Informática I para Bachillerato

C/C++

Estructuras de Datos

José Luis Alonzo Velázquez

CIMAT

Sesión 15

Estructuras

Una estructura es un grupo de variables las cuales pueden ser de diferentes tipos sostenidas o mantenidas juntas en una sola unidad, a diferencia de los arreglos que solo pueden contener un mismo tipo de dato.

Las estructuras de datos se emplean con el objetivo principal de organizar los datos contenidos dentro de la memoria de la PC. Así, nuestra primera experiencia con estructuras comienza desde el momento mismo en que usamos en nuestros programas variables de tipos primitivos (char, short, int, float, etc).

Sintaxis

En C/C++ se forma una estructura utilizando la palabra reservada **struct**, seguida por un campo etiqueta opcional conocida como rótulo de la estructura, y luego una lista de miembros dentro de la estructura. La etiqueta opcional se utiliza para crear otras variables del tipo particular de la estructura:

```
1 struct [ <nombre tipo de estructura > ] {
2     [<tipo> <nombre-variable[,nombre-variable,...] >];
3     [<tipo> <nombre-variable[,nombre-variable,...] >];
4     ...
5     } [ <variables de estructura> ] ;
```

Un punto y coma finaliza la definición de una estructura puesto que ésta es realmente una sentencia C/C++.

Sintaxis: Caso uno, estructura anónima

De acuerdo con la sintaxis general de la orden struct es posible crear estructuras de datos anónimas. Solamente hay que tener en cuenta que en una declaración anónima se debe definir al menos una variable al final de la declaración. Por ejemplo, con el siguiente fragmento de código:

Ver código 1

```
1 struct {
2     int a;
3     int b;
4 } p1;
```

se declara y define la variable estructurada p1, misma que se compone por los miembros a y b; ambos del tipo int.

Ahora bien, la sintaxis mostrada no es tan común ni conveniente, ya que con la misma solamente se esta creando una variable estructurada pero no un nuevo tipo. Es decir, si deseáramos tener otra variable que tuviera las mismas características que posee la variable p1, necesitaríamos escribir exactamente la misma instrucción, salvo que cambiando el nombre de la variable. Por ejemplo:

```
1 struct { int a, b; } p2;
```

Por supuesto, en una misma línea de instrucción podemos definir más de una variable. Ejemplo:

```
1 struct { int a, b; } p1, p2;
```

Entonces, para crear nuevos tipos con struct deberemos de modificar la sintaxis mostrada en los ejemplos anteriores.

Sintaxis: Caso dos, estructura con nombre

Observe que, la sintaxis para declarar estructuras con nombre es bastante parecida a la sintaxis para declarar estructuras anónimas; salvo que una declaración de estructura con nombre se debe especificar el nombre deseado para la misma. Además, en una declaración de estructura con nombre la o las variables definidas al final de la misma son opcionales.

```
1 struct pareja { int a, b; } p1;
```

Observación

En el fragmento de código anterior se declara la estructura identificada como pareja, misma que se compone de los miembros a y b, ambos de tipo int. En el mismo ejemplo, se define la variable p1; la cual es una variable estructurada de tipo pareja.

Uso del nombre

Una vez que una estructura con nombre ha sido creada, la misma puede ser usada para declarar cualquier número de variables. Por ejemplo, en el siguiente fragmento de código se crea la estructura tiempo compuesta por los miembros hora, minuto y segundo; todos del tipo int. En el mismo ejemplo, se declaran las variables t1 y t2.

```
1 /* declaración de estructura tiempo */
2 struct tiempo { int hora, minuto, segundo; };
3
4 /* declaración de variables de tipo tiempo */
5 struct tiempo t1, t2;
```

Nota

En C++ puede obviarse la palabra struct a la hora de declarar variables. Así, en C++ la línea de instrucción struct tiempo t1, t2; (del ejemplo anterior) puede escribirse como: tiempo t1, t2;

Acceso a los miembros de una estructura

En orden de poder leer o escribir uno de los miembros de una variable estructurada se debe usar el operador de acceso (.), o sea, el nombre de la variable seguida por un punto seguido por el nombre del miembro o componente deseado de la estructura. Por ejemplo, para acceder a los miembros de la variable t1 (mostrada arriba) podemos hacerlo de la siguiente manera:

Ejemplo

```
1 t1.hora = 12;
2 t1.minuto = 0;
3 t1.segundo = 0;
4
5 printf ("%i\n", t1.hora);
```

