GUÍA DE APRENDIZAJE PARA CONCURSANTES ICPC Y IOI: ARREGLOS

1. Introducción

En varios problemas debemos trabajar con varias informaciones de forma grupal y dichas informaciones tienen en común que comporten el mismo tipo de dato. Una variante bastante trivial es declarar un variable por cada información con que necesito trabajar. Por ejemplo si tengo el salario mensual de un trabajador durante un año con declarar 12 variables me sería suficiente para luego realizar un grupo de operaciones como el promedio salarial, el mínimo y máximo salario del trabajador. Estas operaciones aunque se pueden implementar no cabe duda que tienen su complejidad en cuanto a la implementación que puede ser un tanto tediosas. Bueno imaginemos que ahora tengamos que hacer ese mismo trabajo con un quinquenio o decada de trabajo del trabajador la complejidad de implementación aumentería casi que literal en cino o diez veces más.

Vamos a ver como podemos resolver esto en programación con el uso de la estructura de datos más primitiva que nos brinda los lenguajes de programación: **arreglos**

2. Conocimientos previos

2.1. Estructura de datos

Una estructura de datos es una forma de organizar un conjunto de datos elementales con el objetivo de facilitar su manipulación. Un dato elemental es la mínima información que se tiene en un sistema. Una estructura de datos define la organización e interrelación de estos y un conjunto de operaciones que se pueden realizar sobre ellos. Cada estructura ofrece ventajas y desventajas en relación a la simplicidad y eficiencia para la realización de cada operación. De está forma, la elección de la estructura de datos apropiada para cada problema depende de factores como la frecuencia y el orden en que se realiza cada operación sobre los datos.

2.2. Estructuras de datos estáticas compuestas

Las estructuras de datos estáticas son aquellas que el tamaño de las mismas no puede ser modificadas en la ejecución del algoritmo. Así su tamaño tiene que ser defino en la creación de la misma. Está estructura almacena varios elementos del mismo tipo en forma lineal.

2.3. Operador *new*

C++ y Java permite a los programadores controlar la asignación de memoria en un programa, para cualquier tipo integrado o definido por el usuario. Esto se conoce como administración dinámica de memoria y se lleva a cabo mediante el operador *new*. Podemos usar el operador *new* para asignar (reservar) en forma dinámica la cantidad exacta de memoria requerida para contener cada nombre en tiempo de ejecución. La asignación dinámica de memoria de esta forma hace que se cree un arreglo (o cualquier otro tipo integrado o definido por el usuario) en el almacenamiento libre (algunas veces conocido como el heap o montón): una región de memoria asignada a cada programa para almacenar los objetos que se asignan en forma dinámica. Una vez que se asigna la memoria para un arreglo en el almacenamiento libre, podemos obtener acceso a éste si apuntamos un apuntador al primer elemento del arreglo.

3. Desarrollo

En programación, un arreglo (llamados en inglés array) es una zona de almacenamiento continuo, que contiene una serie de elementos del mismo tipo.

La forma de acceder a los elementos de un arreglo es directa; esto significa que el elemento deseado es obtenido a partir de su índice y no hay que ir buscándolo elemento por elemento (en contraposición, en el caso de una lista, para llegar, por ejemplo, al tercer elemento hay que acceder a los dos anteriores o almacenar un apuntador o puntero que permita acceder de manera rápida a ese elemento).

Para hacer referencia a cualquiera de estos elementos en un programa, se proporciona el nombre del arreglo seguido del número de posición del elemento específico entre corchetes ([]). Al número de posición se le conoce más formalmente como el **índice** o **subíndice** (este número especifica el número de elementos a partir del inicio del arreglo). El primer elemento en todo arreglo tiene el subíndice 0 (cero) y se conoce algunas veces como el elemento cero.

Un **subíndice** debe ser un entero o una expresión entera (usando cualquier tipo integral). Si un programa utiliza una expresión como un subíndice, entonces el programa evalúa la expresión para determinar el subíndice.

4. Implementación

4.1. C++

4.1.1. Declaración y creación de arreglos

El arreglo se puede declarar según la situación. Ahora veremos cada una de ellas:

```
/* Sabemos la cantidad de elementos a priori,
<tipo_de _dato> <nombre_arreglo> [<cantidad>];*/
bool isPimes[100];

/*La cantidad de elementos puede variar y depende del
  valor de una variable.Es la mas usada
  <tipo_de _dato> * <nombre_arreglo>= new <tipo_de _dato> [<variable>];
  <tipo_de _dato> * <nombre_arreglo>= new <tipo_de _dato> [<cantidad>];
*/
int cantidadMaxima=100;
int * notas;
notas=new int [cantidadMaxima];
double * promedio=new double [100];

/*Cuando conocemos los valores que integran el arreglo*/
string nombres []={ "Luis", "Ernesto", "Susana" };
```

