GUÍA DE APRENDIZAJE PARA CONCURSANTES ICPC Y IOI: LECTURA E IMPRESIÓN DE DATOS EN C++

1. Introducción

En el diseño del algoritmo solución a un problema de concursos hay dos operaciones que no pueden faltar o al menos una de las dos está presente. Estas operaciones son las lectura e impresión de datos. Dichas operaciones están presentes en la casi totalidad de las soluciones algorítmicas a los problemas de concursos. Es por esos que se hace necesario conocer que recursos nos ofrecen los lenguajes de programación para llevar a cabo dichas operaciones.

2. Desarrollo

Para la lectura e impresión de datos se puede realizar por dos métodos por **cin** y **cout** primitivos de C++ o por **printf** y **scanf** primitivos de C, en el programa se pueden usar ambos métodos si es necesario.

2.1. Primitivos de C++

En sus programas, si usted desea hacer uso de los objetos **cin** y **cout** tendrá que incluir el uso de la biblioteca iostream (por medio de la directiva #include). La iostream es la biblioteca estándar en C++ para poder tener acceso a los dispositivos estándar de entrada y/o salida. Si usted usa la directiva #include <iostream.h> o #include <iostream> en sus programas, automáticamente la iostream pone a su disposición los objetos **cin** y **cout** en el ámbito estándar (std), de tal manera que usted puede comenzar a enviar o recibir información a través de los mismos sin siquiera preocuparse de su creación.

Para el manejo de **cin** y **cout** se necesita también el uso de los **operadores de direccionamiento** << y >>. Los operadores de direccionamiento son los encargados de manipular el flujo de datos desde o hacia el dispositivo referenciado por un stream específico. El operador de direccionamiento para salidas es una pareja de símbolos de "menor que"<<, y el operador de direccionamiento para entradas es una pareja de símbolos de "mayor que">>. Los operadores de direccionamiento se colocan entre dos operandos, el primero es el stream y el segundo es una variable o constante que proporciona o recibe los datos de la operación.

2.1.1. Funciones miembro get y getline

La función miembro **get** sin argumentos recibe como entrada un carácter del flujo designado (incluyendo caracteres de espacio en blanco y otros caracteres no gráfi cos, como la secuencia de teclas que representa el fi n de archivo) y lo devuelve como el valor de la llamada a la función. Esta versión de get devuelve EOF cuando se encuentra el fin del archivo en el flujo.

La función miembro **getline** opera de manera similar, inserta un carácter nulo después de la línea en el arreglo de caracteres. La función **getline** elimina el delimitador del flujo (es decir, lee el carácter y lo descarta), pero no lo almacena en el arreglo de caracteres.

2.1.2. Precisión de punto flotante (precision , setprecision)

Para controlar la precisión de los números de punto flotante (es decir, el número de dígitos a la derecha del punto decimal), podemos usar el manipulador de flujo **setprecision** o la función miembro **precision** de ios_base . Una llamada a uno de estos miembros establece la precisión para todas las operaciones de salida subsecuentes, hasta la siguiente llamada para establecer la precisión. Una llamada a la función miembro **precision** sin argumento devuelve la opción de precisión actual (esto es lo que necesitamos usar para poder restaurar la precisión original en un momento dado, una vez que ya no sea necesaria una opción pegajosa).

2.1.3. Anchura de campos (width , setw)

La función miembro **width** (de la clase base ios_base) establece la anchura de campo (es decir, el número de posiciones de caracteres en los que debe imprimirse un valor, o el número máximo de caracteres que deben introducirse) y devuelve la anchura anterior. Si los valores que se imprimen son menos que la anchura de campo, se insertan caracteres de relleno como relleno (padding). Un valor más ancho que la anchura designada no se truncará; se imprimirá el número completo. La función **width** sin argumento devuelve la configuración actual.

2.1.4. Estados de formato de flujos y manipuladores de flujos

Se pueden utilizar varios manipuladores de flujos para especificar los tipos de formato a realizar durante las operaciones de E/S de flujos. Los manipuladores de flujos controlan la configuración del formato de la salida. Todos estos manipuladores pertenecen a la clase ios_base.

