Estructura de Datos y Algoritmos

Entrada y Salida en C++

1. Introducción

Las bibliotecas estándar de C++ proporcionan un amplio conjunto de capacidades de entrada/salida (E/S).

C++ utiliza E/S a prueba de tipos. Cada operación de E/S se realiza automáticamente en una forma sensible con respecto al tipo de datos.

2. Flujos

La E/S de C++ se da en flujos de bytes. Un flujo es simplemente una secuencia de bytes. En las operaciones de entrada, los bytes fluyen desde un dispositivo (por ejemplo un teclado, una unidad de disco) hacia la memoria principal. En operaciones de salida los bytes fluyen de la memoria principal hacia un dispositivo (por ejemplo una pantalla, una impresora, una unidad de disco).

La aplicación asocia significado a los bytes. Pueden representar caracteres ASCII, o cualquier otro tipo de información que pueda requerir una aplicación.

3. cin y cout

cin es el flujo de entrada estándar que normalmente es el teclado y cout es el flujo de salida estándar que por lo general es la pantalla.

4. Impresión de una linea de texto

```
#include<iostream>

int main()
{
        cout << “Hola, que tal\n”;
        return 0;
}
```

iostream es el archivo de encabezado del flujo de entrada/salida. Este archivo debe incluirse cuando se utilizen cin o cout.

En vez de \n podemos poner endl, como a continuación:

```
#include<iostream>

int main()
{
        cout << “Hola, que tal”;
        cout << endl;
```

```
        return 0;
}
```

5. Lectura desde teclado

```
int main()
{   int i1,i2,sum;
    cout << "Ingrese el 1er numero entero\n";
    cin >> i1;
    cout << "\nIngrese el 2do numero entero\n";
    cin >> i2;
    sum = i1+i2;
    cout << "\n La suma es ";
    cout << sum;
    cout << endl;

    return 0;
}
```

La instrucción cin >> i1; obtiene un valor desde el teclado. El usuario debe introducir un valor y luego enter. cin saltea los espacios en blanco, los tabuladores y el salto de linea.

endl envia a la salida un salto de linea.

6. Puesta en cascada de los operadores de inserción/extracción de flujo

Los operadores << y >> pueden utilizarse en forma de cascada, como por ejemplo en:

```
cout << "47 mas 53 es "<< (47 + 53) << endl;
```

se ejecutan como si hubieran sido escritas en la forma:

```
(((cout << "47 mas 53 es ") << (47 + 53)) << endl);
```

o sea que asocia de izquierda a derecha. Esto funciona porque

```
cout << a;
```

devuelve una referencia hacia su operando izquierdo, es decir cout. Por lo tanto

```
(cout << "47 mas 53 es ")
```

envia a la salida la cadena de caracteres especificada y devuelve una referencia a cout.

De igual forma:

cin >> a >> b;

realiza la entrada en a y devuelve una referencia a cin la cual realiza la entrada en b.

7. Salida de caracteres con la función miembro put, put en cascada

La función miembro put envia a la salida un caracter, por ejemplo:

cout.put('A');

lo cual despliega una A en pantalla. Las llamadas a put pueden ponerse en cascada como en:

cout.put('A').put('\n');

lo cual da salida a A seguida de un caracter de nueva linea. El operador punto (.) asocia de izquierda a derecha y la función miembro put devuelve una referencia al objeto mediante el que se realizo la llamada a put.

put también puede invocarse mediante una expresión de valor ASCII, como en cout.put(65) que también da salida a A.

por ejemplo:

cout.put('A').put('\n').put('B').put(65);

imprime:

A
BA

8. EOF en cin

El operador de extracción de flujo era >>. Cuando este encuentra el fin de archivo en una entrada, devuelve cero (false), de lo contrario devuelve una referencia al objeto mediante el cual es llamado (cin).

