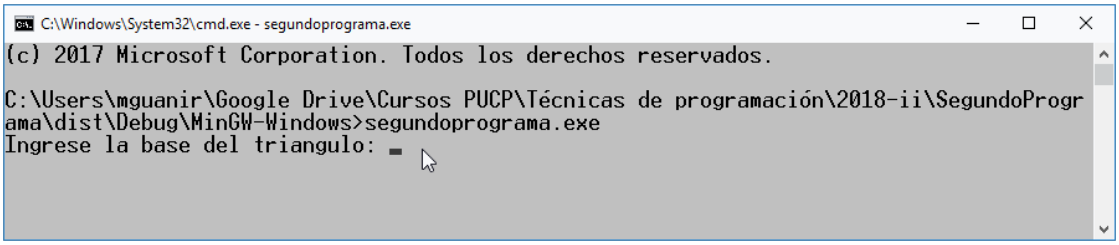
Re direccionamiento de la entrada y salida de datos

Esta es una propiedad del sistema operativo, y se aplica sobre cualquier programa ejecutable (programas con extensión .exe). Esto quiere decir que no depende ni del lenguaje de programación ni del entorno de desarrollo que estemos empleando.
Por lo general, el ingreso de datos se realiza a través del teclado y la salida a través de la pantalla del computador, pues bien, esta propiedad hará que, como su nombre lo indica, se pueda direccionar la entrada a otro medio, por ejemplo un archivo de textos, o el medio de salida, por ejemplo otro archivo de textos o la impresora.

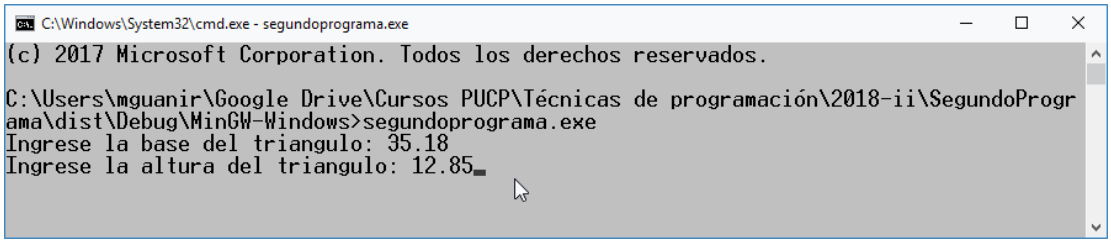
Para realizar esta forma de ejecutar un programa primero debe ubicar el programa ejecutable, para esto debemos seguir los pasos indicados en la "Guía de creación, ejecución y depuración de un programa en C/C++". Como se ve en la guía, para ejecutar el programa solo debemos colocar el nombre del programa en la línea de comandos y al presionar la tecla ENTER [↵] el programa se ejecutará, si el programa requiere que se le ingresen datos, estos serán ingresados por medio del teclado, y los resultados se verán impresos en la pantalla. A continuación mostramos este proceso:



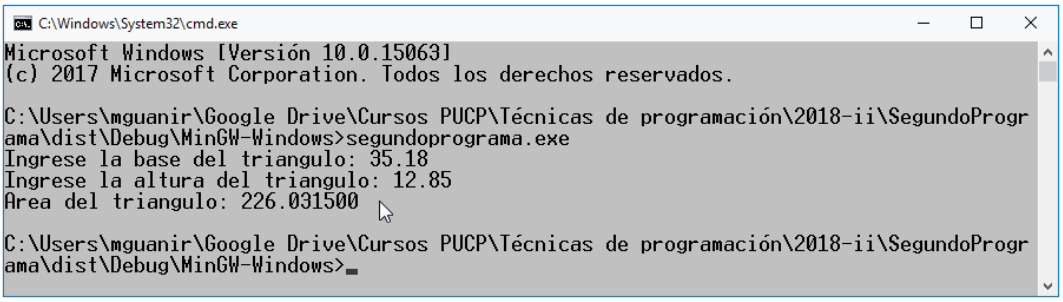
Al presionar la tecla ENTER [↵] el programa nos pedirá los valores como se indica a continuación:



Luego de ingresar los valores:

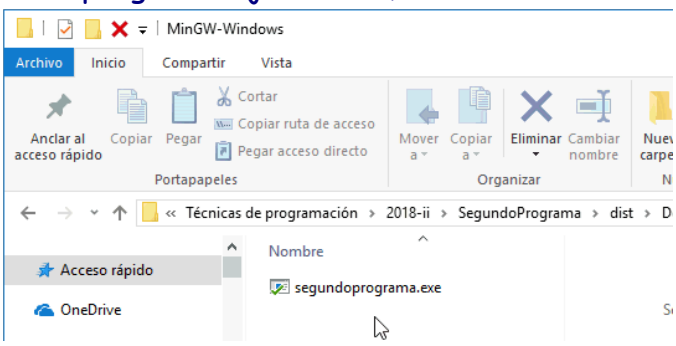


Se mostrarán los resultados:

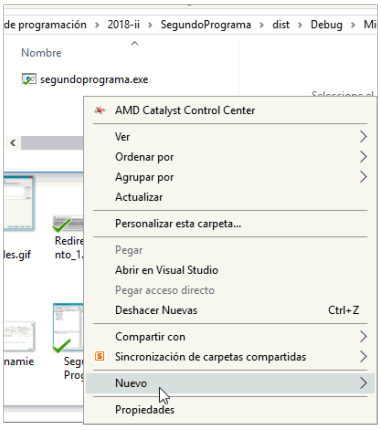


Como se ha visto, se ha ingresado la base y altura por medio del teclado y el resultado salió impreso en la pantalla.