Manipulador de flujo	Descripción
skipws	Omite los caracteres de espacio en blanco en un flujo de entrada. Esta opción se restablece con el manipulador de flujo noskipws .
left	Justifica la salida a la izquierda en un campo. Si es necesario, aparecen caracteres de relleno a la derecha.
right	Justifica la salida a la derecha en un campo. Si es necesario, aparecen caracteres de relleno a la izquierda.
internal	Indica que el signo de un número debe justifi carse a la izquierda en un campo, y que la magnitud del número se debe justificar a la derecha en ese mismo campo (es decir, deben aparecer caracteres de relleno entre el signo y el número).
dec	Especifica que los enteros deben tratarse como valores decimales (base 10).
oct	Especifica que los enteros se deben tratar como valores octales (base 8).
hex	Especifica que los enteros se deben tratar como valores hexadecimales (base 16).

showbase	Especifica que la base de un número se debe imprimir adelante del mismo (un 0 a la izquierda para los valores octales; 0x o 0X a la izquierda para los valores hexadecimales). Esta opción se restablece con el manipulador de flujo noshowbase .
showpoint	Especifica que los números de punto f lotante se deben imprimir con un punto decimal. Esto se usa generalmente con fixed para garantizar cierto número de dígitos a la derecha del punto decimal, aun y cuando sean ceros. Esta opción se restablece con el manipulador de flujo noshowpoint.
uppercase	Especifica que deben usarse letras mayúsculas (es decir, X y de la A a la F) en un entero hexadecimal, y que se debe usar la letra E al representar un valor de punto flotante en notación científica. Esta opción se restablece con el manipulador de flujo nouppercase .
showpos	Especifica que a los números positivos se les debe anteponer un signo positivo (+). Esta opción se restablece con el manipulador de flujo noshowpos .
scientific	Especifica la salida de un valor de punto flotante en notación científica.
fixed	Especifica la salida de un valor de punto flotante en notación de punto fijo, con un número específico de dígitos a la derecha del punto decimal.

2.2. Primitivos de C

En sus programas, si usted desea hacer uso de los objetos **scanf** y **printf** tendrá que incluir el uso de la biblioteca cstdio (por medio de la directiva #include <cstdio>). Cstdio es la biblioteca estándar en C para poder tener acceso a los dispositivos estándar de entrada y/o salida.

Por defecto las funciones de entrada/salida de C (Input/Output) son un conjunto de funciones, incluidas con el compilador, que permiten a un programa recibir y enviar datos al exterior. Para su utilización es necesario incluir, al comienzo del programa, el archivo stdio.h en el que están definidos sus prototipos #include <stdio.h>donde stdio proviene de *standard-input-output*.

Generalmente, **printf()** y **scanf()** funcionan utilizando cada una de ellas una *tira de caracteres decontrol* y una lista de *argumentos*. Veremos estas características; en primer lugar en **printf()**, y a continuación en **scanf()**.

Las instrucciones que se han de dar a **printf()** cuando se desea imprimir una variable dependen del tipo de variable de que se trate. Así, tendremos que utilizar la notación %d para imprimir un entero, y %c para imprimir un carácter, como ya se ha dicho. A continuación damos la lista de todos los identificadores que emplea la función **printf()** y el tipo de salida que imprimen. La mayor parte de sus necesidades queda cubierta con los ocho primeros; de todas formas, ahí están los dos restantes por si desea emplearlos.

Identificador	Salida
%c	Carácter o entero pequeño
%s	Tira de caracteres
%d, %i	Entero decimal
%u	Entero decimal sin signo
%lld	Entero largo
%llu	Entero largo sin signo
%f	Número de punto flotante en notación decimal
%lf	Número de punto flotante en notación decimal con doble presición
%o	Entero octal sin signo
%x	Entero hexadecimal sin signo

El formato para uso de **printf()** es éste:

printf(Control, item1, item2, ...);

item1, **item2**, etc., son las distintas variables o constantes a imprimir. Pueden también ser expresiones, las cuales se evalúan antes de imprimir el resultado. **Control** es una tira de caracteres que describen la manera en que han de imprimirse los items.