Por ejemplo, podemos escribir el programa:

```
int main()
{
    int grado, mayor_grado = -1;
    cout << "Introduzca calificacion (CTRL Z para terminar)\n";
    while (cin >> grado)
    { if (grado > mayor_grado)
        mayor_grado=grado;
    cout << "\nIntroduzca calificacion (CTRL Z para terminar)\n";
    }
    cout << "\n"Mayor calificacion es : "<< mayor_grado << endl;
    return 0;
}
```

cuya ejecución es:

```
Introduzca calificacion (CTRL Z para terminar)
15
Introduzca calificacion (CTRL Z para terminar)
25
Introduzca calificacion (CTRL Z para terminar)
78
Introduzca calificacion (CTRL Z para terminar)
65
Introduzca calificacion (CTRL Z para terminar)
CTRL-Z Mayor calificacion es : 78 Presione una tecla para continuar . . .
```

9. Entrada/salida de strings, función setw

9.1. En la entrada

Es posible asignar una cadena a un arreglo de caracteres utilizando cin y setw como sigue

```
int main()
{       char palabra[20];

        cin >> setw(20) >> palabra;
        return 0;
}
```

La instrucción previa lee los caracteres hasta que encuentre un espacio, tabulación o salto de linea o hasta que se lean 19 caracteres.

La posición número 20 se reserva para el caracter de fin de string que es '\0'. Si se ingresan menos de 19 caracteres y espacio (o tabulación o fin de linea) se coloca el '\0' despues del ultimo caracter leido.

setw(i) indica que se consideren i caracteres de la entrada. Si se ingresan mas de i caracteres, los restantes caracteres quedan en el flujo de entrada (serán ingresados en la proxima instrucción de entrada).

Usamos setw para la entrada de strings de caracteres.

Se podría dar el caso de que quisieramos "consumir" toda la entrada en cada operación de entrada de un programa. Por ejemplo, si tuviera el siguiente trozo de programa:

```
char s1[10], s2[5];
cout << Ïngrese 1er string \n";
cin >> setw(10) >> s1;
cout << Ïngrese 2do string \n";
cin >> setw(5) << s2;
```

si ingresara más de 10 caracteres, o si ingresara dos palabras separadas por blanco, tabulador o nueva linea, leeria en ambos strings, por ejemplo, si ingreso

hola_que_tal

se ingresaria hola_que_ en el 1er string y tal en el segundo.
Si ingresara

hola que tal

ingresaría hola en el primer string y que en el segundo.

Para consumir toda la entrada en cada instruccion de entrada tendria que agregar por ejemplo una instrucción while que consumiera toda la entrada hasta el nueva linea. La instrucción cin saltea los blancos, nueva linea y tabuladores que encuentre en la entrada. Estos quedan de la primera instrucción para la segunda. Si ingreso "hola" seguido de 3 caracteres blancos el segundo cin lee blancos y el nueva linea o sea no lee nada.

El programa para consumir la entrada restante luego de una lectura es:

```
int main()
{   char s1[10],s2[5],c;
    cout << "Ingrese 1er string \n";
    cin >> setw(10) >> s1;
    while ((c=getchar())! ='\n');
    cout << "Ingrese 2do string \n";
    cin >> setw(5) >> s2;
    cout << s1 <<  << s2 << "\n";
    system("PAUSE");
    return 0;
}
```

el while consumira la entrada hasta el nueva linea.

9.2. En la salida

Es posible imprimir un valor en un campo de un ancho de n caracteres utilizando $setw(n)$.

Ejemplo:

```
int main()
{       int i;

        i=876;
        cout << setw(5) << i;
        system("PAUSE");
        return 0;
}
```

Si la salida tiene menos de n posiciones de manera predeterminada queda alineada a la derecha en el campo. En el ejemplo anterior se imprime 876 en un campo de ancho 5 justificado a la derecha.

Si tiene más de n posiciones el campo se extiende para acomodar el valor completo.