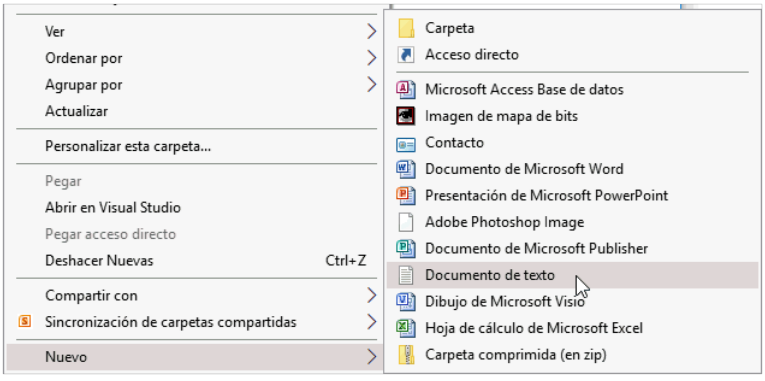
Ahora re direccionaremos primero la entrada de datos, de modo que los datos sean tomados desde un archivo de textos. Para esto sitúese en el explorador de archivos en la carpeta que contiene el programa ejecutable, como se muestra a continuación:



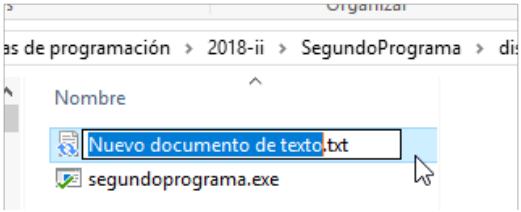
Allí presione el botón derecho del mouse, se desplegará un menú y allí seleccione la opción "Nuevo" como se ve a continuación:



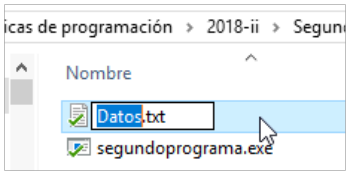
En el nuevo menú que se despliega como se muestra y allí seleccione la opción "Documento de texto":



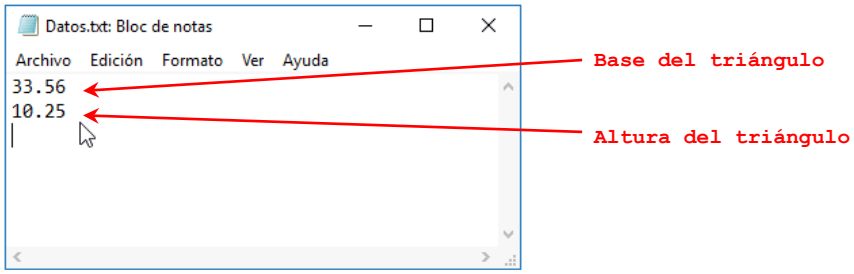
Esto creará un archivo de textos en la carpeta donde está su archivo ejecutable como se ve a continuación:



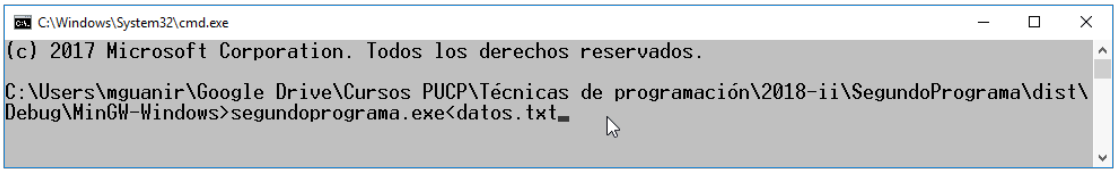
Cámbiele el nombre, por ejemplo póngale "Datos":



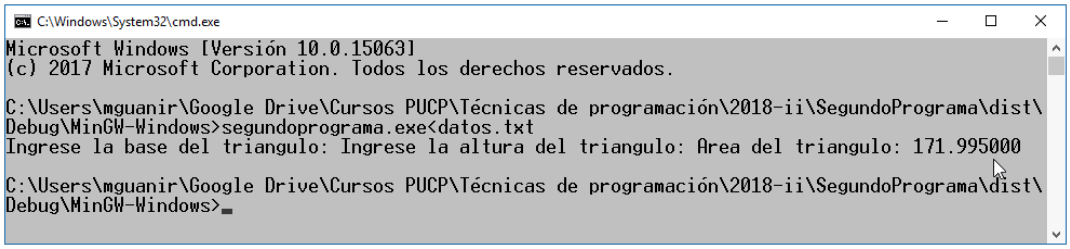
Luego abra el archivo, y allí coloque los datos para la base y altura del triángulo, los datos deben estar en el mismo orden en que ingresó los datos desde la entrada estándar, como se ve a continuación:



Guarde los cambios y luego cierre el archivo.
Luego diríjase nuevamente a la ventana donde se ejecuta el programa, coloque el nombre del programa ejecutable y luego agregue el re direccionamiento como se muestra a continuación:



Como ve, luego del nombre del programa debe colocar el signo < (menor que) seguido del nombre del archivo de textos con los datos. El signo < debe tomarlo como si fuera una flecha que indica la dirección del flujo de datos (programa ← datos) esto le indica al sistema operativo que los datos ingresarán a través del archivo de textos en lugar del teclado. Al presionar la tecla ENTER [↵], el programa se ejecutará y mostrará lo siguiente:



Lo que primero debe fijarse es que, aunque la salida es un poco peculiar, la respuesta final del programa muestra el valor del área del triángulo de manera correcta para los datos colocados en el archivo.

La particularidad de lo mostrado se debe a que para el programa es transparente la forma cómo se ingresan los datos, simplemente ejecutará el programa de la misma manera sea cual fuera el medio de entrada.

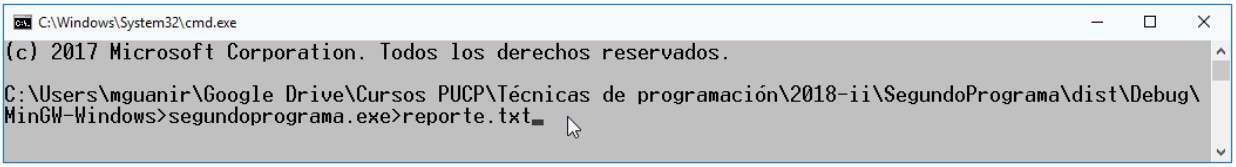
Recordemos el código del programa que estamos ejecutando:

```
9  #include <iostream>
10 #include <iomanip>
11 using namespace std;
12
13 int main(int argc, char** argv) {
14     double base, altura, area;
15     cout << "Ingrese la base del triangulo: ";
16     cin >> base;
17
18     cout << "Ingrese la altura del triangulo: ";
19     cin >> altura;
20
21     area = base*altura/2;
22     cout << "Area del triangulo: "<<area<<endl;
23     return 0;
24 }
```

Cuando se ejecute la línea 15 (cout…) el programa envía al medio de salida, la pantalla del computador, el mensaje para que se ingrese la base del triángulo. Recuerde que se ha re direccionado la entrada, no la salida. Fíjese que la instrucción no envía el cambio de línea (endl). Luego, se ejecuta la línea 16 (cin…), en donde se ingresará el valor para la base, aquí el valor se toma del archivo, por lo que el programa no se detiene e inmediatamente se ejecutan las líneas 18 y 19, de la misma manera que las dos anteriores por lo que en la salida los mensajes se ven uno a continuación del otro. Finalmente se calcula el área y se muestra los resultados, por eso es que los resultados salen uno a continuación del otro.

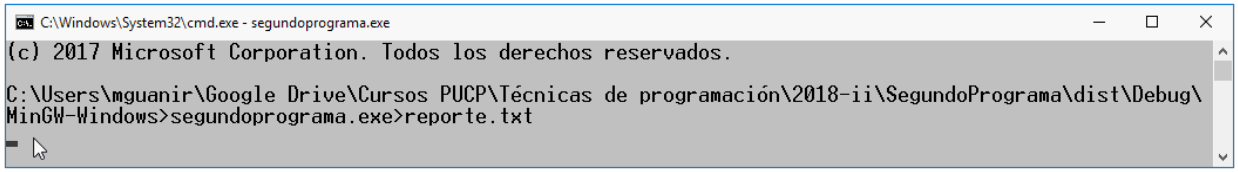
Se recomienda que cuando se planee que un programa será ejecutado cambiando el medio de entrada, no se coloquen mensajes pidiendo los datos, para que sólo se vea el resultado final.

Ahora re direccionaremos la salida, para esto coloque en la línea de comandos lo siguiente:

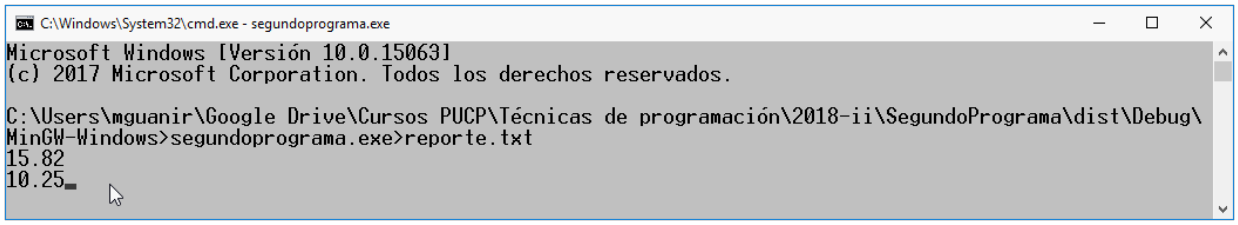


En este caso se usará el caracter > (mayor que) seguido del nombre de un archivo, que también debe ser interpretado como una flecha que indica el flujo de los datos (programa → resultados), esto le indica al sistema operativo que los resultados se enviarán al archivo cuyo nombre colocamos en la línea de comandos. Debe tener cuidado que no exista un archivo con ese nombre en la carpeta donde está su programa ejecutable porque cuando se corra el programa, el archivo será borrado y reemplazado por uno nuevo con los datos impresos por el programa.