Estructuras anidadas

Los miembros de una estructura pueden ser ellos mismos otra estructura previamente identificada o bien una estructura anónima. Por ejemplo, en el siguiente fragmento de código se crean las estructuras pareja y pareja2. Obsérvese cómo dentro de los miembros de pareja2 se declara el miembro X, mismo que es una estructura del tipo pareja. Luego, las variables declaradas a raíz de la estructura pareja2 poseerán los miembros variables a y b heredados de pareja, y c.

```
1 struct pareja { int a, b ; };
2 struct pareja2 { struct pareja X; int c; } P3;
```

Acceso en este caso

Ahora bien, para acceder a los miembros de una estructura dentro de otra estructura se emplea el mismo mecanismo de acceso (el punto). Por ejemplo, para desplegar el miembro a de la variable P3 declarada en el ejemplo anterior, lo haremos más o menos así:

```
1 printf( "%d\n", P3.X.a );
```

Declaración de una estructura

```
1  #include <stdio.h>
2
3  struct punto{
4      int x;
5      int y;
6  };
7  struct linea{
8      punto p1;
9      punto p2;
10 };
11
12 int main(){
13     return 0;
14 }
```

Ejemplo de uso de las estructuras descritas

```
1 int main(){
2     linea l1;
3     l1.p1.x=1;
4     l1.p1.y=1;
5     l1.p2.x=2;
6     l1.p2.y=2;
7     printf("La linea l1 pasa por los puntos (%d,%d),(%d
      ,%d)\n",l1.p1.x,l1.p1.y,l1.p2.x,l1.p2.y);
8     return 0;
9 }
```

Herencia

El termino herencia se usa con gran frecuencia en Programación Orientada al Objeto, y se le relaciona principalmente con las clases. Sin embargo, la herencia está presente siempre y cuando una estructura "struct" posea a otra estructura. En ese sentido, en C++ se presentan dos tipos de herencia:

- herencia por agregación o composición.
- herencia por extensión.

Por ejemplo, en la definición de las estructuras pareja y pareja2, se dice que pareja2 hereda por composición todos los miembros de pareja. Ahora, en el siguiente ejemplo se usa la sintaxis para que la estructura pareja2 herede por extensión los miembros de pareja:

```
1 // solo C++
2 struct pareja { int a, b ; };
3 struct pareja2 : pareja { int c; } P3;
```

typedef

La palabra reservada **typedef** proporciona un mecanismo para la creación de sinónimos para tipos de datos anteriormente definidos. Por lo cual los nombres de los tipos de estructura se declaran comúnmente utilizando typedef.

Por ejemplo:

```
1 typedef struct pareja2 Pareja;
```

typedef

Hay que advertir que en C++ es muy frecuente utilizar typedef en la declaración de estructuras. De hecho, los ficheros de cabecera de los compiladores C++ están repletos de ellos. Es muy frecuente que utilicen expresiones como:

```
1 typedef struct {
2   unsigned char *curp;       // Current active pointer
3   unsigned char *buffer;     // Data transfer buffer
4   int level;                 // fill/empty level of
      buffer
5   ...
6   unsigned char token;       // Used for validity
      checking
7 } FILE;                      // This is the FILE object
```

Necesario???

Es posible crear un typedef al mismo tiempo que se declara una estructura, con o sin nombre, como se ve en los ejemplos. Generalmente no se necesitan un typedef y un nombre al mismo tiempo, ya que cualquiera de ellos sirve para las declaraciones.

```
1 typedef struct mystruct { ..;..; } MST;
2 MST s, *ps, arrs[10]; // igual que struct mystruct s,
      etc.
3 typedef struct { ..; ..; } YST;  // sin nombre
4 YST y, *yp, arry[20];
```

Crear un tipo de dato **estudiante** que contenga:

- nombre(hasta 30 caracteres)

punto extra 3er Parcial

Crear un programa que permita meter los datos de 5 estudiantes, y que imprima todos los datos de un estudiante al ingresar la matricula del estudiante.

Crear un tipo de dato **estudiante** que contenga:

- nombre(hasta 30 caracteres)
- matricula(hasta 5 dígitos)

punto extra 3er Parcial

Crear un programa que permita meter los datos de 5 estudiantes, y que imprima todos los datos de un estudiante al ingresar la matricula del estudiante.

Crear un tipo de dato **estudiante** que contenga:

- nombre(hasta 30 caracteres)
- matricula(hasta 5 dígitos)
- carrera(hasta 30 caracteres)

punto extra 3er Parcial

Crear un programa que permita meter los datos de 5 estudiantes, y que imprima todos los datos de un estudiante al ingresar la matricula del estudiante.

Crear un tipo de dato **estudiante** que contenga:

- nombre(hasta 30 caracteres)
- matricula(hasta 5 dígitos)
- carrera(hasta 30 caracteres)
- semestre(un dígito)

punto extra 3er Parcial

Crear un programa que permita meter los datos de 5 estudiantes, y que imprima todos los datos de un estudiante al ingresar la matricula del estudiante.

