Programación II
Tecnicatura de Video Juegos

Clase - Estructuras

Profesor Adjunto : Ing. Mauricio Prinzo

Agenda

- ✓ Temario
  - ⌗ Introducción
  - ⌗ Sintaxis
  - ⌗ Operaciones
  - ⌗ Funciones
  - ⌗ Estructuras anidadas
  - ⌗ Estructuras con arrays
- ✓ Ejercitación

Introducción

- ✓ En un array, almacenamos datos del mismo tipo.

Valores:	45	34	67	88	.....	
	1	2	3	4		99

- ✓ Y si queremos almacenar distintos tipos de datos????

Empleado	Juan	Perez	Rivadavia 1300	05/03/1982	$ 10.000

Estructuras

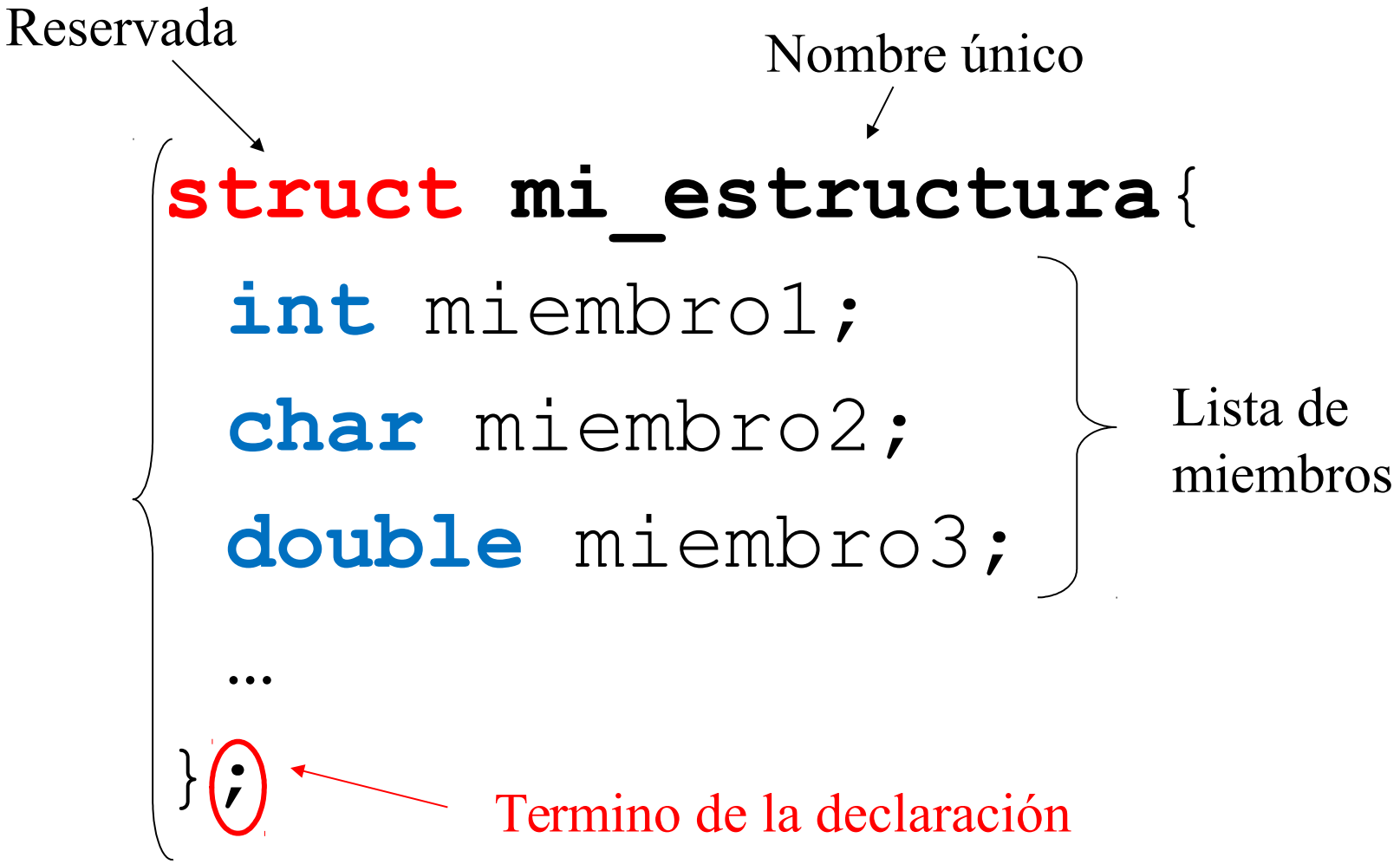
- ✓ Una estructura es una agrupación de datos relacionados lógicamente, y posiblemente de diferentes tipos.
- ✓ Una estructura puede verse como una colección de variables que se referencia bajo un nombre en común.
- ✓ Cada una de estas variables se denominan **"miembros"** de la estructura. Otras denominaciones son:
  - Campo
  - Elemento
  - Atributo
- ✓ Las estructuras son llamadas también muy a menudo registros, o en inglés records.