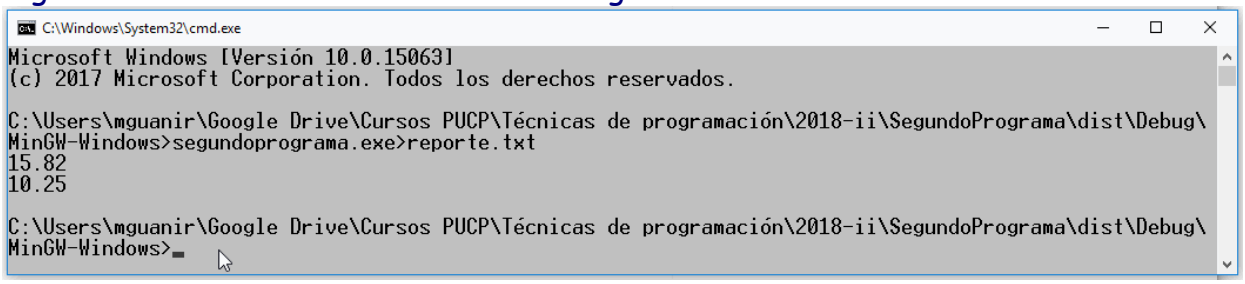
Al presionar la tecla ENTER se ejecutará el programa y se verá lo siguiente:



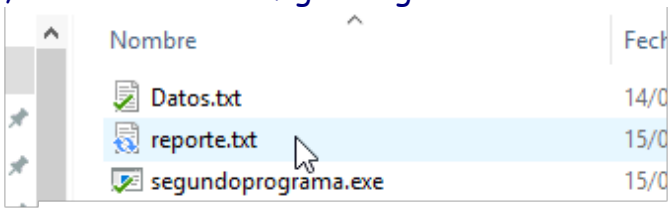
Solamente el cursor (_) pasó a la siguiente línea, esto se debe a que al re direccionar la salida de datos todo lo que se imprima será enviado al archivo de textos, por eso no vemos el mensaje con el pedido de ingreso de la base, tampoco veremos el mensaje con el pedido de ingreso de la altura, sin embargo en estos instantes el programa se ha detenido en la línea 16 (cin…) esperando que usted ingrese el valor para la base. Entonces ingrese la base y luego la altura como se muestra a continuación:



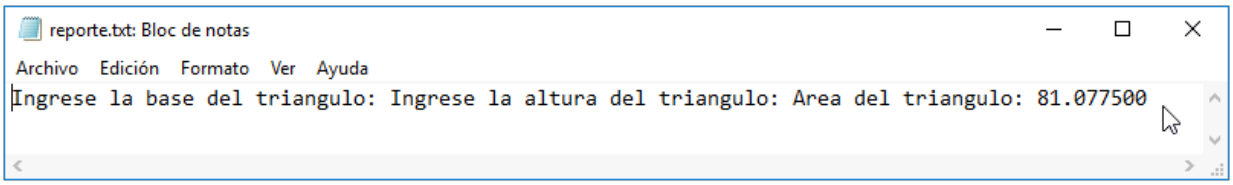
Luego de escribir la altura usted verá lo siguiente:



No se ve el resultado porque se envió al archivo reporte.txt. Para verificar esto, diríjase a la carpeta donde está el programa ejecutable y observe que ha aparecido el archivo en la carpeta, como se ve en la figura siguiente:

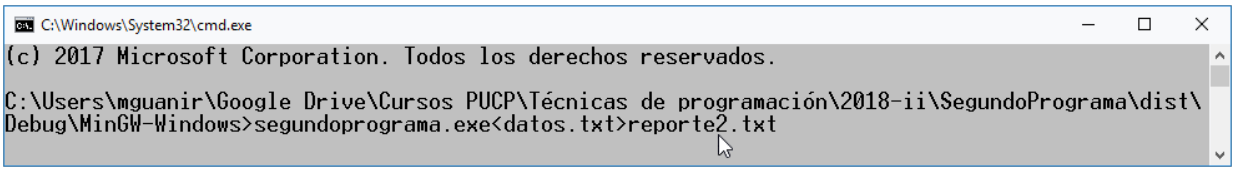


Al abrirlo verá lo siguiente:

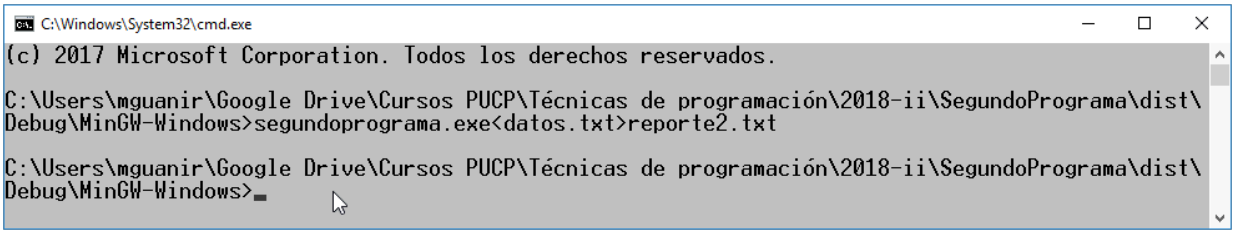


Como ve, el resultado es similar al que apareció en la anterior ejecución del programa pero con la diferencia que esta información está almacenada en un archivo de texto.

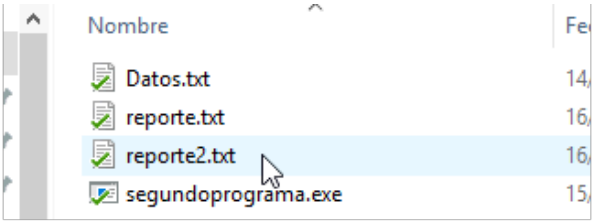
Finalmente re direccionaremos simultáneamente ambos medios, entonces, coloque en la línea de comandos lo siguiente:



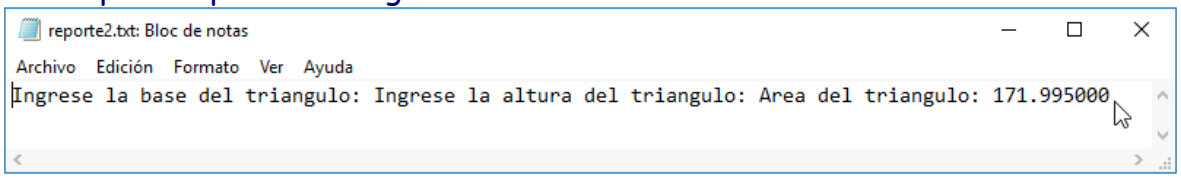
El presionar la tecla ENTER verá lo siguiente:



El programa se ejecutó tomando los datos desde el archivo "Datos.txt" y enviando los resultados al archivo "reporte2.txt" por eso es que parece que no se ejecutó el programa, pero si va nuevamente al explorador de archivos verá que se creó el nuevo archivo, como se ve a continuación:



Al abrirlo podrá apreciar lo siguiente:



El re direccionamiento de los medios de datos es una buena forma de probar los programas que normalmente se ejecutarán desde los medios de entrada y salida estándar, esto para que no tengamos que escribir los datos cada vez que se prueba el programa y para tener un registro de las ejecuciones del programa para poder compararlas.

Está forma de ejecutar un programa es muy práctica, sin embargo solo permite que los datos se ingresen por un único archivo y la salida también se envía a un único archivo. Si se necesitan leer los datos desde varios archivos o los resultados se deben enviar a varios archivos de texto este método no funciona. Por lo tanto debemos estudiar otra forma de escribir nuestros programas y esto se realizará empleando funciones que están adaptadas para este fin. A continuación revisaremos estas funciones.

Manejo de archivos de texto en el Lenguaje C++

Un archivo de textos es una colección de caracteres almacenados en la memoria secundaria del computador. La idea de emplear archivos de textos es poder almacenar información que podrá ser empleada cada vez que se ejecute el programa o por diferentes programas, esto significa que podemos almacenar en un archivo de textos

los datos que vamos a introducir al programa y hacer que el programa los tome de allí sin necesidad que los tengamos que digitar cada vez que queremos ejecutar el programa. La ventaja de emplear archivos de textos es que la consola del computador ha sido diseñada de manera similar a la de un archivo de textos, de modo que introducir datos a un programa desde la consola o desde un archivo de textos es casi lo mismo, las instrucciones que utilizaremos en uno u otro caso serán muy parecidas.

Un archivo de textos se puede crear mediante instrucciones de programa, pero la forma más fácil de hacerlo es empleando un procesador de palabras simple como el "Block de notas" que proporciona el sistema operativo Windows.

Variable de archivo

Para poder utilizar un archivo de textos se requiere relacionar de manera lógica el archivo con el programa. Cuando ingresamos valores por teclado, la conexión se realiza mediante el objeto ***cin,*** mientras que para mostrar un resultado por la pantalla, la conexión se hace por medio del objeto cout. Cuando se trabaja con archivos de textos, como se indicó anteriormente, no se emplea un solo medio de salida y un solo medio de entrada, la información se puede recibir desde varios archivos y enviar a otros tantos archivos. Por esta razón, el sistema no define un objeto para leer de un archivo y tampoco otro para mostrarlo porque no se puede adivinar con cuántos archivos habrá que conectarse simultáneamente en el mismo programa. Por esta razón, el C++ nos proporciona dos ***clases*** (dos tipos de datos) que nos permitan definir los objetos que creamos convenientes para desarrollar un programa. Esto es similar a definir variables para un programa, tenemos el tipo de dato ***int*** y el tipo de dato ***double*** con ellos podemos definir todas las variables que requeriremos.