4.1.2. Almacenar valor en el arreglo

Para almacenar un valor en el arreglo solo debemos inidicar la posición en el arreglo en que se va almacenar dicha posición debe ser un valor o variable entera y debe estar en rango $[0, capacidad-1]$.

```
/* <nombre arreglo>[<posicion>] = <valor>; donde:
 <posicion> puede ser una variable entera, valor entero literal, expresion
    cuyo resultado sea entero
 <valor> puede ser una variable entera , valor entero literal, expresion
    cuyo resultado sea entero */
notas[23]=cantidadMaxima;
notas[cantidadMaxima-50]=34;
notas[10+5]=2*cantidadMaxima;
```

4.1.3. Obtener valor en el arreglo

Para obtener un valor en el arreglo solo debemos inidicar la posición en el arreglo en que se va obtener dicha posición debe ser un valor o variable entera y debe estar en rango $[0, capacidad - 1]$.

```
/* <variable> = <nombre arreglo>[<posicion>]; donde:
 <posicion> puede ser una variable entera, valor entero literal, expresion
    cuyo resultado sea entero
 <variable> va ser la el lugar donde se almacenara una copia del valor
    solicitado al arreglo, debe ser del mismo tipo de dato del arreglo */
int a =notas[23];
int b;
b = notas[cantidadMaxima-50];
```

4.1.4. Recorrer todos los elementos del arreglo

Para recorrer todos los elementos o parte de los elementos almacenados en el arreglo vamos utilizar la instrucción **for**

```
/* for( int i=<posicion_inicial>; i < <posicion_final>; i++ )
   i++ si <posicion_inicial> <= <posicion_final>
   i-- si <posicion_inicial> > <posicion_final>
   i <= <posicion_final> si deseo incluir en el rango la ultima posicion
      pero tiene que ser una posicion valida del arreglo
*/

for(int i=0;i<cantidadMaxima;i++){
   //notas[i] accedo al valor almacenado en la inesima posicion del arreglo
}
```

Noten que acuerdo como hagan el **for** pueden recorrer del principio al final o en sentido contrario

4.1.5. Leer los valores y almacenarlos en el arreglo

Para leer los valores consola y almacernalos directamente en el arreglo vamos a utilizar una combinación de recorrer todos los elementos y almacenar en arreglo visto anteriormente

```
/* En cada iteracion del for el cin lee un valor y dicho valor es
   almacenado en el arreglo en la posicion que indique el valor de la
   variable i en esa iteracion, en la primera el valor seria 0, en la
   segunda 1 y asi sucesivamente. Tener en cuenta que los valores de i
   esten en rango de posiciones validas del arreglo */
for(int i=0;i<cantidadMaxima;i++){
   cin>>notas[i];
}
```

4.1.6. Imprimir los valores almacenados en el arreglo

Para imprimir los valores en consola del arreglo vamos a utilizar una combinación de recorrer todos los elementos y obtener en arreglo visto anteriormente

```
/* En cada iteracion del for el cin imprimir un valor que se corresponde
   con el almacenado en el arreglo en la posicion que indique el valor de
   la variable i en esa iteracion, en la primera el valor seria 0, en la
   segunda 1 y asi sucesivamente. Tener en cuenta que los valores de i
   esten en rango de posiciones validas del arreglo */
for(int i=0;i<cantidadMaxima;i++){
   cout<<notas[i]<<" ";
}
cout<<endl;
for(int i=0;i<cantidadMaxima;i++){
   cout<<notas[i]<<endl;
}
```

En primer **for** se imprime todos los valores del arreglo en una sola linea separados por espacios, mientras en el segundo cada valor almacenado en el arreglo se imprime por linea.

4.2. Java

4.2.1. Declaración y creación de arreglos

El arreglo se puede declarar según la situación. Ahora veremos cada una de ellas:

```
/*La cantidad de elementos puede variar y depende del
 valor de una variable.Es la mas usada
<tipo_de _dato> [] <nombre_arreglo>= new <tipo_de _dato> [<variable>];
```

```
<tipo_de _dato> [] <nombre_arreglo>= new <tipo_de _dato> [<cantidad>];*/
int cantidadMaxima=100;
int [] notas;
notas=new int [cantidadMaxima];
double [] promedio=new double [100];

/*Cuando conocemos los valores que integran el arreglo*/
String [] nombres ={ "Luis", "Ernesto", "Susana" };
```