También se debe hablar de **modificadores de especificaciones de conversión** en printf(), estos son apéndices que se agregan a los especificadores de conversión básicos para modificar la salida. Se colocan entre el símbolo % y el carácter que define el tipo de conversión. A continuación se da una lista de los símbolos que está permitido emplear. Si se utiliza más de un modificador en el mismo sitio, el orden en que se indican deberá ser el mismo que aparece en la tabla. Tenga presente que no todas las combinaciones son posibles.

1. **-**: El ítem correspondiente se comenzará a escribir empezando en el extremo izquierdo del campo que tenga asignado. Normalmente se escribe el ítem de forma que acabe a la derecha del campo. Ejemplo: %-10d
2. **número**: Anchura mínima del campo. En el caso de que la cantidad a imprimir (o la tira de caracteres) no quepa en el lugar asignado, se usará automáticamente un campo mayor. Ejemplo: %4d
3. **.número**: Precisión. En tipos flotantes es la cantidad de cifras que se han de imprimir a la derecha del punto (es decir, el número de decimales). En el caso de tiras, es el máximo número de caracteres que se ha de imprimir. Ejemplo: %.2f (dos decimales)

Para el uso de **scanf()** se utilizan los mismo identificadores que en **printf()**, y al igual que **printf()**, **scanf()** emplea una tira de caracteres de control y una lista de argumentos. La mayor diferencia entre ambas está en esta última; **printf()** utiliza en sus listas nombres de variables, constantes y expresiones; **scanf()** usa punteros a variable. Afortunadamente no se necesita saber mucho de punteros para emplear esta expresión; se trata simplemente de seguir las dos reglas que se dan a continuación:

- Si se desea leer un valor perteneciente a cualquier de los tipos básicos coloque el nombre de la variable precedido por un &.

- Si lo que desea es leer una variable de tipo tira de caracteres, no use &.

Otro detalle a tomar en cuenta es que **scanf()** considera que dos ítems de entrada son diferentes cuando están separados por blancos, tabulados o espacios. Va encajando cada especificador de conversión con su campo correspondiente, ignorando los blancos intermedios. La única excepción es la especificación %c, que lee el siguiente caracter, sea blando o no.

2.2.1. Macros getchar() y putchar()

Las macros **getchar()** y **putchar()** permiten respectivamente leer e imprimir un sólo carácter cada vez, en la entrada o en la salida estándar. La macro **getchar()** recoge un carácter introducido por teclado y lo deja disponible como valor de retorno. La macro **putchar()** escribe en la pantalla el carácter que se le pasa como argumento.

3. Implementación

3.1. Primitivos de C++

```
#include <iostream.h>
#include <math.h>
using namespace std ;
int main(){
   cout << "Hola mundo"; // imprimir mensaje (en la pantalla)
   char nombre [80];
   cout << " Entre su nombre : " ;
   cin >> nombre ;
   cout << " Hola , " << nombre ;
   int A,B,C;
   cin >> A >> B >> C ;
   // Ejemplo con una unica linea, se muestra el uso de cout y endl
   cout << "Bienvenido. Soy un programa. Estoy en una linea de
codigo ." <<endl ;
   // Ejemplo con una unica linea de codigo que se puede fraccionar
   // mediante el uso de '<<'
   cout << "Ahora "
   << "estoy fraccionado en el codigo , pero en la consola me
   muestro como una unica frase ."
   << endl ;

   // Uso de un codigo largo, que cuesta leer para un programador,
   // y que se ejecutara sin problemas.
   // *** No se recomienda hacer lineas de esta manera,
   // esta forma de programar no es apropiada ***
   cout << " Un gran texto puede ocupar muchas lineas . "
   << endl
   << " Pero eso no frena al programador a que todo se pueda
   poner en una unica linea de codigo y que "
   << endl
```

```
   << " el programa , al ejecutarse , lo situe como el
   programador quiso "
   << endl ;

   double raiz2 = sqrt( 2.0 ); // calcula la raiz cuadrada de 2
   int posiciones; // precision, varia de 0 a 9
   cout << "Raiz cuadrada de 2 con precisiones de 0 a 9." << endl
   << "Precision establecida mediante la funcion miembro precision "
   << "de ios_base:" << endl;
   cout << fixed; // usa el formato de punto fijo
   // muestra la raiz cuadrada usando la funcion precision de ios_base
   for ( posiciones = 0; posiciones <= 9; posiciones++ ){
        cout.precision( posiciones );
        cout << raiz2 << endl;
   } // fin de for


   int valorAnchura = 4;
   char enunciado[ 10 ];
   cout << "Escriba un enunciado:" << endl;
   cin.width( 5 ); // introduce solo 5 caracteres de enunciado
   // establece la anchura de campo y despues muestra los caracteres con base
      en esa anchura
   while ( cin >> enunciado ){
        cout.width( valorAnchura++ );
        cout << enunciado << endl;
        cin.width( 5 ); // introduce 5 caracteres mas de enunciado
   } // fin de while
   return 0;
}
```