En el C++ se ha implementado una biblioteca de funciones denominada ***fstream*** (***#include <fstream>***), donde se han definido dos ***clase ifstream*** y ***ofstream*** y con ellas podremos definir los objetos que creamos convenientes para conectarnos con los archivo que contengan los datos que requeriremos para elaborara el programa.

La clase ***ifstream*** permite definir variables de archivo para realizar operaciones de lectura en archivos de texto., mientras que ***ofstream*** permite definir variables de archivo para realizar operaciones de escritura en archivos de texto.

Constructores y destructores

Los constructores y destructores son métodos o funciones miembro definidos para una clase. Tanto los constructores como los destructores se ejecutan de manera automática. Los constructores para las clases ***ofstream*** e ***ifstream*** permiten abrir los archivos, mientras que los destructores permiten cerrarlos.

Definición, enlace y apertura de archivos

Antes de poder leer o escribir datos en un archivo, debemos realizar tres tareas: definir una variable de archivo, enlazarla a un archivo físico y abrir el archivo para que podamos manipularlo. En C++ estas tres tareas se pueden realizar en una sola

instrucción, haciendo uso del constructor de la clase o en dos instrucciones, haciendo uso de un método de la clase, esta operación se describe a continuación.

Para abrir un archivo y poder leer datos del archivo, la instrucción será la siguiente:

Empleando el constructor:

```
#include <fstream>
using namespace std;
...
ifstream arch ("archivo.txt", ios::in);
```

Donde:

- ***ifstream:*** es el nombre de la clase que permite enlazar una variable de archivo para leer los datos que se encuentren allí.
- ***arch:*** es la variable de archivo
- ***"archivo.txt"***: es el nombre del archivo al que se enlazará la variable ***arch***.
- ***ios::in***: indica el modo de apertura (input) que permite leer datos del archivo.

Para que la instrucción se ejecute correctamente el archivo debe existir en el disco.

Sin emplear el constructor:

```
#include <fstream>
using namespace std;
...
ifstream arch;
...
arch.open ("archivo.txt", ios::in);
...
```

Donde:

- ***open:*** es un método que realiza las mismas operaciones de apertura.

El resto tiene la misma función que en el anterior caso.
Para que la instrucción se ejecute correctamente también el archivo debe existir en el disco.

Para abrir un archivo y poder escribir datos en el archivo, la instrucción será la siguiente:

Empleando el constructor:

```
#include <fstream>
using namespace std;
...
ofstream arch ("archivo.txt", ios::out);
```

Donde:

- ***ofstream:*** es el nombre de la clase que permite enlazar una variable de archivo para escribir los datos allí.
- ***arch:*** es la variable de archivo
- ***"archivo.txt"***: es el nombre del archivo al que se enlazará la variable ***arch***.

- ***ios::out***: indica el modo de apertura (output) que permite escribir datos en el archivo.

Al ejecutar esta instrucción, si el archivo existe lo borrará y si no existe lo creará.

Sin emplear el constructor:

```
#include <fstream>
using namespace std;
…
ofstream arch;
…
arch.open ("archivo.txt", ios::out);
…
```

Donde:

- ***open:*** es un método que realiza las mismas operaciones de apertura.

El resto tiene la misma función que en el anterior caso.

Verificación correcta de la apertura de un archivo

Una de las cosas que debemos hacer cuando abrimos un archivo es verificar que se abrió correctamente. De no hacerlo, el programa no funcionará correctamente terminando por interrumpirse abruptamente. Lo crítico de esto es que la interrupción del programa no se va a producir en la línea donde se ejecuta el constructor o el método ***open*** sino en cualquier otra línea del programa, haciendo muy difícil depurar el programa.

Por esta razón estamos obligados a verificar explícitamente la correcta apertura del archivo a fin de poder detectar un posible error y corregirlo.

Afortunadamente la forma de verificación es muy simple ya que existe un método denominado ***is_open()*** para este fin, si el sistema no puede abrir el archivo, el método ***is_open()*** devolverá el valor ***true*** si el archivo se abrió correctamente y ***false*** en caso contrario. Por lo tanto la verificación debe consistir en lo siguiente:

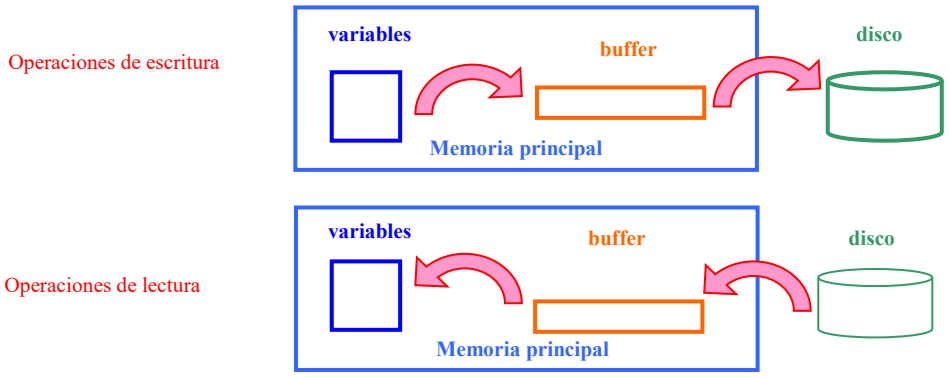
```
#include <fstream>
using namespace std;
…
ifstream arch ("archivo.txt", ios::in);
if (not arch.is_open()){
   cout << "ERROR: no se pudo abrir el archivo archivo.txt" << endl;
   exit(1);
}
…
```

Observe que la verificación debe hacerse inmediatamente después de la apertura. El uso de la función "***exit***" hará que el programa se interrumpa, entienda que esta interrupción debe hacerse porque si no se abrió bien el archivo no se puede continuar con la ejecución del programa.

Cierre de un archivo

Inmediatamente se termine de realizar operaciones con un archivo, éste debe cerrarse. De no hacerse se podría perder información importante del archivo. La razón de esto es que cuando se realizan operaciones de entrada o salida de un archivo, la operación no desplaza la información directamente desde el archivo hacia las variables o viceversa, lo que sucede es que en el momento de abrir el archivo para leer de él, parte de la información contenida en el archivo pasa a un espacio temporal de memoria denominado 'buffer'. Cuando se hace una operación de lectura se toman los datos del buffer y no del archivo, cuando toda la información del buffer se ha leído, nuevamente se le carga otra parte del archivo y el proceso continúa. Cuando se escribe en un archivo, la información se envía al buffer, cuando el buffer se llena, se descarga la información que contiene al archivo. Si el programa se interrumpiera abruptamente antes de cerrar el archivo, el contenido del buffer no pasaría al archivo, y por lo tanto se perdería la información que contiene. Esto es un suceso crítico en operaciones de escritura.

El esquema siguiente muestra este proceso:



Esta forma de manejar los datos se da para evitar demasiadas operaciones directamente sobre el disco, de no hacerlo así se perdería mucho tiempo en el proceso ya que el disco es un dispositivo mucho más lento que la memoria principal del computador, por otro lado se evita un desgaste innecesario del disco. Luego de cerrar un archivo, se puede volver a abrir empleando nuevamente el método ***open***.

Afortunadamente el lenguaje C++ define un ***destructor*** para las clases que definen archivos, este destructor, como ya lo dijimos, se ejecuta automáticamente cuando se sale del ámbito de la variable de archivo, y la tarea que realiza este destructor es la de cerrar el archivo, por lo que no necesitaremos ejecutar algún método. Sin embargo, existe el método que puede cerrar el archivo en el momento que se desee y se llama ***close*** (***arch.close()***).

Entrada y Salida de datos desde archivos de texto

Una vez abierto el archivo, se podrán realizar operaciones de escritura o de lectura, según como fue abierto, y estas operaciones se tornan muy sencillas ya que para hacerlo se emplean elementos muy similares a las que se emplean con la entrada estándar (cout << y cin >>). Estos elementos sólo tienen una variante, se debe colocar la

variable de archivo en lugar de los objetos cout y cin (arch << y arch >>), el resto de la sintaxis, así como la forma como trabaja es idéntico a la forma como se trabaja desde la entrada estándar.

Detección de errores en la entrada de datos

Durante la ejecución de un programa se pueden generar errores al tratar de ingresar datos ya sea desde el teclado como desde un archivo. Estos errores pueden ser detectados durante la ejecución del programa y así poder tomar las medidas que sean convenientes para poder continuar, sin inconvenientes, con la ejecución del programa.

Un error en la entrada de datos se produce por muchas razones, pero son dos las más importantes, se produce un error cuando al leer un dato numérico se introduce una secuencia de caracteres que no corresponden con un número. Por ejemplo, en el siguiente código se quiere leer de un archivo de textos dos valores enteros y luego imprimirlos:

```
#include <iostream>
#include <iomanip>
#include <fstream>
using namespace std;
…
ifstream arch ("archivo.txt", ios::in);
if (not arch.is_open()){
   cout << "ERROR: no se pudo abrir el archivo archivo.txt" << endl;
   exit(1);
}
int a, b, c;
arch >> a >> b;
cout << "A = " << a << endl;
cout << "B = " << b << endl;
…
cin >> c;
…
```

Si en el archivo en lugar de encontrar dos números se colocan valores como: 23 #ER, el programa leerá el valor 23 y lo asignará a la variable "a" pero no podrá leer los caracteres #ER por lo que el sistema levantará una bandera de error y asignará cero a la variable "b". A partir de allí, mientras la bandera esté alzada, cualquier operación de lectura que se haga a continuación no asignará valores (tampoco les pondrá cero). Debe tener en cuenta que bajo esta condición el programa no se detendrá.