4.2.2. Almacenar valor en el arreglo

Para almacenar un valor en el arreglo solo debemos inidicar la posición en el arreglo en que se va almacenar dicha posición debe ser un valor o variable entera y debe estar en rango $[0, capacidad-1]$.

```
/* <nombre arreglo>[<posicion>] = <valor>; donde:
 <posicion> puede ser una variable entera, valor entero literal, expresion
    cuyo resultado sea entero
 <valor> puede ser una variable entera , valor entero literal, expresion
    cuyo resultado sea entero*/
notas[23]=cantidadMaxima;
notas[cantidadMaxima-50]=34;
notas[10+5]=2*cantidadMaxima;
```

4.2.3. Obtener valor en el arreglo

Para obtener un valor en el arreglo solo debemos inidicar la posición en el arreglo en que se va obtener dicha posición debe ser un valor o variable entera y debe estar en rango $[0, capacidad - 1]$.

```
/* <variable> = <nombre arreglo>[<posicion>]; donde:
<posicion> puede ser una variable entera, valor entero literal, expresion
   cuyo resultado sea entero
<variable> va ser la el lugar donde se almacenara una copia del valor
   solicitado al arreglo, debe ser del mismo tipo de dato del arreglo */
int a =notas[23];
int b;
b = notas[cantidadMaxima-50];
```

4.2.4. Recorrer todos los elementos del arreglo

Para recorrer todos los elementos o parte de los elementos almacenados en el arreglo vamos utilizar la instrucción **for**

```
/* for( int i=<posicion_inicial>; i < <posicion_final>; i++ )
```

```
    i++ si <posicion_inicial> <= <posicion_final>
    i-- si <posicion_inicial> > <posicion_final>
    i <= <posicion_final> si deseo incluir en el rango la ultima posicion pero
        tiene que ser una posicion valida del arreglo
   */

   for(int i=0;i<cantidadMaxima;i++){
      //notas[i] accedo al valor almacenado en la inesima posicion del arreglo
   }
```

Noten que acuerdo como hagan el **for** pueden recorrer del principio al final o en sentido contrario

4.2.5. Leer los valores y almacenarlos en el arreglo

Para leer los valores consola y almacernalos directamente en el arreglo vamos a utilizar una combinación de recorrer todos los elementos y almacenar en arreglo visto anteriormente

```
   /* En cada iteracion del for el cin lee un valor y dicho valor es
      almacenado en el arreglo en la posicion que indique el valor de la
      variable i en esa iteracion, en la primera el valor seria 0, en la
      segunda 1 y asi sucesivamente. Tener en cuenta que los valores de i
      esten en rango de posiciones validas del arreglo */
   Scanner in =new Scanner(System.in);
   for(int i=0;i<cantidadMaxima;i++){
      notas[i] = in.nextInt();
   }
```

4.2.6. Imprimir los valores almacenados en el arreglo

Para imprimir los valores en consola del arreglo vamos a utilizar una combinación de recorrer todos los elementos y obtener en arreglo visto anteriormente

```
   /* En cada iteracion del for el cin imprimir un valor que se corresponde
      con el almacenado en el arreglo en la posicion que indique el valor de
      la variable i en esa iteracion, en la primera el valor seria 0, en la
      segunda 1 y asi sucesivamente. Tener en cuenta que los valores de i
      esten en rango de posiciones validas del arreglo */
   for(int i=0;i<cantidadMaxima;i++){
      System.out.print( notas[i]+" ");
   }
   System.out.println();
   for(int i=0;i<cantidadMaxima;i++){
      System.out.println(notas[i]);
   }
```

En primer **for** se imprime todos los valores del arreglo en una sola linea separados por espacios, mientras en el segundo cada valor almacenado en el arreglo se imprime por linea.

5. Complejidad

La complejidad de los arreglos esta dada por el uso de memoria que siempre va ser O(N) donde N es la cantidad de elementos o capacidad que se haya defenido al arreglo. En cuanto la complejidad de tiempo va estar en gran medida del trabajo que hagamos con él.

6. Aplicaciones

Esta estructura de dato son adecuadas para situaciones en las que el acceso a los datos se realice de forma aleatoria e impredecible. Por el contrario, si los elementos pueden estar ordenados y se va a utilizar acceso secuencial sería más adecuado utilizar una lista, ya que esta estructura puede cambiar de tamaño fácilmente durante la ejecución de un programa.

Es utilizada para cuando necesitamos almacenar varias informaciones que pueden responder a un mismo tipo de dato y que nos hace falta manipular de forma grupal con el uso de una sola variable.

7. Ejercicios propuestos

A continuación una lista de ejercicios que se resuelven utilizando arreglos:

- DMOJ - Bellotas.
- DMOJ - Hay Bales
- DMOJ - Cambiando Luces
- DMOJ - Fry y el Fuera de Juego
- CodeForces - A. In Search of an Easy Problem