3.2. Primitivos de C

```
#include <cstdio>
#define PI 3.14159
using namespace std ;
int main (){
   int numero = 5;
   int coste = 50;
   // No olvidar que '\n' es fin de linea
   printf("Los %d jovenes tomaron %d helados .\ n ", numero, numero *2);
   printf("El valor de PI es %f.\n");
   printf("Esta es una linea sin variables.") ;
   printf(" %c %d ",'$ ',coste) ;

   printf("Modificador'-'\n");
   printf("/%-10d/\n",365);
   printf("/%-6d/\n",365);
```

```
    printf("/%-2d/",365);
    printf("en este caso no sirve de nada el modificador\n");
    printf("Modificador 'numero'\n");
    printf("/%2d/",365);
    printf("en este caso tampoco hace efecto el modificador\n");
    printf("/%6d/\n",365);
    printf("/%10d/\n" , 365);
    printf("Modificador'.numero'\n");
    printf("/%.1f/\n",365.897) ;
    printf("/%.6f/\n",365.897) ;
    printf("/%.3s/\n","AsDfGhJkL") ;
    printf("/%.100s/","AsDfGhJkL");
    printf("No se rellena con 0's como en los ejemplos anteriores \n");

    int edad ;
    char nombre [30];
    printf("Ingrese su edad y nombre\n") ;
    scanf("%d%s",&edad,nombre);//Note que & no es necesario en nombre
    printf("Nombre :%s , Edad :%d",nombre,edad);

    putchar('a');

    char c = getchar();

    scanf("%[^\n]", nombre); // Leer todo hasta un salto de linea

    return 0;
}
```

4. Complejidad

Las formas formas primitivas de C de leer e imprimir datos son muchas mas rápidas que las formas primitivas de C++ pero lo anterior se puede solventar facilemente colocando las siguientes lineas al principio de nuestra solución:

```
ios_base::sync_with_stdio(0);
cin.tie(0);
cout.tie(0);
```

Lo anterior permite que las funciones de lectura e impresión primitivas de C++ se equiparen en cuanto a tiempo a las funciones primitivas de C de lectura e impresión, aquí importante es conocer que si utilizamos esta variante no podemos combinar las funciones primitivas de C++ con las de C, debemos usar siempre la de C++. El otro detalle a tener en cuenta es la función **endl** de C++ que imprime un salto de linea en pantalla lo cual es muy útil en varios problemas acorde al formato de salida. Esta función es lenta por tanto se recomienda ser sustituida directamente por su valor ' nó creamos una macro:

```
#define ENDL '\n'
```

y donde queramos utilizarla podemos la macro ENDL.

5. Aplicaciones

Es indudable que si no conocemos como leer e imprimir datos nunca podremos solucionar un ejercicio o problema de concurso ya que los algoritmos solución siempre van a contar con estas operaciones. Tambíen conocer las diferentes variantes de lectura e impresión nos puede ayudar a minimizar la complejidad algorítmica para determinadas entradas y salidas de algunos problemas con nos vamos a enfrentar donde su máxima complejidad puediera estar en estas operaciones.