Exercícios sobre ponteiros

Gabriel Witor Rodrigues de Almeida

9 de Novembro de 2023

1. Suponha a declaração de um programa: $int\ vet[10]$, $*ptr$, $value$. Quais das expressões abaixo são válidas?

Considerando o código como:

```c
#include <stdio.h>

int main(){
    int vet[] = {1,2,3,4,5,6,7,8,9,10};
    int *ptr = vet, value;

    //expressoes entram aqui

    return 0;
}
```

a) $ptr = vet++$

inválida: Essa expressão não é válida. Embora pareça correta, o que está sendo somado nessa expressão não é o endereço do elemento de índice 0 do vetor, mas sim o próprio vetor. Não é possível somar um vetor, e por isso a expressão é inválida.

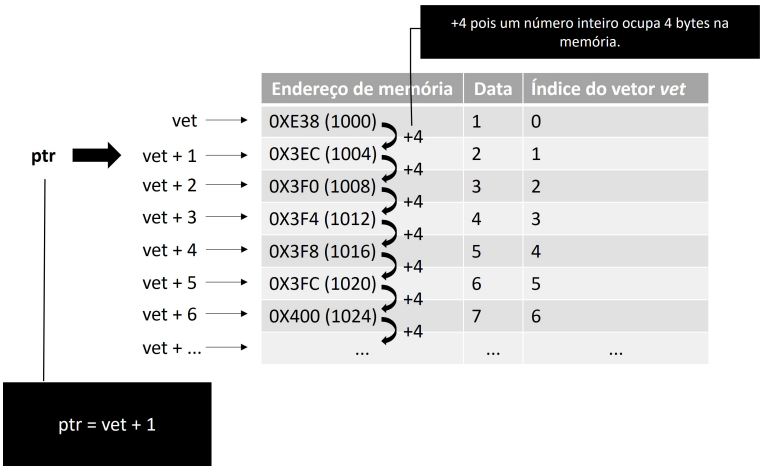
b) $vet[1] = ptr[3]$

válida: Essa expressão é válida. Como *ptr* aponta para *vet*, essa expressão faz com que a posição 1 do vetor receba o valor contido na posição 2 do vetor. Dessa forma, o vetor *vet* passa a ser $vet = \{1, 3, 3, 4, 5, 6, 7, 8, 9, 10\}$.

c) $ptr = vet + 1$

válida: Essa expressão é válida. Ao contrário da expressão da **alternativa a)**, quem está sendo somado nessa expressão é o *endereço do primeiro elemento de vet*. Dessa forma, caso o endereço do elemento de *índice* 0 do *vet* fosse $0X3E8$, isto é, 1000 em hexadecimal, *ptr* apontaria para $0X3EC$, isto é, 1004 em hexadecimal. A diferença de 4 *bytes* se deve a forma como números inteiros são alocados na memória: Cada número inteiro ocupa 4 *bytes* na memória. Ao somar 1 em um endereço de um ponteiro, a operação que está sendo realizada é a de deslocamento dentro da memória: O ponteiro passa a apontar para um endereço **x bytes** adiante do endereço anterior, dependendo do tipo do ponteiro (Para ponteiros de tipo inteiro, são 4 *bytes de deslocamento*, para ponteiros de tipo char é 1 *byte de deslocamento*, etc...).

Dessa forma, como um vetor é a mesma coisa que um ponteiro para o primeiro item desse determinado vetor, e como elementos de um vetor se apresentam um após o outro na memória, a expressão $ptr = vet + 1$ deve ser interpretada como: *ptr recebe o endereço 4 bytes adiante do endereço do primeiro elemento do vetor vet*. A imagem a seguir ilustra como se dá a operação de deslocamento:



d) $value = (*ptr)++$

válida: Essa expressão é válida, pois a operação ++ está sendo realizado no dado apontado por *ptr*. Caso *ptr* apontasse para o endereço de $vet[1]$ (Que vale 2), *value* receberia o valor 2 e o valor da posição $vet[1]$ seria incrementado em 1, resultando em $vet[1] = 3$. O vetor passaria a conter os seguintes elementos: $vet = \{1, 3, 3, 4, 5, 6, 7, 8, 9, 10\}$