Declaración

- La definición de una estructura se realiza fuera de cualquier función, generalmente en la parte superior del archivo.

- Para definir una estructura requerimos:
  - Un nombre
  - Una lista de miembros
    - Nombre
    - Tipo

Sintaxis –Declaración de Estructura



Sintaxis - Uso de Estructuras

- ✓ Para utilizarla hay que definir variables del tipo “estructura”.

- ✓ Para definir estas variables se utiliza la siguiente sintaxis:

```
struct Nombre_Estructura Nombre_Variable;
```

O

```
Nombre_Estructura Nombre_Variable;
```

Ejemplo

```
//creo la estructura con los distintos tipos de datos que necesito
struct DatosPersona
      {
       string Nombre;
       char Inicial;
       int   Edad;
       float Nota;
   } ;

//para utilizar la estructura defino una variable del tipo struct
 struct DatosPersona Alumno1;
 // tambien se puede declarar :
 //DatosPersona Alumno1;
```

Acceso a miembros de Estructuras

- ✓ Para acceder a los valores de los miembros de una variable de tipo estructura se utiliza el operados unario “.”.
- ✓ Cada miembro es una variable común y corriente.

```
struct DatosPersona Alumno1;

 //Para acceder a los miembros, utilizo el nombre de la variable esstructura y "." a cada campo
   Alumno1.Nombre = "Juan";
  Alumno1.Nota = 7.5;
   cout << "El nombre del Alumno es" << Alumno1.Nombre << endl;
```

Ejemplo Completo

```
#include <iostream>
#include <string>

using namespace std;

int main()
{
//creo la estructura con los distintos tipos de datos que necesito
struct DatosPersona
{
      string Nombre;
      char Inicial;
      int   Edad;
      float Nota;
   } ;

//para utilizar la estructura defino una variable del tipo struct
 struct DatosPersona Alumno1;
 // tambien se puede declarar : DatosPersona Alumno1;

 //Para acceder a los miembros, utilizo el nombre de la variable estructura y "." a cada campo
   Alumno1.Nombre = "Juan";
   Alumno1.Inicial = 'J';
   Alumno1.Edad = 20;
   Alumno1.Nota = 7.5;
    cout << "El nombre del Alumno es" << Alumno1.Nombre << endl;
    cout << "La edad es " << Alumno1.Edad << endl;
    cout << "EL promedio es " << Alumno1.Nota << endl;
     system("pause");

 return 0;
}
```

Simplificación

- **Sin Estructuras**

```
char NombreAlumno [64];
int EdadAlumno;
double PromedioAlumno;
```

✓ Con estructuras

```
struct Alumno{
   char Nombre[64];
   int Edad;
   double Promedio;
};
```

