Universidad Autónoma de Entre Ríos
Facultad de Ciencia y Tecnología
Sede: Oro Verde

FUNDAMENTOS DE PROGRAMACIÓN

RESUMEN DE CONTENIDOS N°14

Structs

RESUMEN DE CONTENIDOS N° 14

Structs

Introducción

En resúmenes de contenidos anteriores, se comenzó con el estudio de las Estructuras de Datos.

A modo de recordatorio, diremos que:

Una **Estructura de Datos** es un conjunto de datos homogéneos que se tratan como una sola unidad y bajo un mismo nombre colectivo.

En esta unidad, se abocará el estudio de la Estructura de Datos denominada ***Struct***.

En esta unidad, se han vertido conceptos extraídos de "Cómo programar en C/C++", Segunda Edición, de los autores **P. J. DEITEL - H. M. DEITEL** y "Cómo programar en C++", Sexta Edición, de los autores **P. J. DEITEL - H. M. DEITEL**

Struct.

Un arreglo es una estructura de datos homogénea, es decir sólo admite una colección de elementos de igual tipo. A menudo se requiere organizar los datos de una entidad en una estructura, pero admitiendo información de diferente naturaleza (es decir, de diferente tipo). C++ dispone para este caso del tipo **struct.**

*Un **struct** en C++ es una colección de componentes, los cuales pueden ser de diferente tipo. Cada componente o miembro debe declararse individualmente.*

Su sintaxis general es la siguiente:

```
struct compuesta{
          miembro 1;
          miembro 2;
          miembro 3;
          .....
          miembro n;
                };
```

Ahora, observemos el siguiente ejemplo:

ficha
apellido
nombres
Dni
Edad
Cant_materias

```
struct ficha {
      string apellido;
      string nombres;
      long dni;
      int   edad;
      int   cant_materias;
            };
```

La palabra clave **`struct`** introduce la definición de la estructura **`Ficha`**. El identificador **`Ficha`** es el nombre de la estructura y se utiliza en C++ para declarar variables de tipo estructura.

Al definir una estructura, se está planteando el esquema de la composición pero sin definir ninguna variable en particular. El tipo de dato **`struct`**, es justamente eso: **un tipo de dato**.

Con esto quiere significarse que al declarar a una variable de tipo `struct`, se está creando un tipo de dato propio. Así como se tiene int x, dónde x es una variable simple de tipo entero, ahora se tendrá `struct` x {….}, que será una variable compuesta, conformada por diferentes miembros o campos.

La definición de un `struct` no reserva espacio en memoria; en vez de ello, crea un nuevo tipo de datos que se utiliza para declarar variables de estructura. Estas variables se declaran de la misma forma que las variables de otros tipos. Obsérvense las siguientes declaraciones:

```
struct ficha {
      string apellido;
      string nombres;
      long dni;
      int   edad;
      int   cant_materias;
              };
……
int main()
ficha f;
ficha vec_alum[300];
….
```

en las mismas, se declara a `f`, como una variable con una estructura igual a `ficha`; `vec_alum`, como un arreglo con 300 elementos, cada uno de ellos del tipo `ficha.`

Cómo se mencionó anteriormente, los datos declarados dentro de las llaves de la definición de la estructura, son los **miembros**. Los miembros de una misma estructura deben tener nombres únicos (diferentes), pero dos estructuras distintas pueden contener miembros del mismo nombre sin conflicto. Cada definición de estructura debe terminar con un signo de punto y coma.

```
struct ficha {
      string apellido;
      string nombres;
      long dni;
      int   edad;
      int   cant_materias;
              };
```

```
struct empleado {
      string apellido;
      string nombres;
      long dni;
      int   edad;
      float valor_hora;
              };
```

Los miembros de una estructura pueden ser variables de los tipos de datos fundamentales (por ejemplo, int, double, etc.) o agregados, como arreglos u otras estructuras. Por ejemplo, una estructura **`Empleado`** podría contener miembros de cadenas de caracteres para el primer nombre y el apellido paterno, un miembro int para la edad del empleado, un miembro char que contenga 'M' o 'F' para el género del empleado, y así en lo sucesivo.

