

Algoritmos e Programação II

Ponteiros

Sumário

- Ponteiros
- Operadores & e *
- Tipo base
- Expressões
 - Atribuição
 - Comparação
 - Aritmética
- Ponteiros e vetores

Ponteiro

- *“Ponteiro é uma variável que contém um endereço da memória. Esse endereço é normalmente a posição de uma outra variável na memória.”*
- Fornece meios para
 - Funções modificarem seus argumentos (passagem por referência)
 - Implementar alocação dinâmica de memória
 - Aumentar a eficiência de certas rotinas

Ponteiros (2)

- Declaração

- **tipo *nome_da_variável;**

- **tipo** é qualquer tipo de dado em C e indica o tipo base do ponteiro, ou seja, para que tipo de dado o ponteiro deve apontar. O conteúdo armazenado na variável ponteiro é sempre um endereço, independente do tipo base.
- **nome_da_variável** é o nome da variável ponteiro para o tipo especificado.

Ponteiros

- `char *caractere;`
/* cria um ponteiro que armazenará um endereço de um char (caractere) */
- `int *p1, *p2;`
/* cria dois ponteiros que armazenam endereços de inteiros */

Operador &

- Operador &

- É um operador unário.
- Devolve o endereço da memória do operando.

```
int *p;
```

```
int a=5, q;
```

```
...
```

```
p = &a; //ponteiro recebe o endereço da variável a
```

- Neste exemplo, se a variável 'a' ocupasse a posição de memória 1250, o valor da variável 'p' seria 1250.

Operador *

- Operador *
- É um operador unário.
- Devolve o valor da variável localizada no endereço que o segue (devolve o conteúdo apontado).
- Continuando o exemplo anterior:

```
q = *p; // q=5
```

As variáveis ponteiro devem sempre apontar para o tipo de dados correspondente ao seu tipo base.

Exemplo 1

```
1  #include <stdio.h>
2  int main () {
3      int x, y, *P1, *P2;
4      x = 10;
5      P1 = &x;
6      y = *P1 * 2;
7      printf ("%d, %d, %d\n", x, *P1, y);
8      P2 = &x;
9      *P2 = 30;
10     printf ("%d, %d, %d\n", x, *P1, y);
11     return 0;
12 }
```

Tipo base

- Um ponteiro contém um endereço. Este endereço aponta para exatamente um byte.
- Um inteiro ocupa quatro bytes de memória. Para saber quantos bytes cada tipo de dado ocupa na memória, utilize a função `sizeof()`, por exemplo:

```
int t;  
t = sizeof(double);
```

- É o tipo base que indica o tamanho do conteúdo que o ponteiro aponta.

Expressões - Atribuição

- Atribuição de Ponteiros
 - Acontece como com qualquer variável

```
1  #include<stdio.h>
2
3  int main(){
4      int i=100;
5      int *p1, *p2;
6
7      p1 = &i;
8      p2 = p1;
9
10     printf("%p    %p", p2, p1);
11     return 0;
12
13 }
14
```

Expressões - Atribuição

- Atribuição de Ponteiros

```
1  #include<stdio.h>
2
3  int main()
4  {
5      int x=100;
6      int *p1, *p2;
7
8      p1 = &x;
9      p2 = p1;
10
11     printf("%p %p\n%d %d", p1, p2, *p1, *p2);
12
13     return 0;
14 }
```

Expressões - Atribuição

- Atribuição de conteúdos (usando ponteiros)

```
1  #include<stdio.h>
2
3  int main(){
4      int x=100, y=20;
5      int *p1, *p2;
6
7      p1 = &x;
8      p2 = &y;
9      *p2 = *p1;
10
11     printf("%p %p\n%d %d", p1, p2, *p1, *p2);
12     return 0;
13 }
```

Expressões - Comparação de Ponteiros

- É possível comparar dois ponteiros em uma expressão relacional

```
if (p1>p2) /*Se verdadeiro, significa que p2 está  
    em uma posição de memória mais baixa que p1 */  
    ...
```

Expressões – Aritmética

- Aritmética de Ponteiros

- Adição e Subtração

- Exemplo:

- `pl` é um ponteiro para inteiros e está apontando para o endereço 2000.
 - Considerando os inteiros com tamanho de 4 bytes:
 - `pl++`
 - `pl` apontaria para o endereço 2004
 - Cada vez que é incrementado, aponta para o próximo inteiro
 - `pl--`
 - `pl` apontaria para o endereço 1996

Expressões – Aritmética (2)

- Cada vez que um ponteiro é **incrementado**, ele aponta para a posição de memória do próximo elemento do seu tipo base
- Cada vez que um ponteiro é **decrementado**, ele aponta para a posição de memória do elemento anterior
- Toda aritmética de ponteiros é feita relativamente ao tipo base do ponteiro.
- Pode-se somar e subtrair **inteiros** de ponteiros, por exemplo:
 $p1 + 10$

Ponteiros e Matrizes/Vetores

- C oferece dois métodos para acessar elementos de matrizes:
 - Indexação de matrizes
 - Aritmética de ponteiros
- Em C, o nome de uma matriz (ou vetor) é um ponteiro para o primeiro elemento da matriz (ou vetor)
- Assim, dada a declaração:
 - `char str[20], *ponteiro;`
- as expressões abaixo são equivalentes:
 - `ponteiro = &str[0];`
 - `ponteiro = str;`

Ponteiros e Vetores

- Pode-se acessar os elementos de vetor por meio de:
 - Indexação: `str[2]='A' ;`
 - ou
 - Aritmética de ponteiros: `*(ponteiro+2)='A' ;`

- Exemplo:

```
char a[40], *pa;
```

```
pa = a;
```

```
a[9] = 'x' ;
```

```
*(pa+9) = 'x' ;
```

Essas instruções são equivalentes

Esta instrução é mais rápida!!

Ponteiros e Vetores (2)

- Exemplo

```
1  #include <stdio.h>
2  int main() {
3      int x[5] = {1, 2, 3, 4, 5};
4      int *p, i;
5      p = x; //ou p = &x[0];
6      for (i=0; i<5; i++){
7          printf("%d", x[i]);
8          //ou printf("%d", p[i])
9      }
10     printf("\n");
11     for (i=0; i<5; i++){
12         printf("%d", *(p+i));
13     }
14     return 0;
15 }
```

Exemplo:

Escrever string lida a partir do primeiro espaço (indexação)

```
1  #include <stdio.h>
2  int main() {
3      char str[20], *point;
4      int i;
5      printf("Digite uma string: ");
6      gets(str);
7      if (str[0] != '\0') {
8          for (i = 0; str[i] && str[i] != ' '; i++);
9          if (str[i]) point = &str[i] + 1; /* se nao chegou no fim, parou no espaco, pula espaco */
10         printf("%s", point);
11     }
12     else printf("Nao foi digitada nenhuma palavra!");
13     return 0;
14 }
```

Exemplo: Escrever string lida a partir do primeiro espaço (ponteiros)

```
1  #include <stdio.h>
2  int main() {
3      char str[20], *point;
4      printf("Digite uma string: ");
5      gets(str);
6      if (str[0] != '\0') {
7          for (point = str; *point && *point != ' '; point++);
8          if (*point) point++;
9          printf("%s", point);
10     }
11     else printf("Nao foi digitada nenhuma palavra!");
12     return 0;
13 }
```

Exemplo: Escrever string lida a partir do primeiro espaço

(aritmética de ponteiros)

```
1  #include <stdio.h>
2  int main() {
3      char str[20], *point;
4      int i;
5      printf("Digite uma string: ");
6      gets(str);
7      point = str;
8
9      if (*(point+0) != '\0') {
10         for (i = 0; (*(point+i)) && *(point+i) != ' '; i++);
11
12         /* se nao chegou no fim, parou no espaco, pula espaco */
13         if (*(point+i)) point = point + i + 1;
14         printf("%s", point);
15     }
16     else printf("Nao foi digitada nenhuma palavra!");
17     return 0;
18 }
```

Utilizando ponteiros, resolva os seguintes exercícios:

1. Calcular a média aritmética de 2 números.
2. Calcular a soma de 2 vetores de ordem 10.
3. Contar a quantidade de letras de uma palavra.
4. Calcular o produto escalar de 2 vetores.
5. Ler uma palavra e invertê-la, dentro da mesma string.