Operaciones con estructuras

- ✓ **Limitaciones:**
  - Las operaciones de:
    - # **I/O**

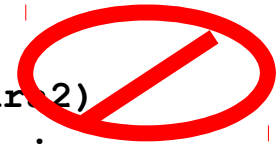
```
cout << Estructura1;
cin >>  Estructura2;
```

    - # **Aritméticas**

```
Estruct1 = Estruct1 + Estruct2
```

    - # **Comparación**

```
if (Estructura1 < Estructura2)
                cout << ...;
```



- ✓ **No tiene sentido a nivel de estructuras, solo se realizan a nivel de miembros.**

```
EJ: if (Alumno1.Nombre ==  Alumno2.Nombre)
```

Estructuras y funciones

- Una función puede devolver una estructura o un puntero-a-estructura:

```
MiEstructura F1(void);      // Devuelve Estructura
struct myst F2(void);       // Devuelve Estructura
MiEstructura* F3(void);     // Devuelve Puntero a Estructura
```

- Una estructura puede ser pasada como argumento a una función de varias formas:

```
void F1(MiEstructura s);                  // Directamente (por valor)
void F2(MiEstructura* puntero);           // Vía puntero (por valor)
void F3(MiEstructura& referencia);        // Indirectamente (por referencia)
```

Estructuras y funciones

- **Para pasar miembros de una estructura a una función, se utiliza el mismo esquema de las variables comunes.**

```
void MostrarNota(int nota);
int ValidarNota(int &nota);
…
Struct alumno a1;
if(ValidarNota(&a1.nota))
   MostrarNota(a1.nota);
```

Estructuras y funciones

- **Para pasar estructuras completas como parámetros se debe especificar el tipo completo de la estructura en la definición del parámetro.**

```
struct Datos
{
// Estos datos no se pueden inicializar
int anio;
int mes;
int dia;
};

// Prototipos de función
void Recibe( Datos &s);
void MuestraxReferecia( Datos &t);
void MuestraxValor( Datos Nacimiento);
```

Ejemplo

```
#include <iostream>
using namespace::std;

struct Datos
{
// Estos datos no se pueden inicializar
int anio;
int mes;
int dia;
};
// Prototipos de funcion
void Recibe( Datos &s);
void MuestraxReferecia( Datos &t);
void MuestraxValor( Datos Nacimiento);

int main()
{
// Declaracion de Elisa como tipo Datos
struct Datos TusDatos;

// Se reciben los datos  mediante la funcion Recibe
Recibe(TusDatos);

// Se imprimen los datos  desde la funcion MuestraxReferencia
cout <<"\nLa fecha de su nacimiento por referencia: "<<endl;
MuestraxReferecia(TusDatos);
// Se imprimen los datos desde la funcion MuestraxValor
cout <<"\nLa fecha de su nacimiento por valor:" <<endl;
MuestraxValor(TusDatos);
cout <<"\nLa fecha de  su nacimiento desde main." <<endl;
cout <<TusDatos.dia <<"/" << TusDatos.mes << "/" << TusDatos.anio << endl << endl;
system("pause");
return 0;
}
```

```
void Recibe( Datos &s)
{
cout << "\nIntroduzca el anio de nacimiento: " <<endl;
cin >> s.anio;
cout << "\nIntroduzca el mes de nacimiento: " <<endl;
cin >> s.mes;
cout <<"\nIntroduzca el dia de nacimiento: " <<endl;
cin >> s.dia;
}

void MuestraxReferecia( Datos &t)
 {  cout <<t.dia <<"/"<<t.mes<<"/"<<t.anio<<endl;
return;
}

void MuestraxValor( Datos Nacimiento)
{ cout << Nacimiento.dia <<"/" <<Nacimiento.mes<<"/"<< Nacimiento.anio << endl;
return;
}
```

Funciones y Arrays

- **Retorno de Función**

- Los Sí
  - Puede no retornar nada (void)
  - Se puede no indicar retorno, en ese caso por defecto es int
  - Puede retornar casi cualquier tipo de dato

