(75.41) Algoritmos y Programación II

Cátedra Lic. Andrés Juárez

2do cuatrimestre de 2018

Estructuras de Datos

Carolina Pistillo

Índice

Los tipos de datos como `int`, `char`, `bool` se pueden identificar como *tipos de datos simples* porque consisten en un único valor simple, como un número, valor de texto o una configuración verdadera o falsa. Para otros tipos de datos como `string` podría argumentarse que no es un tipo de datos simple, y hasta cierto punto, eso es correcto. C/C++ y otros lenguajes similares consideran que un `string` es un *array* de caracteres. Debido a que este módulo cubrirá los array, vamos a estar de acuerdo con eso. La diferencia clave para C++ es que podemos usar un valor `string` como una estructura de datos simple sin preocuparnos por acceder a él como un array de caracteres.

Los *tipos de datos complejos* son adecuados para escenarios donde necesita almacenar múltiples elementos en una sola entidad. Considere una baraja de cartas, los días de la semana, los meses del año e incluso elementos más complejos, como un objeto en el código para representar un auto. Cada uno de estos ejemplos requiere valores múltiples para reflejar el concepto.

Este módulo se centrará en *arrays*, *enums* y *structs* en C++. Podemos representar la baraja de cartas como un *array*, los días de la semana como un *enum* y el auto como un *struct*. La forma más común de representar "objetos" complejos, como un auto, es usar programación orientada a objetos (POO) y *clases*. Estos temas se verán más adelante.

1. Arrays

Un array es un conjunto de objetos que se agrupan y administran como una unidad. Puede pensar en un array como una secuencia de elementos, todos del mismo tipo. Puede construir arrays simples que tengan una dimensión (una lista), dos dimensiones (una tabla o matriz), tres dimensiones (un cubo o matriz tridimensional). Los arrays en C++ tienen las siguientes características:

- Cada elemento en el array contiene un valor.
- Los arrays tienen índice cero, es decir, el primer elemento del array es el elemento 0.
- El tamaño de un array es la cantidad total de elementos que puede contener.
- Los arrays pueden ser unidimensionales, multidimensionales o irregulares.
- El rango de un array es el número de dimensiones en el array.
- Los arrays de un tipo particular solo pueden contener elementos de ese tipo.

1.1. Declaración

Cuando se declara un array, se especifica el tipo de datos que contiene y un nombre para el mismo. La declaración de un array pone el array al alcance, pero en realidad no asigna ninguna memoria para ello. Más adelante cuando veamos punteros y manipulación de memoria veremos cómo crear un array físicamente.

Para declarar un array unidimensional, especifique el tipo de elementos y el nombre usando corchetes, [ ] para indicar que la variable es un array. El tamaño de un array puede ser cualquier expresión entera. El siguiente ejemplo de código muestra cómo declarar un array unidimensional de enteros de tamaño 5.

```
int numeros[5];
```

1.2. Inicialización

Los arrays pueden inicializarse uno por uno o utilizando una sola sentencia de la siguiente forma:

```
int notas[] = { 1, 2, 3, 4, 5 };
```

Note que al omitir el tamaño del array, se crea un array lo suficientemente grande como para contener la inicialización. Para acceder a cada posición se utiliza la siguiente forma:

```
cout << "El tercer numero del array es " << numeros[2] << endl;
```

Puede iterar a través de un array mediante el uso de un ciclo for. Por ejemplo:

```
int numeros[5] = { 1, 2, 3, 4, 5 };

for (int i = 0; i < 5; i++)
{
   numeros[i]++;
   cout << numeros[i] <<endl;
}
```

1.3. Arrays multidimensionales

Un array puede tener más de una dimensión. El número de dimensiones corresponde al número de índices que se utilizan para identificar un elemento individual en el array, rara vez necesitará más de dos. Las variables de arrays multidimensionales se declaran igual que declara un array de dimensión única, pero se separan las dimensiones mediante el uso de más corchetes. El siguiente ejemplo de código muestra cómo crear una matriz de enteros (dos dimensiones) de 10 filas y 5 columnas, inicializada en 0s.

```
int matriz[10][5];

    for(int i=0; i<10; i++)
    {
        for(int j=0; j<5; j++)
        {
            matriz[i][j] = 0;
            cout<<matriz[i][j]<<" ";
            if(j==4)
            {
                cout<<endl;
            }
        }
    }
```