2. Suponha a declaração de um vetor: *int* $vet[50]$. Qual expressão abaixo referência o quinto elemento do vetor?

a) $*(vet + 4)$

Incorreta. Essa expressão não é um endereço. Ela está passando o elemento contido no índice 4 no vetor.

b) $*(vet + 5)$

Incorreta. Essa expressão não é um endereço. Ela está passando o elemento contido no índice 5 no vetor.

c) $vet + 4$

Correta. Essa expressão é um endereço e está referenciando o elemento de índice 4 no vetor.

d) $vet + 5$

Incorreta. Essa expressão é um endereço, porém ela está referenciando o elemento de índice 5 no vetor.

e) nenhuma das alternativas

Incorreta.

3. Analise a sequência de instruções a seguir:

```
1 int x = 10, y = 5;
2 int *ptr1 = &x;
3 int *ptr2 = &y;
```

Quais expressões abaixo são válidas e quais não são válidas? Justifique sua resposta.

a) $y = ptr1 == ptr2;$

Expressão válida. Essa expressão compara dois endereços apontados por $ptr1$ e $ptr2$. y receberia o valor 0, pois a comparação $ptr1 == ptr2$ é falsa. $ptr1$ aponta para x enquanto $ptr2$ aponta para y.

b) $ptr1 \ += \ ptr2;$

Expressão inválida. Essa expressão é inválida pois não é possível somar dois ponteiros.

c) $x = (*ptr1) \ - \ (*ptr2);$

Expressão válida. Essa expressão é válida. x está recebendo o dado apontado por $ptr1$ menos o dado apontado por $ptr2$.

d) $x = ptr1 \ || \ ptr2;$

Expressão válida. Essa expressão é válida. Nesse caso, x está recebendo o valor booleano 1. Em operações *or* com endereços de memória, caso os dois ponteiros sejam nulos ou inválidos, é retornado 0. Caso ao menos um ponteiro seja válido, o valor booleano retornado é 1.

e) $y = (*ptr2) + +;$

Expressão válida. Essa expressão é válida. Nesse caso, y está recebendo o dado apontado por $ptr2$, que é ele mesmo. A operação ++ não muda nada nesse caso, pois o incremento foi realizado após o dado apontado por $ptr2$ ser atribuído à y. Assim sendo, y continua com o valor 5.

4. Reescreva o programa abaixo usando ponteiros:

```c
#include <stdio.h>

#define MAX 255

int main (){

    char str [MAX], caractere;
    int count = 0;

    printf("Entre com a string: ");
    fgets (str, MAX, stdin);
    printf("Entre com o caractere: ");
    scanf(" %c", &caractere);

    for (int i = 0; str[i] != '\0'; i++) {
        if (str[i] == caractere) {
            printf("%d\n", i);
            count++;
        }
    }
        if (count == 0)
            printf("-1\n");

    return 0;
}
```

Uma implementação utilizando ponteiros:

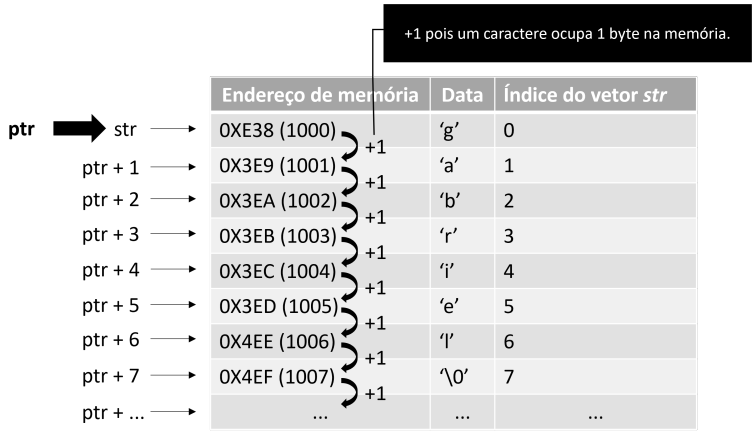
```c
#include <stdio.h>

#define MAX 255

int main (){

    char str [MAX], caractere, *ptr = str;
    int count = 0;

    printf("Entre com a string: ");
    fgets (str, MAX, stdin);

    printf("Entre com o caractere: ");
    scanf(" %c", &caractere);

    for (int i = 0; *(ptr+i) != '\0'; i++) {
        if (*(ptr+i) == caractere) {
            printf("Encontrei o caractere [%c] no indice: [%d]\n",
caractere,i);
            count++;
        }
    }

        if (count == 0)
            printf("-1\n");
```

```
25
26      return 0;
27  }
```