- Los No
  - No puede retornar un array
  - No puede retornar un vector

Estructuras anidadas

- **Los miembros de una estructura pueden ser tipos de datos estructurados, es decir:**
  - Otras estructuras
  - Arreglos

- **Estas estructuras se denominan anidadas.**

- **Incuso pueden ser estructuras recursivas.**

Ejemplo – Anidadas con Arreglos

```
#include <iostream>
#include <string>

using namespace std;

struct Alumno{
string Nombre;
string Apellido;
string Materia;
double notas[3];  // defino un vector dentó de la estructura
};

double promedio(struct Alumno a);

int main()
{
struct Alumno Alumno1;
cout <<"Ingrese el nombre del alumno "<< endl;
cin >> Alumno1.Nombre ;
cout <<"Ingrese el Apellido del alumno "<< endl;
cin >> Alumno1.Apellido ;
cout <<"Ingrese la materia el alumno "<< endl;
cin >> Alumno1.Materia;
cout <<"Ingrese la 1er Nota materia el alumno "<< endl;
cin >> Alumno1.notas[0];
cout <<"Ingrese la 2da Nota materia el alumno "<< endl;
cin >> Alumno1.notas[1];
 cout <<"Ingrese la 3er Nota materia el alumno "<< endl;
cin >> Alumno1.notas[2];
cout << "El promedio es " << promedio(Alumno1) << endl;
system("pause");
   return 0;
}

double promedio(struct Alumno a){
return (a.notas[0] + a.notas[1] + a.notas[2])/3.0;
}
```

Ejemplo – Anidadas con Estructuras

```
#include <iostream>
#include <string>
using namespace std;

struct FechaNacimiento
{
   int  dia;
   int  mes;
   int anyo;
};

struct DatosPersona
{
   string nombre;
   struct FechaNacimiento DiaDeNacimiento;
   float nota;
};

int main()
{
   DatosPersona persona;

   persona.nombre = "Carlitos";
   persona.DiaDeNacimiento.mes = 8;
   persona.nota = 7.5;
cout << "La nota es " << persona.nota <<endl;
   system("pause");
   return 0;
}
```

Arrays de estructuras

- **Se puede crear arreglos cuyos elementos sean variables de estructura.**
- **Se definen de manera similar al caso común.**

```
Tipo arreglo[N]

struct Estructura arreglo[N];
```

Ejemplo

```
// Array de registros

#include <iostream>
#include <string>
using namespace std;
;
int main()
{
   struct datosPersona
   {
      string nombre;
      int   edad;
      float nota;
   };

   datosPersona  persona [50]; // creo un vector de esturcturas

   for (int i=0; i<5; i++)
   {
      cout << "El nombre de la persona? " << i << endl;
      cin >> persona[i].nombre;
   }

   cout << "La persona 3 es " << persona[2].nombre << endl;
 system("pause");
   return 0;
}
```

Búsqueda de Estructuras

- **La búsqueda de estructuras es similar a la búsqueda de datos simples.**
- **Existen dos detalles importantes:**
  - Definir el concepto de igualdad entre estructuras
    - No se puede usar “= =“
    - Puede ser a través de un campo
    - Puede ser a través de varios campos
  - Definir valor “no encontrado”

Ordenamiento de estructuras

- **Al igual que en las busquedas, el procedimiento es similar.**
- **Solo falta definir la relación de orden**
  - Puede estar definida por un solo campo
  - Puede estar defnida por varios campos
    - Por lo general, se define un campo principal y otro para el “desempate”.

Ejemplo

```
struct Album{
	char Grupo[32];
	string titulo[32];
	int precio;
};
void Bubblesort_up(struct album Coleccion[]){
int i,j;
	for(i=1;i<N;i++)
		for(j=0;j<(N-i);j++)
		if(Coleccion[j].precio>Coleccion[j+1].precio){
		struct album aux = Coleccion[j+1];
		Coleccion[j+1] = Coleccion[j];
		Coleccion[j] = aux;
	}
}
```

¿Preguntas?