Un error también se produce cuando, al leer los datos de un archivo, no se puede verificar que los datos ya se terminaron y se intenta leer luego de llegar al final del archivo. El siguiente ejemplo ilustra este problema:

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

```
ifstream arch ("archivo.txt", ios::in);
if (not arch.is_open()){
   cout << "ERROR: no se pudo abrir el archivo archivo.txt" << endl;
   exit(1);
}
int dato, suma=0;
while (true) {
      arch >> dato;
      if (dato < 0) break;
      suma += a;
}
cout << "Suma = " << suma << endl;
…
```

Si el archivo no tuviera valores negativos, luego de leer todos los datos del archivo, al querer seguir leyendo por no encontrar la condición de fin de datos, el sistema levantará la bandera de error por el fin del archivo, ya no se seguirá leyendo pero el programa no se detendrá y entrará en un ciclo sin fin.

Afortunadamente, el lenguaje C++ tiene herramientas para poder detectar estos errores y poder corregirlos. Los métodos ***fail()*** y ***eof()***, están diseñados para estos fines. El método ***fail()***, devuelve ***true*** si se ha producido un error en el ingreso de datos y ***false*** en caso contrario. El método ***eof()*** devuelve ***true*** si se ha producido un error en el ingreso de datos si se intentó leer luego del final del archivo y ***false*** en caso contrario.

El siguiente ejemplo muestra cómo leer datos de un archivo de textos sin colocar un dato centinela.

```
…
int dato, suma=0;
while (true) {
      arch >> dato;
      if (arch.eof()) break;
      suma += a;
}
cout << "Suma = " << suma << endl;
…
```

El programa leerá los datos del archivo y cuando se intente leer un dato luego del fin del archivo se levantará la bandera y el método *eof()* devolverá true por lo que la instrucción *if* ejecutará la orden *break*.

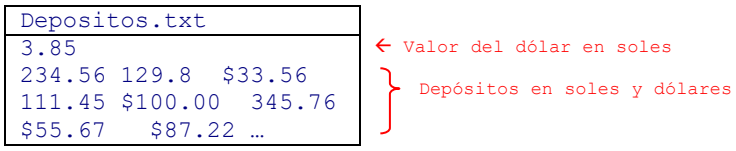
Cómo bajar la bandera de error

Cuando se levanta la bandera de error ya no se puede leer datos, sin embargo si se desea seguir leyendo entonces antes habrá que bajarla. El método que permite hacerlo es el método *clear()*.

El siguiente ejemplo permite apreciar la forma cómo se pueden emplear todos estos métodos.

"Se tiene un archivo de textos en los que se ha registrado los depósitos hechos en una cuenta bancaria, los depósitos pueden estar tanto en soles como en dólares. En la primera línea del archivo se encuentra el valor equivalente de un dólar en soles. En las siguientes líneas del archivo se encuentran los depósitos. La forma de saber en qué moneda está el depósito es porque los depósitos en dólares están precedidos por el signo de $ (p. e.: $23.45), mientras que a los depósitos en soles no le preceden algún carácter (p. e.: 23.45). No hay un patrón que defina el orden en que se realizan los depósitos. El programa debe determinar el monto total depositado. Debe mostrar este valor en soles y su equivalente en dólares."

Un ejemplo de este archivo de textos se muestra a continuación:



El programa que da solución a este problema se muestra a continuación:

```
/*
 * Proyecto: DetecionDeErrores
 * Archivo:  main.cpp
 * Autor:    J. Miguel Guanira E.
 *
 * Created on 27 de julio de 2022, 09:36 PM
 */
#include <iostream>
#include <fstream>
#include <iomanip>
using namespace std;

int main(int argc, char** argv) {
    ifstream arch("Depositos.txt",ios::in);
    if (not arch.is_open()){
        cout <<"ERROR: No se pudo abrir el archivo Depositos.txt"<<endl;
        exit(1);
    }
    double valorDelDolar, deposito, totalDepositado=0;
    arch >> valorDelDolar;  // Leemos el valor equivalente en soles
    while(true){
        arch >> deposito;   // Tratamos de leer el depósito
        if (arch.eof()) break;  // Si leemos luego del final del archivo, salimos
        if (arch.fail()){   // Si el depósito está precedido por un $ no lo
                            // podrá leer por lo que levantará la bandera de error.
            arch.clear();   // Bajamos la bandera para poder seguir leyendo.
            arch.get();     // Sacamos el caracter $ del archivo
            arch >> deposito;  // Leemos el depósito en dólares.
            deposito *= valorDelDolar; //Converimos el depósito en soles
        }
        totalDepositado += deposito;
    }
    cout.precision(2);
    cout << fixed;
    cout << "Total depositado en soles  : " << setw(10)<<totalDepositado<<endl;
    cout << "Total depositado en dolares: " << setw(10)
            <<totalDepositado*valorDelDolar << endl;
    return 0;
}
```

Debemos enfatizar que el método *fail()* detecta cualquier error de entrada de datos, mientras que al método *eof()* solo detecta el error del fin del archivo, por eso en el programa, primero verificamos el fin del archivo y luego el error por el caracter $.