

Tipos estructurados en C++

UNIDAD 3

PROGRAMACIÓN AVANZADA

Tipos estructurados en C++

Arreglos y Registros. Declaración. Acceso a elementos. Operaciones básicas. Combinación arreglos y registros.

ARREGLOS

Tipos de Arreglos

UNIDIMENSIONAL

Son utilizados para almacenar múltiples valores en una única variable.

Este tipo de arrays (vectores), permiten almacenar muchos valores en posiciones de memoria continuas, lo cual permite acceder a un valor u otro de manera rápida y sencilla. Estos valores pueden ser números, letras o cualquier tipo de variable que deseemos incluso tipos de datos complejos.

BIDIMENSIONAL

Una matriz es una estructura conformada por filas y columnas, idealmente más de dos filas y columnas.

La intersección de una fila y una columna de la matriz son las casillas y cada una de ellas podrá poseer información, simple o compleja.

Una matriz es un vector cuyas posiciones (de la cero a la n) son, cada una de ellas, otro vector.

Arreglos



Arreglo Unidimensional

Declaración:

La forma de declarar un arreglo unidimensional en C/C++ es:

```
tipoDato nombreArreglo[numeroElementos];
```

tipoDato: Tipo de dato que tendrán los elementos del arreglo.

nombreArreglo: Nombre que asignamos para referirnos al arreglo.

numeroElementos: Valor entero que delimita el tamaño del arreglo.

Arreglo Unidimensional

- En C/C++, los elementos de un arreglo de tamaño N están indexados a partir de cero hasta N-1.
- Para acceder a los elementos individuales de un arreglo, se escribe el nombre del arreglo seguido del subíndice del elemento entre corchetes.
- Ejemplo:

```
int a[3];  
a[0] = 5;  
a[1] = 8;  
a[2] = a[0] * a[1];
```

Arreglos en C++. Declaración



NO USAR LA LIBRERÍA
vector

¡Precaución!

El compilador de C/C++ **no verifica** que los subíndices de los elementos de un arreglo estén dentro de los límites correspondientes.

El programador debe tener cuidado de no sobrepasar estos límites. De lo contrario, es posible que sobrescriba la memoria asignada a otras variables.

Ejemplo:

```
int main() {  
    int a[10], x[10];  
    a[0] = 1;  
    x[10] = 2;//ERROR  
    cout << a[0] << endl;  
    return 0;  
}
```



Arreglo Unidimensional

Inicialización

- **Con** la declaración
`int num[3]={3,6,9};`

Arreglo Unidimensional

Lectura

- Dependiendo el tipo de dato se utiliza la función de entrada correspondiente especificando la posición donde se recibirá el nuevo valor dentro del arreglo

```
Sea int num[3];
```

```
cin>>num[0];
```

Escritura

- La muestra se debe realizar elemento por elemento. Es decir, se debe indicar la posición que ocupa el elemento a mostrar.

```
cout<<num[0];
```

Arreglo Unidimensional

Asignación

- El valor asignado debe pertenecer al tipo de dato del arreglo.
- Puede realizarse
 - A una posición determinada
`num[0]=6;`
 - A todo el arreglo
`vec=num; //vec debe ser del mismo tipo y tamaño que num`

Arreglo Unidimensional

Recorrido

- Para recorrer todo el arreglo se utiliza una estructura repetitiva variando el índice desde 0 a $N-1$, siendo N el tamaño del arreglo.

Arreglo Unidimensional

RECORRIDO CON WHILE

```
int main(){
int i=0,num[3]={3,6,9};
while(i<3) {
    cout<<num[i];

    i++;
}
}
```

RECORRIDO CON FOR

```
int main(){
int i,num[3]={3,6,9};
for(i=0;i<3;i++)
    cout<<num[i];
}
```

Práctica #1

- Realizar un programa que permita cargar los N primeros elementos impares positivos. Mostrarlos en orden inverso al generado.

Cadenas

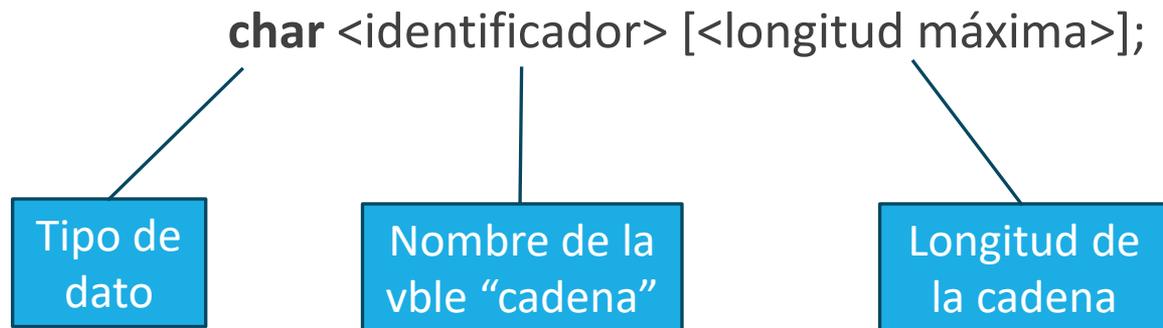
ARREGLO DE CARACTERES



CADENAS

Una **cadena** en C++ es un conjunto de caracteres o valores de tipo **char**, terminados con el carácter nulo, es decir con el valor numérico 0.

