Ponteiros, Aritmética de Ponteiros, Passagem de Argumentos

1. Introdução aos Ponteiros

Um ponteiro é uma variável que armazena o endereço de memória de outra variável. Ele é uma referência (ele aponta) para um endereço de memória, possibilitando a passagem de argumentos para uma função por referência, o gerenciamento de memória dinâmico e a implementação de estruturas de dados complexas. Os ponteiros permitem operações de baixo nível e otimização de desempenho.

Importante: o ponteiro não armazena um valor, mas sim a localização na memória em que este valor está armazenado.

Para declarar um ponteiro, é necessário especificar o tipo de dado que ele irá referenciar. A sintaxe é:

```
tipo_dado *nome_ponteiro;
```

- tipo_dado especifica o tipo de dado que o ponteiro aponta (int, float, double, char, void, etc.).
- nome_ponteiro é a descrição (nome) do ponteiro. Observe que antes do nome, é incluído o operador "*", responsável por declarar nome_ponteiro como um ponteiro.

Exemplo de declaração:

```
int *p; // Declara um ponteiro "p" para um inteiro
```

A partir da declaração do ponteiro p, é possível que ele armazene endereços de memória de outras variáveis do tipo inteiro, permitindo o acesso, manipulação e referência de valores diretamente na memória.

É possível encontrar citações nas quais o operador * é chamado de operador de "desreferência". Isso se refere ao fato do operador * permitir o acesso ao valor armazenado no endereço de memória apontado pelo ponteiro. Sua utilização faz sentido quando o objetivo é recuperar e manipular o valor real ao invés do endereço de memória.

Exemplo de código 1:

```
#include <stdio.h>

int main () {
  int *p;  // Declara um ponteiro para inteiros
  int nro_1 = 350;
  int nro_2 = 1080;

  // o formato %p exibe o endereço
  printf("Endereço de p: %p \n", &p);  // Imprime o endereço de p
  printf("Endereço de nro_1: %p \n", &nro_1);  // Imprime o endereço de nro_1
  printf("Endereço de nro_2: %p \n", &nro_2);  // Imprime o endereço de nro_2

  printf("Valor de p: %p\n", p);  // Imprime o conteúdo do endereço de p (não acessível)
```

```
  p = &nro_1;  // p recebe o endereço de nro_1
  printf("Valor de p: %p \n", p);  // Imprime o conteúdo do endereço de p (endereço de nro_1)
  printf("Valor *p: %d \n", *p);  // Imprime o conteúdo do endereço apontado por p

  p = &nro_2;  // p recebe o endereço de nro_2
  printf("Valor de p: %p \n", p);  // Imprime o conteúdo do endereço de p (endereço de nro_2)
  printf("Valor de *p: %d \n", *p);  // Imprime o conteúdo do endereço apontado por p

  *p = 256;   // Atribiu o valor ao endereço apontado por p
  printf("Valor de nro_2: %d \n", nro_2);

  return 0;
}
```

Saída:

```
Endereço de p: 0x7ff7bae82630
Endereço de nro_1: 0x7ff7bae8262c
Endereço de nro_2: 0x7ff7bae82628
Valor de p: 0x22
Valor de p: 0x7ff7bae8262c
Valor *p: 350
Valor de p: 0x7ff7bae82628
Valor de *p: 1080
Valor de nro_2: 256
```

Veja uma simulação da execução do programa:

Execução	p (0x7ff7b1723630)	nro_1 (0x7ff7b172362c)	nro_2 (0x7ff7b1723628)
int *p;	0x22 (não acessível)	-	-
int nro_1 = 350;	0x22 (não acessível)	350	-
int nro_2 = 1080;	0x22 (não acessível)	350	1080
p = &nro_1;	0x7ff7b172362c	350	1080
p = &nro_2;	0x7ff7b1723628	350	1080
*p = 256;	0x7ff7b1723628	350	256

1.1. Ponteiro nulo

Um ponteiro nulo (NULL) aponta para nada, para um endereço de memória inválido, ou seja, ele é utilizado quando o ponteiro não tem um valor válido para referenciar. A macro "NULL" define o valor de ponteiros nulos, que equivalem a 0 (zero) no C ANSI.

Uma boa prática de programação é inicializar os ponteiros com NULL antes de realizar a atribuição de um endereço de memória válido a eles, indicando que o ponteiro aponta para nada e evitando comportamentos indefinidos. Ao liberar a memória alocada, também se deve atribuir NULL ao ponteiro.

Exemplo de código 2:

```
int *p = NULL; // Declara um ponteiro NULL
if (p == NULL) {
  printf("O ponteiro é NULL \n");
}
```

Atenção: a tentativa de desreferenciar um ponteiro nulo irá resultar em erro de acesso à memória, exibindo mensagem de erro similar a “Segmentation fault”.
Exemplo de código 3:

```
...
int *p = NULL;


printf ("Endereço de p: %p \n", &p);
printf ("Valor da variável apontada por p (*p): %d \n", *p);
...
```

1.2. Ponteiro void

Um ponteiro void pode apontar para qualquer tipo de dado. Ele é considerado um tipo de ponteiro genérico. Contudo, é preciso converter o ponteiro para o tipo de dado que ele está apontando antes de realizar a desreferência.

A vantagem de se utilizar um ponteiro void (genérico) está na possibilidade do ponteiro apontar para qualquer tipo de dado. Apesar disso, a conversão do ponteiro para o tipo apropriado antes do processo de desreferência é obrigatória, e caso seja realizada de forma incorreta, irá resultar em erros. Outro ponto para se considerar é a dificuldade de compreensão (manutenção) de um código com excesso de ponteiros genéricos.