Inicialización de estructuras

Para declarar variables de tipo `struct` se debe indicar lo siguiente:

```
ficha x,y;  // x e y se declaran de tipo ficha
```

Se puede combinar la definición de la composición de la estructura con la declaración de las variables y con su inicialización:

```
struct ficha{
       string apellido;
       string nombres;
       long dni;
       int edad;
       int cant_materias;
            } x = {“Lopez”, “Gerardo”, 24567890, 21, 11};
```

En este ejemplo, `ficha` es el nombre de la estructura, `x` la variable de tipo ficha y los datos especificados entre { }, los valores iniciales de los miembros `apellido, nombres, dni, edad, cant_materias.`

Procesamiento de una variable struct

Para procesar la información relacionada a una estructura se puede operar con sus miembros individualmente o en ocasiones con la estructura completa. Para acceder a un miembro individual se debe utilizar el identificador de la variable **`struct`**, un punto de separación y el nombre del miembro componente:

`variable.miembro`

Considérese el siguiente código de ejemplo.

```
1. #include <iostream>
2. using namespace std;
3. //-------------------------------------------------------
4. struct registro{
5.      sttring ape;
6.      string nom;
7.      long dni;
8.                 };
9. //-------------------------------------------------------
10. int main( )
11. {
12.  registro f,z;
13.  getline(cin, f.ape);
```

```
14.  getline(cin, f.nom);
15.  cin>>f.dni;
16.  z=f;
17.  cout<<z.ape<<" "<<z.nom<<"---DNI:"<<z.dni<<endl;
18.  return 0;
19. }
```

Obsérvese en la línea 12 la declaración de las variables tipo struct **f** y **z**. Las líneas 13, 14 y 15 permiten asignar datos ingresando los valores de los miembros en modo consola. En la línea 16 se asigna la variable struct completa **f** a una variable de igual tipo **z**. En la línea 17 se muestran los miembros de **z**.

Arreglo de structs

Es posible emplear arreglos como miembros de una composición **`struct`**. Pero también podemos definir un arreglo cuyos elementos sean estructuras (**`struct`**).

```
struct ficha {
       string apellido;
       string nombres;
       long dni;
       int edad;
       int cant_materias;
                  };

ficha z[5] ={
           {"Lopez","Gerardo",24567890, 21, 11},
           {"Giménez","Ana",25689901,20},
           {"Zapata","Andrés", 26701231,19,8},
           {"Farías","Marina",23199870,22,15},
           {"Martino","Manuel",24500654,21,10},
     };
```

En el caso anterior se declara e inicializa un arreglo **z** de 5 elementos de tipo ficha. Para mostrar por pantalla el miembro `dni` del tercer elemento del arreglo, deberá escribirse:-

```
cout<<z[2].dni;
```

Para cambiar el miembro edad a 22 en el cuarto elemento del arreglo **z**, deberá escribirse:-

```
z[3].edad=22;
```

Para leer los apellidos, nombres y dni en modo consola, deberá escribirse:

```
for (int i=0; i<5; i++)
{
     getline(cin, z[i].apellido);
     getline(cin, z[i].nombres);
```

```
      cin>>z[i].dni;
    }
```

Observación: cin.getline() es una función que permite el correcto ingreso de una cadena de caracteres. Esta función necesita como mínimo 2 parámetros: el nombre del string y la cantidad de caracteres del mismo.

Para mostrar un listado con los miembros apellido y dni del arreglo **z**, se codifica de la siguiente forma:

```
for (int i=0; i<5; i++)
{
     cout<<z[i].apellido<<" "<<z[i].dni<<endl;
}
```

