Estructura de Datos y Algoritmos - Ejercicios de Punteros y Estructuras

Tecnólogo en Informática • Mariano Zunino

Lenguaje de Programación C • Estructura de Datos y Algoritmos

1. PUNTEROS

Referencia teórica

Un **puntero** es una **variable** cuyo **valor** es la **dirección de memoria** de un **dato**, y cuyo **tipo** indica el **tipo de dato** al que apunta.

Declaración de Punteros

```c
int *punt;     // puntero a entero llamado "punt"
char *punt1;   // puntero a char llamado "punt1"
float *punt2;  // puntero a float llamado "punt2"
```

Operadores Fundamentales

- **Operador de indirección (*):** Devuelve el valor almacenado en la dirección apuntada
- **Operador de dirección (&):** Devuelve la dirección de memoria de una variable

Ejemplo Básico

```c
int *punt;     // declaración del puntero
int x = 10;    // variable entera
int y = 20;    // otra variable entera

punt = &x;     // punt apunta a la dirección de x
*punt = 4;     // modifica el valor de x a través del puntero
punt = &y;     // ahora punt apunta a y
*punt = 8;     // modifica el valor de y
printf("%d, %d", x, y); // imprime: 4, 8
```

Relación entre Punteros y Arrays

El nombre de un array es equivalente a la dirección de su primer elemento:

```c
int miarray[7];
int *punt;
punt = miarray;        // equivale a punt = &miarray[0]
*punt = 5;            // equivale a miarray[0] = 5
*(punt + 2) = 10;     // equivale a miarray[2] = 10
```

Paso por Referencia en Funciones

Los punteros permiten modificar variables desde dentro de funciones:

```c
void intercambiar(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

int main() {
    int x = 5, y = 10;
    intercambiar(&x, &y);  // ahora x=10, y=5
    return 0;
}
```

Ejercicios Prácticos

Ejercicios Básicos de Punteros

Ejercicio 1

```c
int *punt;
int x = 7, y = 5;
punt = &x;
*punt = 4;
printf("%d, %d", x, y);
```

Ejercicio 2

```c
int *punt;
int x = 7, y = 5;
punt = &x;
x = 4;
printf("%d, %d", *punt, y);
```

Ejercicio 3

```c
int *punt;
int x = 7, y = 5;
punt = &x;
x = 4;
punt = &y;
printf("%d, %d", *punt, x);
```

Ejercicio 4

```c
int *punt;
int x = 7, y = 5;
punt = &x;
*punt = 3;
punt = &y;
*punt = x;
x = 9;
printf("%d, %d", *punt, x);
```

Ejercicio 5

```c
int *punta, *puntb;
int x = 7, y = 5;
punta = &x;
*punta = 3;
puntb = &y;
*puntb = x;
x = 9;
printf("%d, %d", *puntb, x);
```

Estructura de Datos y Algoritmos - Ejercicios de Punteros y Estructuras

Ejercicio 6

```c
int *punta, *puntb;
int x = 7, y = 5;
punta = &x;
*punta = 3;
puntb = &y;
*puntb = x;
x = 9;
printf("%d, %d", *puntb, *punta);
```

Ejercicio 7

```c
int *punta, *puntb;
int x = 7, y = 5;
punta = &x;
*punta = 3;
puntb = &y;
*puntb = x;
x = 9;
puntb = punta;
printf("%d, %d", *puntb, y);
```

Ejercicios de Punteros y Arrays

Ejercicio 8

```c
int *punt, i;
int x[5] = {1, 2, 3, 4, 5};
punt = x;
*punt = 9;
for(i = 0; i < 5; i++)
    printf("%d,", x[i]);
```

Ejercicio 9

```c
int *punt, i;
int x[5] = {1, 2, 3, 4, 5};
punt = &x[0];
*punt = 9;
punt[3] = 7;
for(i = 0; i < 5; i++)
    printf("%d,", x[i]);
```

Ejercicio 10

```c
int *punt, i;
int x[5] = {1, 2, 3, 4, 5};
punt = x;
*x = 11;
*(punt + 3) = 9;
for(i = 0; i < 5; i++)
    printf("%d,", x[i]);
```

Ejercicio 11

```c
int *punt, i;
int x[5] = {1, 2, 3, 4, 5};
punt = x;
*(punt + 2) = 9;
*(x + 3) = 7;
punt[1] = 11;
for(i = 0; i < 5; i++)
    printf("%d,", *(punt + i));
```

Ejercicio 12

```c
int *punt, i;
int x[5] = {1, 2, 3, 4, 5};
punt = x + 4;
*(punt - 2) = 9;
punt--;
*punt = 7;
punt[1] = 11;
for(i = 0; i < 5; i++)
    printf("%d,", *(x + i));
```

Ejercicio 13

```c
int *punt, i;
int x[5] = {1, 2, 3, 4, 5};
punt = &x[0] + 3;
*(punt - 2) = 9;
punt--;
*punt = 7;
punt[1] = 11;
punt = x;
for(i = 0; i < 5; i++)
    printf("%d,", punt[i]);
```

Ejercicios de Punteros y Funciones

Ejercicio 14

```c
void suma_dos(int *x, int *y, int *z) {
    *x = *x + 2;
    *y = *y + 2;
    *z = *z + 2;
}

int main() {
    int x = 3, y = 10, z = 15;
    suma_dos(&x, &y, &z);
    printf("%d %d %d", x, y, z);
    return 0;
}
```

Ejercicio 15

```c
void datos(int *x, float *y, char *c) {
    *x = 8;
    *y = 4.2;
    *c = 'g';
}

int main() {
    int x = 9;
    float y = 44.6;
    char c = 'a';
    datos(&x, &y, &c);
    printf("%d %.1f %c", x, y, c);
    return 0;
}
```