Ejemplo:

```
int main()
{
        cout << setw(2) << "hola que tal";
        system("PAUSE");
        return 0;
}
```

imprime "hola que tal"justificado a la izquierda.

Observar que podemos utilizar setw en la salida, para la impresión de distintos tipos de datos (enteros, punto flotante, caracteres, strings). En el caso de punto flotante necesitamos setear algunas banderas (lo veremos más abajo).

Para utilizar setw debemos incluir el archivo <iomanip>.

10. Entrada de strings, funciones cin.get() y cin.getline

10.1. cin.get() (sin argumentos)

La función cin.get() lee un caracter desde el teclado. a=cin.get() almacena el caracter leido en la variable a. Se pueden introducir blancos, tabuladores y nueva linea en a (idem to getchar).

Consideremos el siguiente programa:

```
int main()
{
    char a,b,c;
    a=cin.get();
    b=cin.get();
    c=cin.get();
    cout << a << b << c;
    system("PAUSE");
    return 0;
}
```

Si ingreso una palabra de largo mayor o igual a tres, se cargarán caracteres en a, b y c. Si hubieran más instrucciones de lectura a continuación leerian los caracteres restantes (si el largo de la palabra ingresada es mayor que tres).

Veamos otro programa:

```
int main()
{
    char a,b,c;
    a=cin.get();
    cout << a;
    b=cin.get();
    cout << b;
    c=cin.get();
    cout << c;
    system("PAUSE");
    return 0;
}
```

Lo que tenemos que observar, es que si ingresamos tres caracteres o más desde un principio, estos se leen en los sucesivos get y se imprimen todos juntos luego, mientras que si ingresamos de a un caracter se solapan la lectura y la impresión. Por ejemplo:

Entrada : hola
*Salida : hol*Presione una tecla para continuar . . .

lee e imprime todo junto, mientras que si ingreso una a (y un nueva linea que ingresa la a) se imprimira la a y el nueva linea. Luego del nueva linea puedo ingresar b y nueva linea, se imprimira la b y el mensaje del pause.

10.2. cin.get(a,b,c) (con 3 argumentos)

Hay otra versión de la función get que toma tres argumentos: un arreglo de caracteres, un límite de tamaño y un delimitador (con un valor predeterminado de '\n'). Esta versión lee caracteres desde el flujo de entrada, lee hasta uno menos que el número de caracteres especificado y término o término antes si lee el delimitador. Esta función inserta el caracter nulo '\0' al final del arreglo de caracteres. El delimitador no se coloca en el arreglo pero permanece en el flujo de entrada y será el siguiente caracter que se lea en una instrucción de entrada.

Al igual que el get sin argumentos ingresa espacios en blanco, nueva linea y tabulador (en el caso de que estos no sean delimitadores).

Por ejemplo:

```
int main()
{
    char c,arr[10];
    cin.get(arr,5,'\n');
    cout << arr << endl;
    c=getchar();
    cout << "Imprimo el delimitador: ";
    cout << c << endl;
```

```
    system("PAUSE");
    return 0;
}
```

lee los cuatro caracteres desde el teclado en arr y el delimitador que es nueva linea queda en la entrada. Este es leido por el getchar (podriamos haber puesto un cin.get()) y se imprime:

"Imprimo el delimitador: "

y luego un salto de linea antes del mensaje del pause.

Si en vez de '\n' en el get coloco ':' como delimitador y tengo un ':' en la entrada antes de leer 4 caracteres, termina la lectura (o sea quedan en arr los primeros 3 o menos caracteres) y el ':' queda en la entrada. luego imprime los caracteres ingresados, un nueva linea, la frase

"Imprimo el delimitador:"

y el ':'.