La manera de definir una cadena es la siguiente:



CADENAS

Cuando se declara una cadena hay que tener en cuenta que se tiene que reservar una posición para almacenar el **carácter nulo terminador**, de modo que, si se quiere almacenar la cadena "HOLA", se tendrá que declarar la cadena como:

```
char saludo[5];
```

CADENAS

En un programa C++, una cadena puede almacenar información en forma de texto, como nombres de personas, mensajes de error, números de teléfono, etc.

La asignación directa sólo está permitida cuando se hace junto con la declaración.

El siguiente ejemplo producirá un error en el compilador, ya que una cadena definida de este modo se considera una constante.

```
char saludo[5];  
saludo = "HOLA";
```

CADENAS

- La manera correcta de asignar una cadena, luego de definirla es:

```
char saludo[5]; //definición
```

```
saludo[0] = 'H';
```

```
saludo[1] = 'O';
```

```
saludo[2] = 'L';
```

```
saludo[3] = 'A';
```

```
saludo[4] = 0;
```

- O bien, en la definición:

```
char saludo[5] = "HOLA";
```

Lectura de cadenas

- La lectura usual de datos con el objeto *cin* y el operador `>>`, si se aplica a datos de cadena produce anomalías, ya que el objeto *cin* termina la operación de lectura siempre que se encuentra un *espacio en blanco*.
- El método recomendado para la lectura de cadenas es utilizar una función denominada ***getline()***, en unión con *cin*, en lugar del operador `>>`. La función `getline` permite a *cin* leer la cadena completa, incluyendo cualquier espacio en blanco.

```
1  #include <iostream>
2  using namespace std;
3  int main(){
4      char ejemplo[31]; // definición arreglo de caracteres
5      cout << "Escriba una frase \n ";
6      cin.getline(ejemplo,30); // lectura de cadena completa
7      cout << "\t\" << ejemplo << "\"\n";
8      cout << "Escriba otra frase \n ";
9      cin >> ejemplo; // lectura de palabra hasta encontrar blanco
10     cout << "\t\" << ejemplo << "\"\n";
11     system("PAUSE");
12     return 0;
13 }
14
```

Lectura de cadenas

El objeto *cin* del flujo de entrada incluye entre otras las funciones de entrada:

`cin.getline()`: permite a `cin` leer la cadena completa, incluyendo cualquier espacio en blanco

`cin.get()`: se utiliza para leer carácter a carácter

`cin.ignore()`: lee uno o más caracteres del flujo de entrada `cin` sin procesar

Nota: también puede utilizarse `fflush(stdin)` para limpiar el buffer de entrada

Lectura de cadenas

```
1  #include <iostream>
2  using namespace std;
3  bool esvocal(char ch)
4  {
5      ch= toupper(ch);
6      return ( ch == 'A' || ch == 'E' ||ch == 'I' ||ch == 'O' ||ch == 'U');
7  }
8  int main(){
9      char ch;
10     int cuenta = 0;
11     //itera hasta que se teclee el final del flujo de entrada, dado por control + Z.
12     while (cin.get(ch))
13         if (esvocal(ch))
14             cuenta++;
15     cout << " numero de vocales leidas " << cuenta << "\n";
16     system("PAUSE");
17     return 0;
18 }
```

BIBLIOTECA string.h

La biblioteca **<string.h>** contiene un conjunto de funciones para manipular cadenas: *copiar, cambiar caracteres, comparar cadenas, etc.*

Las funciones más elementales son:

strcpy (c1, c2); Copia c2 en c1

strcat (c1, c2); Añade c2 al final de c1

strlen (cadena); Devuelve la longitud de la cadena

strcmp (c1, c2); Devuelve cero si c1 es igual a c2; no-cero en caso contrario

strlwr(c1); Convierte todos los caracteres de la cadena c1 a minúscula

strupr(c1); Convierte todos los caracteres de la cadena c1 a mayúsculas

Para trabajar con estas funciones, al comienzo del programa hay que escribir

```
#include <string.h>
```

CADENAS - EJEMPLO

```
#include <iostream>
#include <string.h>
#include <locale.h>
using namespace std;
int main() {
    setlocale(LC_ALL, "");
    char completo [80];
    char nombre[32] = "José Francisco";
    char apellidos [32] = "de San Martín";
    /* Construye el nombre completo */
    strcpy ( completo, nombre );          /* completo <- "Jose Francisco" */
    strcat ( completo, " ");              /* completo <- "Jose Francisco " */
    strcat ( completo, apellidos );      /* completo <- "Jose Francisco de San Martin" */
    cout<< "El nombre completo es "<< completo ;
    return 0;
}
```

string

El tipo de dato **string** en C++ corresponde a una clase que permite almacenar un conjunto de caracteres como un objeto, además tiene asociados funciones que se usan con dichos datos.