Exemplo de código 4:

```
...
void *p;  // Declarando um ponteiro void
int nro = 2023;
p = &nro;  // Atribuindo o endereço de nro ao ponteiro void
int *pInt = (int *)p;  // Convertendo o ponteiro void para um ponteiro int
printf("Valor de x: %d \n", *pInt);  // Valor de nro: 2023
...
```

2. Aritmética de ponteiros

É possível realizar operações matemáticas diretamente nos endereços de memória por meio de ponteiros, possibilitando percorrer estruturas de dados como arrays ou buffers. Veja as operações:

- Incremento (p++): avança o ponteiro para o próximo elemento.
- Decremento (p--): retrocede o ponteiro para o elemento anterior.
- Adição (p + n): avança o ponteiro n elementos à frente.
- Subtração (p - n): retrocede o ponteiro n elementos para trás.

Importante: O deslocamento do ponteiro é realizado com base no tamanho do tipo do ponteiro. Portanto, a aritmética de ponteiros deve considerar o tipo de dado correto para garantir que os deslocamentos sejam realizados de forma adequada.

Exemplo de código 5:

```
#include <stdio.h>

int main() {
  int array[] = {7, 14, 21, 28, 35};
  int *p = array; // Ponteiro apontando para o primeiro elemento do array

  printf("%d \n", *p); // Saída: 7

  p++; // Avança para o próximo elemento
  printf("%d \n", *p); // Saída: 14

  p = p + 2; // Avança dois elementos
  printf("%d \n", *p); // Saída: 28

  p = array; // Reinicializa o ponteiro para o início do array
  printf("%d \n", *p); // Saída: 7

  return 0;
}
```

3. Ponteiro para arrays

Um ponteiro para array permite o acesso e a manipulação de uma estrutura de array diretamente na memória, possibilitando percorrer e manipular os elementos do array de forma mais eficiente (principalmente em arrays maiores). Além disso, ponteiros para arrays são utilizados em algoritmos de busca e ordenação com o intuito de torná-los mais otimizados e eficientes.

Exemplo de código 6:

```
#include <stdio.h>

int main() {
  int nros[5] = {9, 18, 27, 36, 45};
  int *p = nros; // O ponteiro p aponta para o primeiro elemento de nros

  for (int i = 0; i < 5; i++) {
    printf("Elemento %d: %d \n", i, *p);
    p++; // Avança o ponteiro para o próximo elemento
  }

  return 0;
```

```
}
```

Veja alguns trechos de código para manipulação de arrays que são equivalentes:

```
#include <stdio.h>

int main() {
  int array_1[5] = {9, 18, 27, 36, 45};
  int array_2[5] = {10, 20, 30, 40, 50};

  int *p1;
  int *p2;

  p1 = array_1; // Aponta para a primeira posição do array
  p2 = &array_2[0]; // Aponta para a primeira posição do array. Não se esqueça do &

  for (int i = 0; i < 5; i++) {
    printf("Array 1 - Elemento %d: %d \n", i, *(p1 + i));
  }

  for (int i = 0; i < 5; i++) {
    printf("Array 2 - Elemento %d: %d \n", i, *p2++);
  }

  return 0;
}
```

4. Passagem de argumentos por valor e referência

A forma de passagem dos argumentos para uma função determina como as modificações irão afetar as variáveis originais e como a memória do computador será utilizada.

Na passagem por valor, uma cópia do conteúdo da variável é passada para a função, ou seja, qualquer alteração realizada no parâmetro não irá afetar a variável original, fora do escopo da função. Isso evita alterações indesejadas nos valores originais, mas é ineficiente para grandes conjuntos de dados devido à cópia extra destes dados.

Exemplo de código 7:

```
#include <stdio.h>

void incrementar(int nro) {
  nro++;
  printf("Valor dentro da função: %d \n", nro);
}

int main() {
  int nro = 10;
```

```
  incrementar(nro);
  printf("Valor fora da função: %d \n", nro); // Saída: Valor fora da função: 10
  return 0;
}
```

Na passagem por referência, o endereço de memória da variável é passado para função, permitindo que qualquer modificação no conteúdo desta variável altere também o valor original da variável. Essa forma economiza memória (pois não cria uma cópia extra da variável), sendo mais eficiente para manipulação de grandes conjuntos de dados, e facilita a modificação dos valores originais. Contudo, pode tornar o processo de depuração e identificação de inconsistências no programa mais difícil.

Exemplo de código 8:

```
#include <stdio.h>

void incrementarPorReferencia(int *nro) {
  (*nro)++;
  printf("Valor dentro da função: %d \n", *nro);
}

int main() {
  int nro = 10;
  incrementarPorReferencia(&nro);
  printf("Valor fora da função: %d \n", nro); // Saída: Valor fora da função: 11
  return 0;
}
```

5. Exercícios

5.1. Declare um ponteiro para inteiros e associe ele a uma variável do tipo inteiro. Utilize o ponteiro para realizar a alteração do valor da variável.

5.2. Implemente uma função que receba um array de inteiros e seu tamanho. A função deve retornar o maior valor e sua posição no array usando ponteiros.

5.3. Escreva uma função que recebe um array de inteiros e retorna a soma dos elementos.

5.4. Crie uma função que recebe um array de caracteres (String) e retorna seu tamanho (comprimento).

5.5. É possível implementar um array de ponteiros? Justifique. Em caso afirmativo, elabore um exemplo.

5.6. Podemos criar um ponteiro de ponteiro? Crie um breve texto que discuta essa questão e identifique situações nas quais a abordagem possa ser utilizada.

5.7. Realize uma pesquisa sobre a relação entre ponteiros e as linguagens Java e JavaScript. Java utiliza ponteiros? E JavaScript? Elabore um texto simples (de até dois parágrafos) sobre o tema. Cite as referências pesquisadas.