Otros ejemplos con struct

```
/* Declaración de un struct Direccion con el esquema para contener
calle y número y otro struct Persona que, entre otros miembros,
contiene uno del tipo Direccion */

#include <iostream>
#include <iomanip>
using namespace std;

int main() {
     struct Direccion {
          string calle;
          int numero;
     };
     struct Persona {
          string apeynom;
          Direccion domi;
          } candidato;

/* Se define una variable candidato del tipo Persona. Para el
ingreso de calle y número, se debe referenciar de la variable
candidato el elemento domi (tipo Direccion) y del mismo, calle y/o
numero */

     cout<< "Ingrese apellido y nombre: ";
     getline(cin, candidato.apeynom);
     cout<< "Ingrese Calle donde vive: ";
     getline(cin, candidato.domi.calle);
     cout<< "Ingrese numeración: ";
     cin>> candidato.domi.numero;
     cout<< "Candidato " << candidato.nombre << endl;
```

```
     cout<< "Domicilio " << candidato.domi.calle << ' ' <<
candidato.domi.numero;
     return 0;
}


/* Declaración de un struct Libro con el esquema para contener datos
de un libro y de la variable colección que es un array de Libro */
#include <iostream>
#include <iomanip>
using namespace std;

int main() {
     struct Libro {
          string titulo;
          string editorial;
          int anio_edicion;
     };

     Libro coleccion[10];

     for (int i=0; i< 10; i++)
     {
          cout<< "Datos del libro : " << i << endl;
          cout<< "Ingrese titulo: ";
          getline(cin, coleccion[i].titulo);
          cout<< "Ingrese editorial:  ";
          getline(cin, coleccion[i].editorial);
          cout<< "Ingrese año edición: ";
          cin>> coleccion[i].anio_edicion;
     }
     cout<< "Listado de Libros " << endl;
     cout<< " Titulo           Editorial       Año edición " <<
endl;
     for (int i=0; i< 10; i++)
     {
       cout<< coleccion[i].titulo<<"    "<<
coleccion[i].editorial<<"    "<< coleccion[i].anio_edicion<< endl;
     }
     return 0;
}
```

Tipos definidos por el usuario: typedef

En C++ es posible identificar a tipos existentes o estructuras de datos con nombres y tratarlos como si fueran nuevos tipos de datos. Entonces, las variables o funciones podrán declararse en base a estos nuevos nombres que representan tipos de datos de C++. Estos tipos de datos definidos por el usuario no tienen la importancia que revisten en otros lenguajes.

Para asignar un nombre correspondiente a una definición de tipo en C++ deberá emplearse la palabra reservada **`typedef`**.

A continuación, algunos ejemplos:

```
typedef unsigned char byte;
/*se le da el alias byte a los char sin signo */

typedef int tabla[50][12];
/*definición del nombre tabla como matriz de 50x12 enteros */


typedef struct {
        string apellidos;
        string nombres;
        long dni;
        int edad;
        int cant_materias;
            } ficha // definición del tipo ficha como estructura
```

En base a las definiciones de tipo anteriores, son válidas las siguientes declaraciones:

```
byte b;
ficha x,y[200];
tabla m,t;
```

En este ejemplo, **b** se declara de tipo byte; **x** es una variable individual que puede almacenar la información de una entidad de acuerdo a los miembros definidos en el tipo **struct ficha;** la variable **y** es un **array** donde cada **elemento** es del tipo **struct ficha**.; las variables **m** y **t** se declaran como arreglos bidimensionales de 50x12 elementos enteros.

Síntesis de Estructuras de Datos (Arrays y Struct)

1. Un arreglo es una colección de datos relacionados de igual tipo e identificados bajo un nombre genérico único.
2. La estructura completa de un arreglo ocupa un segmento de memoria único y sus elementos se almacenan en forma contigua.
3. Para procesar un elemento del arreglo, se debe especificar el nombre del arreglo y uno o más índices que determinan la posición relativa del elemento.
4. El índice que determina la posición de los elementos de un arreglo comienza siempre con cero.
5. En la declaración se debe establecer la cantidad de elementos (dimensión) que puede tener como máximo el arreglo en el programa o definir los elementos que lo componen.
6. La dimensión de un arreglo es la cantidad de elementos que un arreglo puede almacenar y debe ser siempre mayor o igual a su longitud.
7. Si se requieren dos índices para establecer la posición de un elemento, el arreglo es una tabla o matriz. Con 3 índices es una matriz tridimensional y con más índices, una matriz multidimensional.
8. El nombre de un arreglo representa la dirección de memoria del inicio del arreglo (de su primer elemento).

9. Un struct es una estructura de datos de C++ que permite organizar un conjunto de datos de diferentes tipos relacionados. Estos datos que conforman el struct se denominan miembros.
10. Los miembros de un struct pueden ser de tipo simple o también otras estructuras de datos: arreglos, structs, etc.
11. También los elementos de un arreglo pueden ser de tipo struct.
12. Los objetos cin y cout para flujos de entrada y salida no permiten operar estructuras de datos completas. Se deben utilizar los componentes de una estructura, como el elemento de un arreglo o un miembro de un struct.
13. C++ admite la identificación de tipos a través de la palabra clave `typedef`. Estas definiciones suelen ser globales y se realizan usualmente fuera de la función principal de programa `main()`.