A implementação utiliza um ponteiro para percorrer o vetor *str*. A expressão $*ptr = str$; faz com que o ponteiro aponte para o endereço do elemento de índice 0 no vetor *str*. O loop *for* utiliza o ponteiro *ptr* para percorrer a string *str* através da memória. O loop *for* é iniciado com $i = 0$, ou seja, a verificação **(ptr+i) != '\0'* está analisando o caractere contido na posição da memória $ptr + 0$. O endereço apontado por *ptr* é o endereço do elemento de índice 0 da string *str*, portanto o que está sendo verificado é se o caractere de índice 0 de *str* é igual a '\0'.

A verificação é realizada suscetivamente, até que a comparação seja verdadeira e o laço *for* seja encerrado. Vale salientar que quando *i* é incrementado em 1, há um deslocamento na memória de *1 byte*. Isso se dá ao fato de que cada variável tipo *char* é alocada em *1 byte*.

Dentro do laço *for*, a lógica por trás de contabilizar quantas vezes determinado caractere se encontra na string é igual à lógica por trás de encontrar o caractere '\0' e encerrar o laço. A única diferença é que a verificação se dá com o caractere digitado pelo usuário ao invés do caractere '\0'. Quando a comparação é verdadeira, a posição que o caractere foi encontrado é impressa e o contador *count* é incrementado em 1. Há a possibilidade do caractere digitado pelo usuário não ser encontrado nenhuma vez, e quando isso ocorre o número -1 é impresso. A imagem a seguir ilustra como é realizado o deslocamento na memória



5. Indique quais as saídas produzidas pelo programa a seguir. Faça o teste de mesa de cada instrução e verifique os resultados. Depois, você pode executar o código comparando os resultados.

```c
#include <stdio.h>

int main () {
    int x, y = 27;

    int *pt1 = &x;

    int *pt2 = &y;

    int **ppt = &pt1;

    **ppt = *pt2;

    (*pt2)++;

    x--;
    printf("%d %d\n", *pt1, *pt2);

    (**ppt) += --(*pt2);

    printf("%d\n", **ppt);
    printf("%d %d\n", x, y)
    printf("%d\n", pt1 == &y);
    printf("%d\n", &x != pt2);

    return 0;
}
```

Resultados obtidos:

- **Primeiro printf:** 26 28
- **Segundo printf:** 53
- **Terceiro printf:** 53 27
- **Quarto printf:** 0
- **Quinto printf:** 1

Uma breve explicação dos resultados obtidos:

Primeiramente, as variáveis x e y são iniciadas. Depois, os ponteiros $pt1$ e $pt2$ são criados, $pt1$ apontando para x e $pt2$ apontando para y. Depois, o ponteiro ppt é criado, apontando para o endereço de $pt1$.

Já na linha 12, as operações com os ponteiros são iniciadas. A primeira expressão $**ppt = *pt2$ faz com que o valor do dado apontado por ppt, que aponta pro $pt1$, que aponta pra variável x passe a valer o dado apontado por $pt2$ ($y = 27$). Dessa maneira, x e y passam a valer ambos 27.

A expressão que segue na linha 14, isto é, a expressão $(*pt2)++$ incrementa o valor do dado apontado por $pt2$. A variável y agora passa a valer 28.

A expressão da linha 16 decrementa o valor de x, que agora passa a valer 26. São impressos então os valores de x e y, que valem 26 e 28, respectivamente.

A última expressão que realiza alguma operação com os vetores $((**ppt) += --(*pt2))$, presente na linha 17, modifica o valor do dado apontado por ppt, que aponta pra $pt1$, que aponta para x, fazendo com que x receba o valor dele próprio mais o valor do dado apontado por $pt2$ (y) decrementado uma vez. Portanto, x passa a valer 53, e como o valor do dado apontado por $pt2$ sofreu decremento, y passa a valer 27.