Si en cambio tuviera:

```
int main()
{
    char arr1[10],arr2[10];
    cin.get(arr1,10,'\n');
    cout << arr1;
    cout << "\n";
    cin.get(arr2,10,'\n');
    cout << arr2;
    system("PAUSE");
    return 0;
}
```

e ingreso "hola que tal" y nueva linea, el primer get leerá 9 caracteres ("hola que ") y "tal" y nueva linea serán leidos por el segundo get. Se imprimen 1ro el "hola que ", un nueva linea y luego "tal" y el mensaje del pause, sin saltar linea despues del tal (el nueva linea quedó en la entrada).

Si ingresara menos de 9 caracteres (por ejemplo "hola") y nueva linea, el nueva linea queda en la entrada y es leido por el segundo get, el cual termina sin leer nada mas que ese nueva linea que sigue quedando en la entrada.

10.3. cin.getline(a,b,c) (con 3 argumentos)

Esta instrucción toma tres argumentos: un arreglo de caracteres en el que se almacena la linea de texto, una longitud y un caracter delimitador. La diferencia con cin.get con 3 argumentos es que el caracter delimitador es consumido por el getline (lo quita del flujo de entrada).

Por ejemplo:

char sentencia[20];

cin.getline(sentencia,20,'\n');

declara el arreglo sentencia de 20 caracteres, lee del teclado una linea de texto y la carga en el arreglo. La función termina la lectura de los caracteres cuando encuentra el caracter '\n'. Si se ingresan más de 19 caracteres se trunca. Se asume el tercer argumento es '\n' por defecto por lo que la llamada a la función anterior se podría haber escrito como

cin.getline(sentencia,20);

getline permite ingresar caracteres blancos, nueva linea y tabuladores dentro del texto.

Otra diferencia con get es que si ingreso igual o más que la cantidad de caracteres indicada en la instrucción se setea un bit que hace que fallen las instrucciones getline que aparezcan luego. Para que las instrucciones siguientes no fallen se deben resetear los bits de estado (con cin.clear()) luego de la instrucción getline que dio el error, por ejemplo:

```
int main()
{
    char arr1[10],arr2[10];
    cin.getline(arr1,10,'\n');
    cin.clear();
    cin.getline(arr2,10,'\n');
    cout << arr1 << arr2;
    system("PAUSE");
    return 0;
}
```

resetea el bit y hace que la entrada que sobre del 1er getline sea leida por el segundo.

11. Funciones ignore, putback y peek

11.1. ignore

La función ignore() saltea un caracter de la entrada. ignore(i) saltea i caracteres de la entrada. Se utiliza del siguiente modo:

cin.ignore(i);

por ejemplo:

```
int main()
{
    char arr[10];
```

```
    cin.ignore(5);
    cin.get(arr,10,'\n');
    cout << arr;
    system("PAUSE");
    return 0;
}
```

Si introduzco en la entrada cinco letras a y diez letras b, va a imprimir nueve letras b.

11.2. putback

cin.putback(c) coloca el caracter obtenido previamente con un get del flujo de entrada de nuevo en dicho flujo. El caracter obtenido previamente con get debe ser c.

```
int main()
{
    char arr[10],c;
    c=cin.get();
    cin.putback(c);
    cin.get(arr,10,'\n');
    cout << arr << endl << c;
    system("PAUSE");
    return 0;
}
```

Puedo utilizar c=getchar() o cin >> c en vez de cin.get(). Si ingreso por ejemplo "holaↄ nueva linea, se imprimirá "hola"(el caracter leido en el get se coloca de nuevo en la entrada y se vuelve a leer en el segundo get), despues de hola se imprime el caracter que es "h".

11.3. peek

La función peek devuelve el siguiente caracter de un flujo de entrada sin eliminar el caracter del flujo.

```
int main()
{
    char arr[10],c;
    c=cin.peek();
    cin.get(arr,10,'\n');
    cout << arr << endl << c;
    system("PAUSE");
    return 0;
}
```

Si ingreso por ejemplo "hola", se imprimirá "hola"(el caracter leido en el peek se mantiene en la entrada y se lee en el get). Se imprime "hola"nueva linea y "h".

