Arreglos en C++

Club de Programacion Competitiva Pu++

30 de marzo del 2020

Una vez que sabemos utilizar los tipos básicos que nos ofrece C++ (`int`, `double`, `...`), sería bueno saber como trabajar con conjuntos de estos. Podemos pensar por ejemplo en las calificaciones de un grupo como un conjunto de `int`, o la temperatura durante el último mes como un conjunto de `double`.

Llamamos una **estructura de datos** a una manera de guardar y manipular eficientemente una cantidad arbitraria de datos que necesitemos.

La EDD mas sencilla y mas usada en programación competitiva es un **arreglo**. Podemos pensar un arreglo como un contenedor de datos de una capacidad fija.

Un arreglo de enteros de tamaño 5, podría verse de la siguiente manera:

3	4	5	1	4

Es importante recalcar que la capacidad de un arreglo es fija; no puede modificarse despues de que se crea. Esto trae sus pros y contras, y mas adelante veremos maneras de evadir algunos de los contras ;)

En un arreglo los datos estan inherentemente ordenados, por lo que la manera de acceder a estos es por medio de su posición, o **índice**.

Debido a una tradición milenaria, los índices empiezan desde cero, es decir, el primer elemento del arreglo tiene índice 0, el segundo tiene índice 1, y asi sucesivamente.

En el ejemplo anterior, el entero en el índice 2 es el 5.

Debe ser claro ahora que un que un arreglo de tamaño `n` tiene índices `0, ... , n - 1`.

Un arreglo en C++ se declara con el siguiente formato

```
tipo nombre[tamaño]
```

`tipo` es el tipo de datos de los elementos del arreglo y `nombre` es el nombre de la variable que ocuparemos para referirnos al arreglo.

Por ejemplo, para guardar las edades y estaturas de un grupo de 30 personas podemos declarar

```
int edad[30];
double estatura[30];
```

Para acceder o modificar el arreglo `a` en el índice `i`, lo haremos escribiendo `a[i]`

```
int a[10];
a[0] = 3;
cout << a[0] << endl;
// Esto imprime:
// 3
```

El índice al que queremos acceder tambien puede ser en si una variable. Entonces, ¿Se les ocurre alguna manera de recorrer un arreglo por completo usando los ciclos que ya saben?

La manera mas clara y usual de hacerlo es usando un ciclo `for`

```
int n = 100;
int a[n];
for(int i = 0; i < n; i++){
  a[i] = 1;
}
```

El ejemplo anterior hace 1 todos los elementos de arreglo.

Aunque también podemos recorrer el arreglo usando un `while`, en la práctica no se ve mucho. Sin embargo, les recomendamos intentar programarlo c:

Otros ejercicios que deberian intentar:

- Recorrer solo la primera mitad de un arreglo
- Recorrer solo la segunda mitad de un arreglo
- Recorrer un arreglo en sentido contrario
- Recorrer solo los índices pares de un arreglo
- Combinaciones de los anteriores

Ademas de los arreglos unidimensionales (los que hemos explicado hasta el momento), C++ tambien nos permite trabajar con arreglos de 2, 3 o más dimensiones, con un funcionamiento en muchos sentidos similar. Los siguientes ejemplos son con arreglos 2D, y para mas dimensiones el comportamiento es similar.

Un arreglo 2D lo podemos entender como un tablero con cierto numero de filas y columnas, y se declara de la siguiente manera

```
// n es el numero de filas
// m es el numero de columnas
int n = 3, m = 5;

int a[n][m];
```

El arreglo del ejemplo anterior se vería de la siguiente manera, con los números en azul indicando los índices de fila y los rojos los índices de columna

0, 0	0, 1	0, 2	0, 3	0, 4
1, 0	1, 1	1, 2	1, 3	1, 4
2, 0	2, 1	2, 2	2, 3	2, 4

Para acceder o modificar un valor en específico, accedemos especificando ambos índices

```
a[1][3] = 10;
cout << a[1][3] << endl;
// Esto imprime:
// 10
```

Para arreglos de más dimensiones (más de 3 es raro incluso en programación competitiva), todo lo explicado es análogo.