O segundo $printf$ imprime o valor do dado apontado por ppt, que por sua vez aponta para $pt1$, que aponta para x. Dessa forma, o valor de x (53) é impresso. Quanto ao terceiro $printf$, os valores das variáveis x e y são impressos. Referente ao quarto $printf$, a expressão booleana impressa verifica se o endereço apontado por $pt1$ é o endereço de y. $pt1$ aponta para o endereço de x, portanto um 0, que representa $false$, é impresso. No que tange o quinto $printf$, é impressa a expressão booleana que verifica se o endereço de x é o mesmo endereço apontado por $pt2$. Nessa caso, o valor impresso é 1, pois $pt2$ aponta para y.

6. Indique quais as saídas produzidas pelo programa a seguir. Faça o teste de mesa de cada instrução e verifique os resultados. Depois, você pode executar o código comparando os resultados.

```
1 #include <stdio.h>
2     int main () {
3
4     int vet1 [] = {1, 2, 3, 4, 5, 6, 7};
5     int vet2 [] = {7, 6, 5, 4, 3, 2, 1};
6
7     int *ptr1 = vet1;
8     int *ptr2 = vet1 + 3;
9     int *ptr3 = vet2 + 5;
10
11    (*ptr1)++;
12    (*ptr2)++;
13    (*ptr3)--;
14
15    printf("vet1[0]: %d, vet1[3]: %d\n", vet1 [0], vet1 [3]);
16    printf("vet2 [0]: %d, vet2[5]: %d\n", vet2 [0], vet2 [5]);
17
18    return 0;
19 }
```

Resultados obtidos:

- **Primeiro printf:** $vet1[0]: 2,\ vet1[3]: 5$
- **Segundo printf:** $vet2[0]: 7,\ vet2[5]: 1$

Uma breve explicação dos resultados obtidos:
Primeiramente, os vetores $vet1$ e $vet2$ são inicializados. Depois, três ponteiros são inicializados:

- $*ptr1 = vet1;$

  $ptr1$ aponta para o endereço do elemento de índice 0 do $vet1$.

- $*ptr2 = vet1 + 3;$

  $ptr1$ aponta para o endereço do elemento de índice 3 do $vet1$. A operação de soma faz com que o ponteiro desloque 3 vezes na memória, ou seja, desloque 12 *bytes* (4 *bytes* por deslocamento, pois o vetor é formado por números inteiros, e cada número inteiro é alocado na memória utilizando-se 4 *bytes*.)

- $*ptr3 = vet2 + 5;$

  $ptr3$ aponta para o endereço do elemento de índice 5 do $vet2$.

Após a criação dos vetores, algumas operações com os valores dos dados apontados pelos ponteiros são realizadas:

- $(*ptr1)++$;

  Essa operação incrementa o valor do dado apontado por $ptr1$ em uma unidade. Dessa maneiram, como $ptr1$ aponta para $vet1[0] = 1$, $vet1[0]$ passa a valer 2.

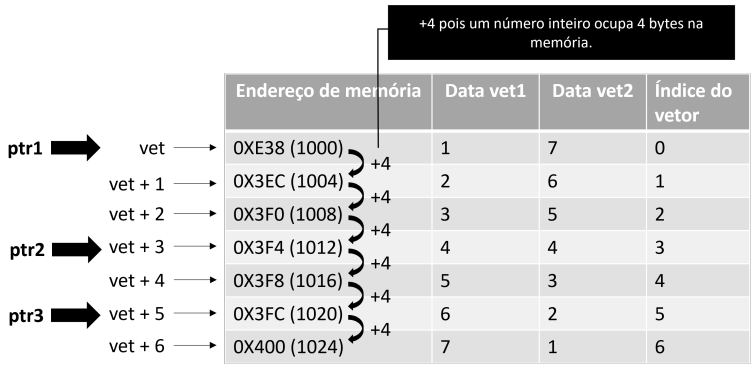
- $(*ptr2)++$;

  Essa operação incrementa o valor do dado apontado por $ptr2$ em uma unidade. Assim sendo, o valor do dado apontado por $ptr2$, que é $vet1[3] = 4$ passa a valer 5.