11.4. eof

cin.eof() es falso si no ha sucedido el fin de archivo y true cuando se ingresa el CTRL-z.

12. E/S sin formato mediante read, gcount y write

read y write da entrada o envía a la salida algún número de bytes desde o hacia un arreglo de caracteres que está en memoria.

read(var,n) lee n caracteres y los guarda en var. Por ejemplo:

```
char buffer[5];
cin.read(buffer,5);
cout.write(buffer,5);
system("PAUSE");
return 0;
```

lee 5 caracteres, los guarda en buffer y luego los imprime. Incluye la lectura de blancos, nueva linea y tabulador. Si se ingresan más de 5 caracteres trunca. Si pusiera por ejemplo:

```
char buffer[5];
cin.read(buffer,5);
cout.write(buffer,10);
cout << endl;
cout.write(buffer,3);
cout << endl << "gcount "<< cin.gcount() << endl;
cout.write(buffer,cin.gcount());
system("PAUSE");
return 0;
```

He ingreso más que 5 caracteres, trunca, imprime los 5 caracteres que se leyeron en buffer y en la impresión alinea a la izquierda y rellena con blancos hasta llegar a las 10 posiciones, el write 3 imprime los 1os 3 caracteres leidos, el cin.gcount() imprime 5 y el siguiente write imprime gcount() caracteres o sea 5.

El read espera en la entrada hasta que se ingresen los 5 caracteres.

Si se ingresa EOF (CTRL Z) antes de leer los 5 caracteres da error (establece el fail bit).

```
char buffer [] = "BUEN DIA";
cout.write(buffer,5);
```

imprime "BUEN "(trunca si hay mas caracteres en buffer).

La función cin.gcount() reporta cuantos caracteres ha leido la última operación de entrada. En el ejemplo anterior 5. Si se ingresa EOF gcount da el numero de caracteres ingresados antes del CTRL Z.

Manipuladores de flujo

Realizan tareas de formato. Proporcionan capacidades tales como establecimiento de anchura de campos, de precisiones, de los caracteres de relleno de campo, establecimiento y restablecimiento de indicadores de formato, vaciado de flujo, etc.

13. Bases: dec,oct,hex y setbase

Los enteros normalmente se interpretan como valores decimales. Para cambiar la base sobre la que se interpretan los enteros insertamos hex (para establecer la base a hexadecimal), oct (para establecer la base a octal), o dec (para reestablecer la base a decimal).

La base también se puede cambiar utilizando setbase el cual toma un argumento entero de 10, 8 o 16.

Se requiere el archivo de encabezado <iomanip.h>.

Ejemplo:

```
int main()
{
    int n;

    cout << "Introduca un numero decimal : ";
    cin >> n;
    cout << endl;
    cout << n << " en hexadecimal es: "
         << hex << n << '\n'
         << dec << n << " en octal es: "
         << oct << n << '\n'
         << setbase(10) << n << " en decimal es: "
         << n << endl;

return 0; }
```

Imprime lo siguiente:

```
Introduzca un numero decimal : 20
20 en hexadecimal es: 14
20 en octal es: 24
20 en decimal es: 20
```

14. Precisión de punto flotante (precision, setprecision)

Podemos controlar la precisión de los números de punto flotante (es decir número de digitos a la derecha del punto decimal) utilizando setprecision(lugares) o cout.precision(lugares).

Una llamada a cualquiera de estas dos funciones establece la precisión para todas las operaciones de salida subsecuentes hasta la siguiente llamada a precision o setprecision.

La funcion precision sin argumentos devuelve el valor de precisión actual.

