Projeto e Análise de Algoritmos

Matheus Gabriel

Agosto de 2024

1 Ponteiros

1.1 Uso simples

Considere o seguinte código em C:

```c
int x = 1, y = 2, z[10];
int *ip;

ip = &x;   // ip aponta para a variável x
y = *ip;   // y recebe o valor de x (que é 1)
*ip = 0;   // o valor de x é alterado para 0
ip = &z[0]; // ip aponta para o primeiro elemento do array z
```

Na verdade, ao contrário do que muitos pensam, o ponteiro é armazenado na variável **ip** e não no asterisco `*`. O asterisco é utilizado para duas finalidades diferentes:

- **Declaração de ponteiro:** Para declarar um ponteiro, como em `int *ip;`.
- **Desreferenciação:** Para acessar o valor apontado pelo ponteiro, como em `*ip`.

1.1.1 Maneira errada

```c
void troca(int a, int b) {
    int aux;
    a = b;
    b = aux;
}

int main (void) {
    int x = 1, y = 2;
    printf("Antes: x = %d e y = %d", x, y);
    troca(x, y);
    printf("Depois: x = %d e y = %d", x, y);
    return 0;
}
```

Antes: x = 1 e y = 2
Depois: x = 1 e y = 2

1.1.2 Maneira certa

```c
void troca(int *a, int *b) {
    int aux;
    aux = *a;  // Armazena o valor de *a em aux
    *a = *b;   // Atribui o valor de *b a *a
    *b = aux;  // Atribui o valor de aux a *b
}

int main(void) {
    int x = 1, y = 2;
    printf("Antes: x = %d e y = %d\n", x, y);
    troca(&x, &y);  // Passa os endereços das variáveis x e y
    printf("Depois: x = %d e y = %d\n", x, y);
    return 0;
}
```

Antes: x = 1 e y = 2
Depois: x = 2 e y = 1

1.2 Alocação dinâmica de memória

O `malloc` é usado para fazer essa alocação dinâmica de memória:

```c
int main(void) {
    int i;
    int *arr;
    int n = 5; // Número de elementos no array

    // Aloca memória para um array de n inteiros
    arr = (int *)malloc(n * sizeof(int));

    // Inicializa o array com valores
    for (i = 0; i < n; i++) {
        arr[i] = i * 10;
    }

    // Imprime os valores do array
    printf("Valores no array:\n");
    for (i = 0; i < n; i++) {
        printf("arr[%d] = %d\n", i, arr[i]);
    }

    // Libera a memória alocada
    free(arr);

    return 0;
}
```

2 Estruturas `structs`

O código abaixo usa duas estruturas, `Ponto` e `Retângulo`, note o modo que elas são declaradas:

```c
#include <stdio.h>

typedef struct {
    int x;
    int y;
} Ponto;

typedef struct {
    Ponto cantoInferiorEsquerdo;
    Ponto cantoSuperiorDireito;
} Retangulo;
```

```c
int main(void) {
    Ponto p1 = {0, 0};
    Ponto p2 = {10, 5};

    Retangulo r;
    r.cantoInferiorEsquerdo = p1;
    r.cantoSuperiorDireito = p2;

    return 0;
}
```

No código `struct Ponto p1;`, a variável é do tipo `struct Ponto` e não apenas `Ponto`.

3 Estruturas auto-referenciais

Por exemplo, uma lista ligada é um conjunto de itens onde cada item é parte de um nó que também contém um link para um nó. Em C temos:

```c
typedef struct node *link;

struct node {
    int item;
    link next;
};
```