Ejercicio 16

```c
void datos(int *x, float *y, char *c) {
    printf("%p %.1f %c ", x, *y, *c);
    *x = 8;
    *y = 4.2;
    *c = 'g';
}

int main() {
    int x = 9;
    float y = 44.6;
    char c = 'a';
    datos(&x, &y, &c);
    printf("%d %.1f %c", x, y, c);
    return 0;
}
```

Ejercicio 17

```c
void datos(int x, float y, char c) {
    printf("%d %.1f %c ", x, y, c);
    x = 8;
    y = 4.2;
    c = 'g';
}

int main() {
    int x = 9;
    float y = 44.6;
    char c = 'a';
    datos(x, y, c);
    printf("%d %.1f %c", x, y, c);
    return 0;
}
```

Ejercicio 18

```c
int datos(int x, float y, char c) {
    printf("%d %.1f %c ", x, y, c);
    x = 8;
    y = 4.2;
    c = 'g';
    return x;
}

int main() {
    int x = 9;
    float y = 44.6;
    char c = 'a';
    x = datos(x, y, c);
    printf("%d %.1f %c", x, y, c);
    return 0;
}
```

Ejercicio 19

```c
char datos(int *x, float *y, char *c) {
    printf("%p %.1f %c ", x, *y, *c);
    *x = 8;
    *y = 4.2;
    *c = 'g';
    return 'h';
}

int main() {
    int x = 9;
    float y = 44.6;
    char c = 'a';
    c = datos(&x, &y, &c);
    printf("%d %.1f %c", x, y, c);
    return 0;
}
```

2. ESTRUCTURAS (STRUCTS)

Referencia teórica

Una estructura es un tipo de dato compuesto que permite agrupar variables de diferentes tipos bajo un mismo nombre. Es útil para representar entidades que tienen múltiples atributos relacionados.

Definición de Estructuras

```
struct estudiante {
    char nombre[50];     // campo nombre
    int edad;           // campo edad
    float promedio;     // campo promedio
};
```

Declaración de Variables

```
struct estudiante alumno1;              // variable individual
struct estudiante curso[30];            // array de estructuras
```

Acceso a Campos

Operador punto (.)

Para variables de estructura:

```
alumno1.edad = 20;
strcpy(alumno1.nombre, "Juan Pérez");
printf("Edad: %d", alumno1.edad);
```

Operador flecha (->)

Para punteros a estructuras:

```
struct estudiante *punt = &alumno1;
punt->edad = 21;
strcpy(punt->nombre, "Ana García");
printf("Nombre: %s", punt->nombre);
```

Arrays de Estructuras

```
struct estudiante curso[3];
for(int i = 0; i < 3; i++) {
    curso[i].edad = 18 + i;
    sprintf(curso[i].nombre, "Estudiante%d", i+1);
}
```

Ejercicios Prácticos

Ejercicio 1: Comprensión Básica

```
struct jugador {
    char nombre[50];
    int edad;
    float altura;
};

int main() {
    struct jugador mijugador;
    strcpy(mijugador.nombre, "MarcoPolo");
    mijugador.edad = 45;
    mijugador.altura = 1.8;
```

```
    printf("Nombre: %s\nAltura: %.1f\nEdad: %d\n",
           mijugador.nombre, mijugador.altura, mijugador.edad);
    return 0;
}
```

Ejercicio 2: Arrays de Estructuras

```
struct medidas {
    int alto, ancho, largo;
};

int main() {
    int i;
    struct medidas cubiletes[5];

    for(i = 0; i < 5; i++) {
        cubiletes[i].alto = 4;
        cubiletes[i].ancho = 2 * i;
        cubiletes[i].largo = i + 1;
    }

    for(i = 0; i < 5; i++) {
        printf("Cubilete %d: %d alto, %d ancho, %d largo\n",
               i, cubiletes[i].alto, cubiletes[i].ancho, cubiletes[i].largo);
    }
    return 0;
}
```

Ejercicio 3: Sistema de Gestión de Jugadores

Crear un programa que gestione información de 10 jugadores. El programa debe:

1. Solicitar al usuario el nombre, edad y altura de cada jugador
2. Presentar un menú con las siguientes opciones:
   - Listar todos los nombres
   - Listar todas las alturas
   - Listar todas las edades
   - Salir del programa

Requisitos técnicos:

- Usar una estructura `jugador` con campos apropiados
- Implementar un array de 10 jugadores
- Usar funciones para cada operación del menú

Ejercicio 4: Sistema Avanzado de Jugadores

Extender el ejercicio anterior agregando las siguientes funcionalidades:

4. Buscar un jugador por nombre y mostrar su información completa
5. Encontrar y mostrar el jugador más alto (nombre, edad y altura)
6. Calcular y mostrar la edad promedio de todos los jugadores

Requisitos adicionales:
- Implementar búsqueda de cadenas (usar `strcmp`)
- Manejar casos donde no se encuentra el jugador
- Validar entrada de datos

Ejercicio 5: Geometría con Estructuras

Crear un programa que trabaje con puntos en el plano cartesiano:

1. Definir una estructura `punto` con coordenadas x e y (números reales)
2. Solicitar las coordenadas de tres puntos que forman un triángulo
3. Calcular y mostrar el perímetro del triángulo

Fórmula de distancia:

```
distancia = sqrt((x2-x1)² + (y2-y1)²)
```

Notas:
- Incluir `#include <math.h>` para usar `sqrt()`
- Compilar con `-lm` para enlazar la librería matemática
- Validar que los tres puntos no sean colineales