- $(*ptr3)--$;

  Essa operação decrementa o valor do dado apontado por $ptr3$ em uma unidade. Portanto, o valor do dado apontado por $ptr3$ ($vet[5] = 2$) passa a valer $vet[5] = 1$

O esquema ilustra o processo de deslocamento na memória:



7. Faça um programa que receba um vetor de 20 elementos inteiros, em seguida, percorra o vetor através do ponteiro ptr-inicio - a partir do início do vetor e outro ponteiro ptr-fim a partir do final do vetor, até os dois ponteiros se encontrarem no meio do vetor.

```c
#include <stdio.h>

#define MAXSIZE 20

int main(){
    int vetor[] =
    {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19};
    int *ptr_inicio = vetor, *ptr_fim = &vetor[MAXSIZE-1];

    do{
        printf("%d %d\n",*ptr_inicio,*ptr_fim);
        ptr_inicio++, ptr_fim--;

    } while (ptr_inicio<ptr_fim);

    return 0;
}
```

8. Utilizando aritmética de ponteiros, mostre como exibir a frase "não gosto de programar em C" como "gosto de programar em C".

```c
#include <stdio.h>

#define FRASE "n o gosto de programar em C"

int main(){

    char * ptr = FRASE;

    for(int i = 0; *(ptr) != 'g'; i++)
        ptr++;

    while(*ptr != '\0'){
        printf("%c",*ptr);
        ptr++;
    }

    return 0;
}
```

9. Implemente uma funçao que receba um vetor de inteiros, o tamanho do vetor e um inteiro pos passado por referência. A função retorna o maior elemento do vetor e, na variável pos, a posição do maior elemento do vetor.

```c
#include <stdio.h>

#define MAXSIZE 10

void IndiceMaiorElemento(int *vet,int *pos);

int main(){
    int vet[10] = {0,1,2,3,4,5,6,7,8,9}, pos;

    IndiceMaiorElemento(vet, &pos);

    printf("Indice do elemento de maior valor: %d",pos);


    return 0;
}

void IndiceMaiorElemento(int *vet,int *pos){
    int posMaiorElemento = 0;

    for(int i = 0; i < MAXSIZE; i++)
        posMaiorElemento = (vet[posMaiorElemento] < vet[i]) ? i :
    posMaiorElemento;

    *pos = posMaiorElemento;
}
```

10. Faça um programa que leia uma matriz quadrada de ordem 4 X 4 de números inteiros. Depois, leia um número x e verifique quantas vezes x aparece na matriz.

```c
#include <stdio.h>

#define MAXSIZE 4

int main(){
    int matrix[MAXSIZE][MAXSIZE], * ptr = &matrix[0][0];

    for(int i = 0; i < MAXSIZE*MAXSIZE; i++){
        printf("Digite o elemento numero [%d]: ",i+1);
```

```
        scanf("%d",(ptr+i));
    }

    int x, count = 0;
    printf("Digite um valor inteiro: ");
    scanf("%d",&x);

    ptr = &matrix[0][0];

    for(int i = 0; i < MAXSIZE*MAXSIZE; i++)
        if(*(ptr+i) == x)
            count++;

    printf("O numero [%d] aparece [%d] vezes na matriz.",x,count);

    return 0;
}
```

11. Faca um programa que leia uma string de no maximo 100 caracteres. Em seguida, implemente uma função para calcular e mostrar o total de palavras da string lida. Para isso, utilize o protótipo de função a seguir.

```
int totalPalavras (char *str) {

}
```

```
#include <stdio.h>

#define MAXSIZE 100

int totalPalavras (char *str) {
    int count = 0;
    for(int i = 0; str[i] != '\0'; i++)
        if(str[i] == ' ')
            count++;

    return count+1;
}

int main(){
    char frase[MAXSIZE];

    printf("Digite uma frase: ");
    fgets(frase,MAXSIZE,stdin);

    printf("Total de palavras: %d\n",totalPalavras(frase));

```

```
22      return 0;
23  }
```