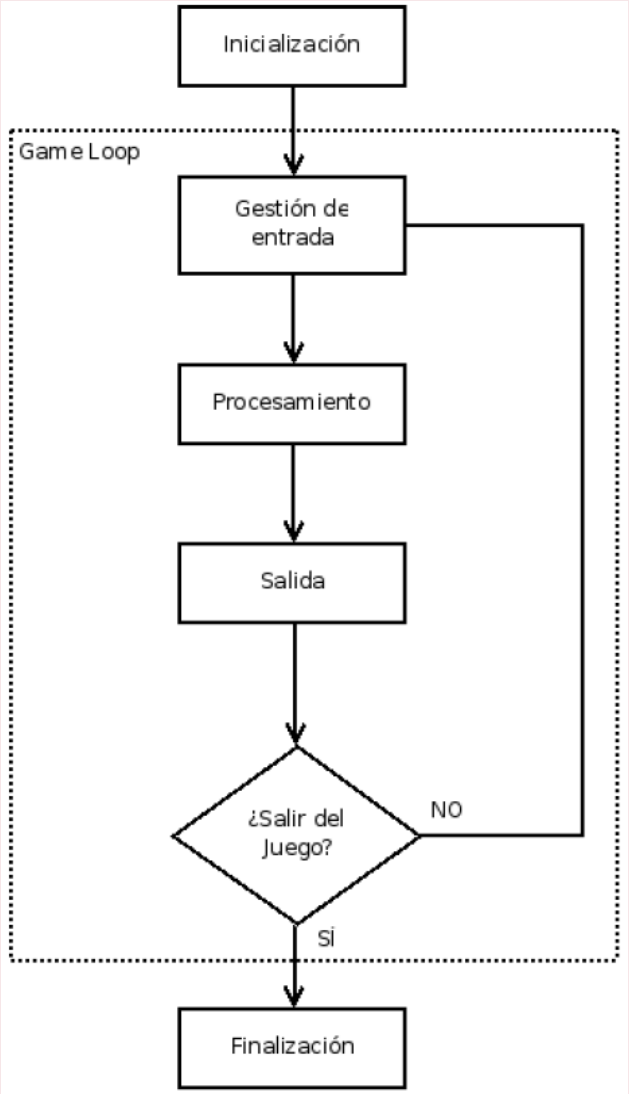
Crear un tipo de dato **estudiante** que contenga:

- nombre(hasta 30 caracteres)
- matricula(hasta 5 dígitos)
- carrera(hasta 30 caracteres)
- semestre(un dígito)
- promedio(con decimales)

punto extra 3er Parcial

Crear un programa que permita meter los datos de 5 estudiantes, y que imprima todos los datos de un estudiante al ingresar la matricula del estudiante.

Estructura de un videojuego



La función básica de un juego seria la siguiente:

```
1 Mientras (no finalice){
2     Procesar_Entradas();
3     Actualizar_Graficos();
4     Actualizar_Sonido();//si hay sonido
5     Mostrar_Graficos();
6     Reproducir_Sonidos();//si hay sonido
7 }
```

Puyo puyo

Puyo Puyo es un juego de rompecabezas fue inspirado por ciertos elementos de los juegos Tetris y Dr. Mario. El objetivo del juego es derrotar al oponente en una batalla llenando su área de juego hasta la parte superior de la pantalla con basura.

- Los Puyos, pequeñas criaturas gelatinosas con ojos, caen desde la parte superior de la pantalla en pares de distintos colores.
- El par puede moverse de izquierda a derecha, y rotarse 90° en sentido de las manecillas del reloj o en sentido contrario.
- La pareja cae hasta tocar el suelo del área de juego u otro par de puyos, siguiendo las leyes de la gravedad.
- El par se rompe, de modo que el otro puyo cae libremente hasta que el caigan en otro puyo o en la parte inferior de la pantalla.