Por ejemplo:

```
int main()
{
    double root2= sqrt(2.0);
    int lugares;

    cout << setiosflags (ios::fixed);
    cout << "Raiz cuadrada de 2 con precision 0-3.\n";
    cout << "usando precision\n";

    for(lugares=0; lugares <= 3; lugares++)
    { cout.precision(lugares);
      cout << root2 << '\n'; }

    cout << "usando setprecision\n";

    for(lugares=0; lugares <= 3; lugares++)
       cout << setprecision(lugares) << root2 << '\n';

    return 0;
}
```

la llamada

```
cout << setiosflags(ios::fixed);
```

especifica que la salida de un valor de punto flotante debe estar en notación de punto fijo con un número de digitos especifico a la derecha del punto decimal especificado por precision. Si no se especifica presicion el valor se desplega como fue cargado.

imprime:

```
Raiz cuadrada de 2 con precision 0-3.
usando precision
1
1.4
1.41
1.414
usando setprecision
1
1.4
1.41
1.414
```

15. Anchura de campo (width)

La función width de ios establece la anchura de campo (numero de posiciones) con que un valor debe enviarse a la salida o numero de posiciones con que un valor debera introducirse en una entrada y devuelve la anchura anterior.

Por ejemplo

```
int main()
{
    char arr[10];
    cin.width(5);
    cin >> arr;
    cout << arr;
    system("PAUSE");
    return 0;
}
```

consume 4 caracteres de la entrada (coloca el '\0' en arr en la posicion 5). Un valor mas ancho en la entrada que el especificado por width se trunca.

En la salida, un valor más grande que el ancho indicado no se truncará sino que se imprimira completo.

El establecimiento de anchura se aplica solo para la siguiente salida despues la anchura se establece implicitamente a 0, es decir los valores de salida serán tan anchos como necesiten serlo.

width sin argumentos devuelve el valor actual. En el ejemplo, si coloco cout << cin.width() luego de la declaración cin.width(5), imprime 5.

16. Estados de formato de flujo

Utilizamos las funciones setf, unsetf y flags para setear y resetear los indicadores.

Los indicadores son:

- ios::skipws = se saltea los caracteres de espacio en blanco en un flujo de entrada
- ios::left = alinea la salida a la izquierda de un campo. Los caracteres de relleno aparecen a la derecha en caso necesario
- ios::right = alinea la salida a la derecha de un campo. Los caracteres de relleno aparecen a la izquierda en caso necesario.
- ios::internal = indica que el signo de un numero debe estar alineado a la izquierda y la magnitud alineada a la derecha. Los caracteres de relleno aparecen en el medio, entre el signo y el numero.

- ios::dec = especifica que los enteros deben tratarse como valores decimales.

- ios::oct = especifica que los enteros deben tratarse como valores octales.

- ios::hex = especifica que los enteros deben tratarse como valores hexadecimales.

- ios::showbase = especifica que la base de un numero debe aparecer en la salida al inicio de un numero (0 para los octales, 0x para los hexadecimales).

- ios::showpoint = especifica que los numeros de punto flotante deben aparecer en la salida con un punto decimal.

- ios::uppercase = especifica que se debe utilizar X mayuscula en 0X antes de un entero hexadecimal y E mayuscula cuando se utiliza notacion cientifica para numeros en punto flotante.

- ios::showpos = especifica que los numeros positivos y negativos deben estar precedidos del signo.

- ios::scientific = especifica que la salida de un valor de punto flotante debe estar en notacion cientifica

setiosflags hace lo mismo que setf. resetiosflags hace lo mismo que unsetf. Se necesita el archivo de encabezado iomanip.

16.1. Ceros a la derecha y puntos decimales

El indicador showpoint se establece para forzar que un número de punto flotante aparezca en la salida con su punto decimal y sus ceros a la derecha. 79.0 se imprimira como 79 cuando showpoint no esté establecido y como 79.000 (con tantos ceros a la derecha como lo especifique la precision) cuando si esté establecido.