Tips para programacion competitiva

Ahora les presentaremos unos cuantos tips que son muy útiles a la hora de programar soluciones a problemas de programación competitiva.

Es importante mencionar, sin embargo, que algunos de estos tips podrian verse como “malas prácticas” de programación en otros ambientes. Ténganlo en cuenta y úsenlos con discreción.

Un arreglo en C++, por defecto, no se inicializa con ningún valor en específico; el arreglo despues de declararse contiene lo que sea que estaba antes en el cacho de memoria que se le asignó, que usualmente es basura. Para comprobarlo pueden intentar algo como lo siguiente

```
int a[10];
cout << a[0] << endl;
//Esto deberia imprimir algun valor arbitrario
```

Esto es algo que siempre hay que tener en cuenta, ya que muchos bugs se llegan a dar debido a esto. **Siempre inicializen los valores antes de utilizarlos**.

Algunas veces necesitaremos inicializar un arreglo con valores que conocemos de antemano, lo cual podríamos hacer como

```
int a[4];
a[0] = 1;
a[1] = 2;
a[2] = 4;
a[3] = 8;
```

Sin embargo, C++ nos provee una manera mas eficiente

```
int a[4] = {1, 2, 4, 8};
```

Tambien es muy usual querer inicializar un arreglo con 0 en todos sus elementos. Un ciclo `for` es la manera estándar de hacerlo

```
int n = 100;
int a[n];
for(int i = 0; i < n; i++){
  a[i] = 0;
}
```

Pero para C++ nos da una manera mas fácil

```
int n = 100;
int a[n] = {};
```

En muchos problemas tendremos que resolver varios casos que se nos presentarán, y es posible que necesitemos un arreglo para resolver cada caso

```
int main(){
  // ...
  // Para cada caso
  for(int t = 0; t < tests; t++){
    // ...
    int a[n];
    // ...
  }
}
```

Podemos ver que estamos declarando `a` tantas veces como casos de prueba, lo cual puede aumentar el tiempo de ejecución considerablemente. Para evitar esto podemos usar lo siguiente

Declaramos `a` como un arreglo global del tamaño máximo que pueda tomar `n` segun la especificación del problema, y ocupar en cada caso solo los elementos que necesitemos.

```
const int MAXN = 100;
int a[MAXN];

int main(){
  // Para cada caso
  for(int t = 0; t < tests; t++){
    /*
      aqui podemos usar a, que ya esta declarado
      (por cierto, esto es un comentario de bloque)
     */
  }
}
```

El modificador `const` en la declaración de `MAXN` significa que este entero es una constante y su valor no puede se modificado una vez que se asigna.

Aunque usar variables globales no es correcto en programacion en general, suele ser util en programacion competitiva. Aparte de lo que acabamos de mencionar, resulta que `gcc` (el compilador de C++ que usamos y el que se usa en la mayoría de las competencias) inicializa en 0 todas las variables, incluidos arreglos, que se declaren como globales (fuera de una función).

Otro consejo es siempre declarar nuestros arreglos con cierto "colchón" en cuanto a la capacidad para evitar errores de índice fuera de rango.

```
const int MAXN = 100;
// Declaramos el arreglo con un colchon de 10
int a[MAXN + 10];

int main(){
  // ...
}
```

Aunque los arreglos puedan parecer una estructura muy sencilla, son extremadamente útiles al resolver problemas, y dominar su uso seguramente les ayudará a mejorar mucho.

Los problemas del siguiente concurso les ayudarán a familiarizarse más con el uso de los arreglos y otros temas que hemos visto hasta el momento.

`https://vjudge.net/contest/366125`

Sigan praticando!