2. Enums

Una enumeración es una estructura que le permite crear una variable con un conjunto fijo de valores posibles. El ejemplo más común es usar una enumeración para definir los días de la semana. Solo hay siete valores posibles para los días de la semana, y puede estar razonablemente seguro de que estos valores nunca cambiarán.
Por defecto, los valores del `enum` comienzan en 0 y cada miembro sucesivo se incrementa en un valor de 1.

2.1. Creando y usando Enums

Para crear una enumeración, la declara con la siguiente sintaxis, que demuestra la creación de una enumeración llamada Dia, que contiene los días de la semana:

```
enum Dia { LUNES, MARTES, MIERCOLES, JUEVES, VIERNES, SABADO, DOMINGO };
```

Como resultado, la enumeración previa 'Dia' contendría los valores:

```
LUNES = 0
MARTES = 1
MIERCOLES = 2
JUEVES = 3
VIERNES = 4
SABADO = 5
DOMINGO = 6
```

Puede cambiar el valor predeterminado especificando un valor inicial para su enumeración:

```
enum Dia { LUNES = 1, MARTES, MIERCOLES, JUEVES, VIERNES, SABADO, DOMINGO };
```

En este ejemplo, el `LUNES` recibe el valor `1` en lugar del valor predeterminado de `0`. Ahora el `MARTES` es `2`, el `MIERCOLES` es `3`, etc.

Para cambiar el tipo de datos predeterminado de su enumeración, preceda a la lista con un tipo de datos, como por ejemplo:

```
enum Dia: int { LUNES = 1, MARTES, MIERCOLES, JUEVES, VIERNES, SABADO, DOMINGO };
```

2.2. Usando un Enum

Para usar la enumeración, cree una instancia de su variable `enum` y especifique qué miembro `enum` desea usar.

```
Dia diaFavorito = DOMINGO;
```

El uso de enumeraciones tiene varias ventajas:

- Mejora de la capacidad de administración. Al restringir una variable a un conjunto fijo de valores válidos, es menos probable que experimente argumentos inválidos y errores de ortografía.
- Experiencia de desarrollador mejorada. En la mayoria de los IDEs, se le indicará los valores disponibles cuando utilice una enumeración.
- Mejora de la legibilidad del código. La sintaxis enum hace que su código sea más fácil de leer y entender.

3. Structs

Los `struct` son esencialmente estructuras de datos livianas que representan piezas de información relacionadas por un solo elemento. Por ejemplo:

- Una estructura llamada `Punto` puede consistir en campos para representar una coordenada `x` y una coordenada `y`.
- Una estructura llamada `Circulo` puede consistir en campos para representar una coordenada `x`, una coordenada `y`, y un `radio`.
- Una estructura llamada `Color` puede consistir en campos para representar un `componente rojo`, un `componente verde` y un `componente azul`.

3.1. Creando un Struct

Las siguientes opciones son ambas válidas para la creación de un `struct`

Opción 1:

```
struct Alumno
{
    int padron;
    float nota;
};
```

Opción 2:

```
typedef struct
{
    int padron;
    float nota;
} Alumno;
```

3.2. Usando un Struct

Para crear una instancia de un `struct`, use la nueva palabra clave, como se muestra en el siguiente ejemplo:

```
Alumno Matias;
Matias.padron = 88122;
Matias.nota = 9;
```

4. String como estructura de datos simple

C++ proporciona una clase de `string` que se puede usar así:

```
1  #include <iostream>
2  #include <string>
3
4  using namespace std;
5
6  int main()
7  {
8      string s1 = "Hola ";
9      string s2 = "Juan";
10     cout<< s1 + s2 <<endl; //Hola Juan
11
12     return 0;
13 }
```

Referencias

- Introduction to C++ - MIT Open Course
- Algorithms and Data Structures - Microsoft, EdX