Ejemplo:

```
int main()
{
    cout << “Antes de establecer showpoint ”\n”;
         << “9.9900 se imprime como: ”<< 9.9900
         << “\n9.9000 se imprime como: ”<< 9.9000
         << “\n9.0000 se imprime como: ”<< 9.0000
         << “\ndespues de establecer showpoint\n”;
    cout.setf(ios::showpoint);
    cout << setprecision(6);
    cout << ”9.9900 se imprime como: ”<< 9.9900
```

```
        << "\n9.9000 se imprime como: "<< 9.9000
        << "\n9.0000 se imprime como: "<< 9.0000 << endl;

    return 0;
}
```

Imprime:

```
Antes de establecer showpoint
9.9900 se imprime como 9.99
9.9000 se imprime como 9.9
9.0000 se imprime como 9
despues de establecer showpoint
9.9900 se imprime como 9.99000
9.9000 se imprime como 9.90000
9.0000 se imprime como 9.00000
```

16.2. Alineación

Los indicadores left y right permiten que los campos se alineen a la izquierda o a la derecha con caracteres de relleno.

El caracter de relleno por defecto es el blanco.

La alineación por defecto es a la derecha.

Por ejemplo

```
int main()
{
    cout << "por defecto \n";
    cout << setw(10) << "hola";
    cout.setf(ios::left, ios::adjustfield);
    cout << "\ncon alineacion a la izquierda \n";
    cout << setw(10) << "hola";
    cout << "\nreestablece el predeterminado\n";
    cout.unsetf(ios::left);
    cout << setw(10) << "hola";
    system("PAUSE");
    return 0;
}
```

imprime hola alineado a la derecha, luego alineado a la izquierda y luego alineado a la derecha de nuevo.

El argumento ios::adjustfield se debe proporcionar como segundo argumento para setf cuando se establecen left, right o internal.

otra forma de setear la alineación es con setiosflags. En vez de cout.setf(ios::left) podemos colocar

```
cout << setiosflags(ios::left)
```

y en vez de cout.unsetf(ios::left)

cout << resetiosflags(ios::left)

el programa quedaria:

```
int main()
{
    cout << "por defecto \n";
    cout << setw(10) << "hola";
    cout << "\ncon alineacion a la izquierda \n";
    cout << setw(10) << setiosflags(ios::left) << "hola";
    cout << "\nreestablece el predeterminado\n";
    cout << setw(10) << resetiosflags(ios::left) << "hola";
    system("PAUSE");
    return 0;
}
```

el funcionamiento es el mismo que el del programa anterior.

16.3. Relleno, fill, setfill

La función fill especifica el caracter de relleno que es blanco por defecto.

```
int main()
{
    cout << "por defecto \n";
    cout << setw(10) << "hola\n";
    cout << "relleno = * \n";
    cout.fill('*');
    cout << setw(10) << "hola\n";
    cout << "relleno = + \n";
    cout.fill('+');
    cout << setw(10) << "hola";
    system("PAUSE");
    return 0;
}
```

imprime hola alineado a la derecha, *****hola (el hola y nueva linea utilizan 5 posiciones) y luego ++++++hola (no hay nueva linea, el hola ocupa 4 posiciones).

en vez de cout.fill podría haber utilizado cout << setfill(c). En el caso anterior:

```
int main()
{
    cout << "por defecto \n";
    cout << setw(10) << "hola\n";
    cout << "relleno = * \n";
```

```
    cout << setw(10) << setfill('*') << "hola\n";
    cout << "relleno = + \n";
    cout << setw(10) << setfill('+') << "hola";
    system("PAUSE");
    return 0;
}
```

que funciona igual que el programa anterior.

16.4. Bases, ios::dec, ios::oct, ios::hex, ios::showbase

ios::oct, ios::hex e ios::dec especifican que los enteros deben tratarse como valores octales, hexadecimales y decimales respectivamente. Se asume la base es 10 en forma predeterminada.

Se utilizan con ios::basefield como segundo argumento.