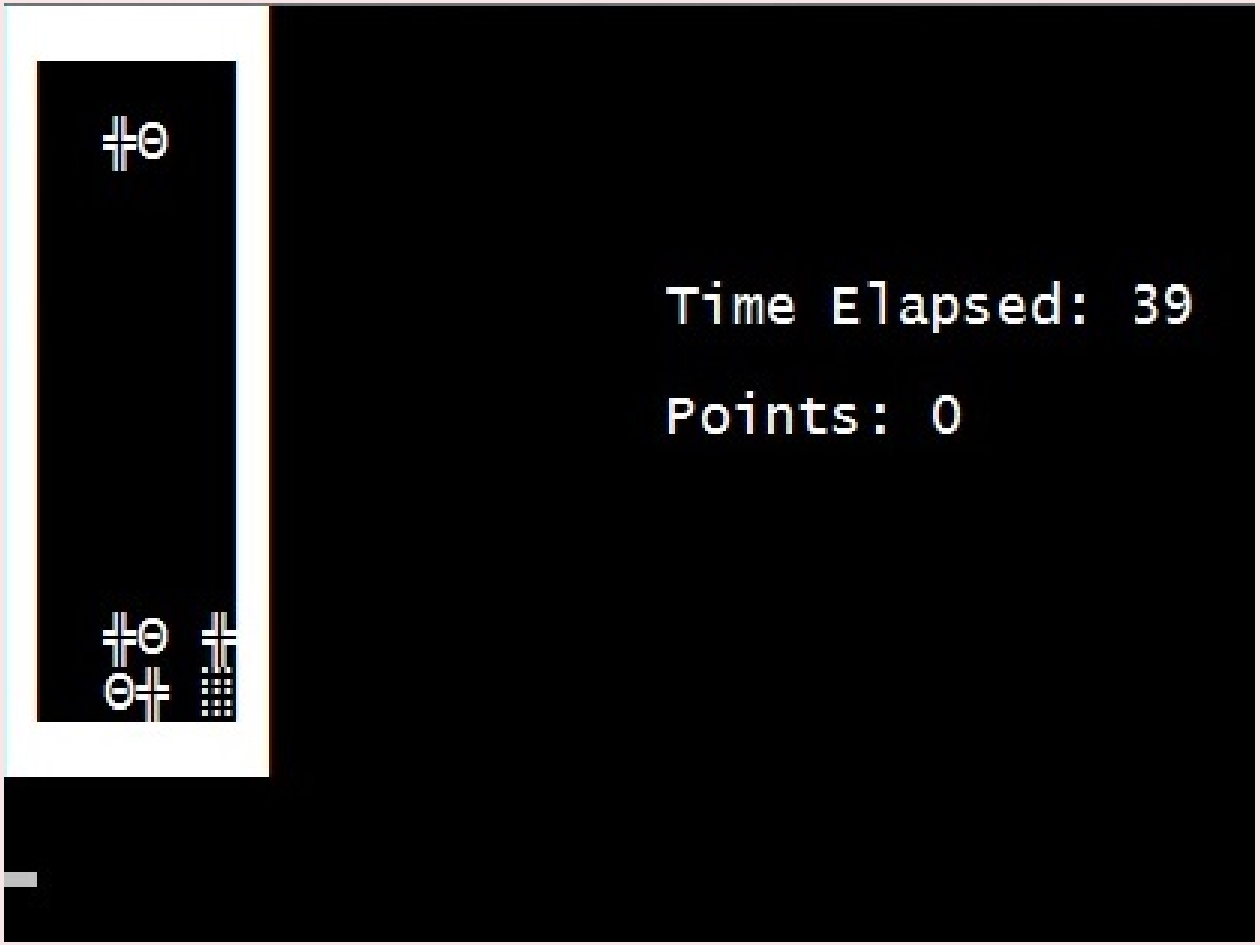
Puyo puyo

- Cuando cuatro o más puyos del mismo color forman un grupo en el que están unidos horizontal o verticalmente, explotan y desaparecen. A esto se le llamada Cadena.
- Cuando los puyos desaparecen, aquellos que están encima de ellos caen hasta aterrizar en otras piezas o en la parte inferior de la pantalla.
- Los Combos se crean del mismo modo, cuando se forma un grupo de más de cuatro puyos, o más de un grupo se forma al tiempo. Todos los puyos del "combo" son borradas al mismo tiempo.
- Las Reacciones en Cadena, se crean cuando los puyos que caen, ya por acción del jugador o por unos puyos que caen de una cadena anterior, causan una reacción en cadena donde los grupos de puyos se borran, uno por vez.

Puyo puyo

Hay más reglas y opciones pero por el momento intentaremos programar estas.

Versión simplificada de Puyo puyo



Crear la función goto_xy

```
1  #include <stdio.h>
2  #include <windows.h>
3  int goto_xy (int x, int y){
4      COORD coord;
5      coord.X = x;
6      coord.Y = y;
7      SetConsoleCursorPosition(GetStdHandle(
       STD_OUTPUT_HANDLE), coord);
8      return 0;
9  }
10
11 int main(int argc, char *argv[]){
12     system("cls");
13     goto_xy(10,10);
14     printf("HOLA MUNDO");
15     return 0;
16 }
```


Como Programar en C/C++, Deitel (Prentice Hall), 2da Edición.


Programming Principles and Practice Using C++, Bjarne Stroustrup.


http://www.codeblocks.org


http://www.wxwidgets.org