Declaración

```
string nombre_vble;
```

Nota: se debe utilizar la librería `string`

Inicialización

Para definir variables de tipo `string` con valores iniciales se puede usar la siguiente sintaxis:

```
string nombre_variable = "texto";
```

también esta otra forma

```
string nombre_variable ("texto");
```

string

El manejo básico de las cadenas no reviste mayor dificultad:

- Se declaran como cualquier otra variable en C y C++: **string** cadena;
- Se les asigna valor con el signo = , como se hace con los números enteros o reales (o como las cadenas en otros lenguajes).
- Su valor se muestra en pantalla con "cout", igual que para las demás variables.

Declaración y Asignación a un tipo string.

```
1  #include <iostream>
2  #include <string>
3
4  using namespace std;
5
6  int main()
7  {
8  //forma #1 de inicialización
9  //aunque prácticamente no se usa
10 string cad_1("Hola mundo");
11 //forma #2 de inicializar y asignar una cadena
12 string cad_2 = "Segunda forma";
13
14 //asignar/copiar cad_1 en cad_2
15 cad_2 = cad_1;
16
17 //asignar un solo caracter a un tipo string
18 cad_1 = 'P';
19
20 return 0;
21 }
```

Comparaciones entre strings

La comparación entre objetos *string* se puede llevar a cabo fácilmente mediante el uso de los operadores: `==`, `<=`, `>=`, `<`, `>`, `!=`, que son los mismos que se usan para las operaciones de tipo lógicas, hay que aclarar que *se distingue* entre mayúsculas y minúsculas.

```
1  #include <iostream>
2  #include <string>
3
4  using namespace std;
5
6  int main()
7  {
8  //creamos y asignamos las cadenas str1 y str2
9  string str1="abcd", str2="abcd";
10 cout<<endl;
11 //comparamos si las cadenas son iguales
12 if(str1==str2)
13 {
14 cout<<"str1 es igual a str2"<<endl<<endl;
15 }
16
17 return 0;
18 }
```

Concatenación de strings

El operador '+' (más) permite concatenar dos o más cadenas, entonces en el siguiente ejemplo si tengo a str1, str2 que contienen las cadenas "Julio" y "Cesar" respectivamente se puede unir (concatenar) el contenido de str2 a str1 solo poniendo str1 = str1 + str2, o cómo óptimo a la hora de escribir con el operador +=, entonces se tiene str1+=str2 y con esto, la cadena 1 debería contener ambos nombres, el código es:

```
1  #include <iostream>
2  #include <string>
3
4  using namespace std;
5
6  int main()
7  {
8  //definimos e inicializamos ambas cadenas
9  string str1="Julio ", str2="Cesar";
10
11 //realizamos la concatenación de ambas variables
12 str1+=str2;
13
14 //mostramos el nuevo contenido de str1
15 cout<<"El nuevo contenido de str1 es: "<<str1<<endl;
16 return 0;
17 }
```

string. Mayúsculas y Minúsculas. Longitud

`toupper()` // convierte a mayúscula

`tolower()` // convierte a minúscula

Inconveniente: *Lo hace carácter por carácter.*

`length()` // método que permite obtener la longitud de una cadena.

Ejemplo.

```
string cadena="Hola Mundo";
```

```
cout<<"La longitud de cadena es: "<<cadena.length();
```

Práctica #2

Realizar un programa que solicite 2 palabras, las convierta a mayúscula y las muestre ordenadas alfabéticamente.

Arreglo Bidimensional

- A diferencia del arreglo unidimensional, requiere de dos subíndices, para denotar la ubicación de un elemento dentro de la matriz. El primer subíndice indicará la fila, y el segundo representará a la columna.
- La declaración de una matriz en C++ es:

```
tipoDato nombreArreglo[cantidadFilas][cantidadColumnas];
```

Arreglo Bidimensional

`int x[M][N];`

| | | | | | | | |
|---------------------|------------------------|------------------------|--|--|--|--|--------------------------|
| <code>x[0]</code> | <code>x[0][0]</code> | <code>x[0][1]</code> | | | | | <code>x[0][N-1]</code> |
| <code>x[1]</code> | <code>x[1][0]</code> | <code>x[1][1]</code> | | | | | <code>x[1][N-1]</code> |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| <code>x[M-1]</code> | <code>x[M-1][0]</code> | <code>x[M-1][1]</code> | | | | | <code>x[M-1][N-1]</code> |

Arreglo Bidimensional

- **Ejemplo:**

- Se desea implementar un arreglo de 10 x 5 donde el elemento (i,j) sea igual a (i + j).

- **Solución:**

```
int i, j;  
int x[10][5];  
for (i = 0; i < 10; i++)  
    for (j = 0; j < 5; j++)  
        x[i][j] = i + j;
```

Práctica #3

- Escribir un programa que lea una matriz de enteros de 4 filas y 4 columnas y a continuación intercambie la fila i con la fila j , siendo i y j dos valores introducidos por teclado.

Continuará...