Ejemplo

```
int main()
{
    int x=100;

    cout << setiosflags(ios::showbase)
    << "Imprimiendo nros precedidos por su base: \n";
    << x << endl;
    cout.setf(ios::oct, ios::basefield);
    cout << x << endl;
    cout.setf(ios::hex, ios::basefield);
    cout << x << endl;
    system("PAUSE");
    return 0;
}
```

imprime:

```
Imprimiendo nros precedidos por su base:
100
0144
0x64
```

16.5. Notación científica, notación fija

Utilizamos el indicador ios::floatfield como segundo argumento con ios::scientific e ios::fixed.

ios::scientific se utiliza para forzar la salida de un número a formato cientifico (con exponente e en la notación). ios::fixed fuerza que un número de punto flotante se despliegue con un número especifico de dígitos (lo especificado por precision) a la derecha del punto decimal.

Si no se especifican los indicadores el formato depende de como se asigno el valor al número.

Ejemplo:

```
int main()
{
    double x=.001234, y=1.97e9;

    cout << “Formato predeterminado:”<< endl;
    cout << x <<  << y << endl;
    cout << ”Formato cientifico:”<< endl;
    cout.setf(ios::scientific, ios::floatfield);
    cout << x <<  << y << endl;
    cout << ”Formato fijo:”<< endl;
    cout.setf(ios::fixed, ios::floatfield);
    cout << x <<  << y << endl;
    system(”PAUSE”);
    return 0;
}
```

la ejecución imprime:

```
Formato predeterminado:
0.001234 1.97e+009
Formato cientifico: 1.234000e-003 1.970000e+009
Formato fijo:
0.001234 1970000000.000000
```

16.6. Control de mayusculas/minusculas

El indicador ios::uppercase se establece para forzar que se envie a la salida una X o E mayusculas con los enteros hexadecimales o con los valores de punto flotante en notación científica.

Ejemplo

```
int main()
{
    cout << setiosflags(ios::uppercase);
    cout << 4.345e10 << endl;
    cout << hex << 123456789 << endl;
    cout << resetiosflags(ios::uppercase);
    cout << 4.345e10 << endl;
    cout << hex << 123456789 << endl;
    system(”PAUSE”);
    return 0;
}
```

imprime:

```
4.345E+010
75BCD15
4.345e+010
75bcd15
```

17. Estados de error de flujo

Tenemos bits que se establecen cuando ocurren errores en la entrada o condiciones particulares como fin de entrada.

Por ejemplo, el **eofbit** se establece automáticamente para un flujo de entrada cuando se encuentra el EOF. La llamada

cin.eof()

devuelve true si se ha encontrado el EOF y false en caso contrario.

El **failbit** se establece cuando sucede un error de formato en el flujo pero no se han perdido caracteres. El bit correspondiente es cin.fail().

El **badbit** se establece cuando sucede un error que da como consecuencia la pérdida de datos. Estas fallas son por lo general no recuperables. El bit correspondiente es cin.bad().

El **goodbit** se establece si ninguno de eofbit, failbit o badbit se estableció. El bit correspondiente es cin.good().

El medio preferido para probar el estado de un flujo es utilizar las funciones eof, bad, fail y good.

La función clear restaura el estado de un flujo a "bueno"para que la E/S pueda continuar en ese flujo. La instrucción:

cin.clear()

limpia a cin y reestablece goobit. La instrucción

cin.clear(ios::failbit)

establece en 1 el failbit.

Ejemplo:

```
int main()
{
    int i;

    cout << "Introduzca un caracter \n";
    cin >> i;
    cout << "\n cin.fail() "<< cin.fail();
    cout << "\n cin.good() "<< cin.good();
    cin.clear(ios::goodbit);
    cout << "\n cin.fail() "<< cin.fail();
    cout << "\n cin.good() "<< cin.good();
    system("PAUSE");
    return 0;
}
```

Si introducimos el caracter 'a', cin.fail() tendrá el valor 1 y cin.good() tendrá el valor 0. Luego de reestablecer el goodbit el failbit tendrá valor 0 y el goodbit el valor 1.