

1º Lista de Exercícios

Isaac Gomes Medeiros - 20170035516
Programação Avançada - 246M12

(Questão 1)

Ponteiros tem como principal finalidade viabilizar o uso de de uma variável diversas vezes, sem que a mesma tenha que ser necessariamente modificada. Sua aplicação na engenharia é fundamental em quesitos como otimização e organização. Entre vários aspectos que o uso do ponteiro é uma vantagem, podemos citar sua viabilidade para o gerenciamento de estruturas que são alocadas na memória dinamicamente, a passagem de vetores e matrizes para funções de forma mais eficiente etc.

(Questão 2)

p == &i; Resultado = 1

*p - *q; Resultado = -2

**&p; Resultado = 3

3 - *p/(*q) + 7; Resultado = 10

(Questão 3)

P; Resultado = 4094

*p+2; Resultado = 7

**&p; Resultado = 5

3**p; Resultado = 15

**&p+4; Resultado = 9

(Questão 4)

p = i; Resultado = Illegal

q = &j; Resultado = Legal

p = &*&i; Resultado = Legal

i = (*&j); Resultado = Illegal

i = *&j; Resultado = Legal

i = *&*&j; Resultado = Legal

q = *p; Resultado = Illegal

i = (*p)++ + *q; Resultado = Legal

(Questão 5)

- A- 20
- B- 29.0
- C- P
- D- e
- E- P
- F- e
- G- t
- H- 31
- I- 45
- J- 27
- L- 31
- M- 45
- N- 27

(Questão 6)

contador/valor/valor/endereco/endereco

$i = 0$ vet[0] = 1.1*(f + 0) = 1.1&vet[0] = <Local de memoria de vet[0]>(f + 0) = <Local de memoria de vet[0]>

$i = 1$ vet[1] = 2.2*(f + 1) = 2.2&vet[1] = <Local de memoria de vet[1]>(f + 1) = <Local de memoria de vet[1]>

$i = 2$ vet[2] = 3.3*(f + 2) = 3.3&vet[2] = <Local de memoria de vet[2]>(f + 2) = <Local de memoria de vet[2]>

$i = 3$ vet[3] = 4.4*(f + 3) = 4.4&vet[3] = <Local de memoria de vet[3]>(f + 3) = <Local de memoria de vet[3]>

$i = 4$ vet[4] = 5.5*(f + 4) = 5.5&vet[4] = <Local de memoria de vet[4]>(f + 4) = <Local de memoria de vet[4]>

(Questão 7)

*(pulo+2)

(Questão 8)

p = mat + 1; -> Valida, pois o p está recebendo o local de memoria de mat[1], o ponteiro em si foi declarado como uma variável.

p = mat++; -> Invalido. Mat não pode ser incrementado, é um vetor.

`p = ++mat;` Invalido. Mat não pode ser incrementado, é um vetor.

`x = (*mat)++;` Válido, pois o x receberá o primeiro índice de mat.

(Questão 9)

No primeiro, mostra as posições de `vet[0]`, `vet[1]` e `vet[2]` que é 4, 9 e 13.

No segundo, vai mostrar o local na memória de `vet[0]`, `vet[1]` e `vet[2]`.

(Questão 10)

x for declarado como char:

`x+1=4093`

`x+2=4094`

`x+3=4095`

x for declarado como int:

`x+1= 4094`

`x+2= 4096`

`x+3= 4098`

x for declarado como float:

`x+1=4096`

`x+2=5000`

`x+3=5004`

x for declarado como double:

`x+1=5000`

`x+2=5008`

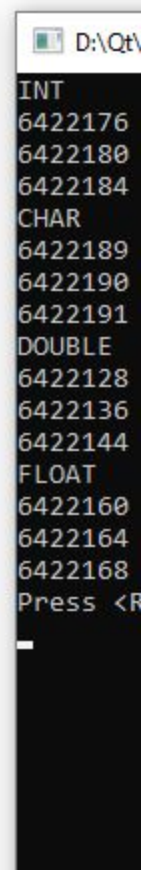
`x+3=5016`

(Questão 11)

```
#include <iostream>

using namespace std;

int main()
{
    char z[4];
    int x[4];
    float y[4];
    double w[4];
    printf("INT\n%d\n", (x+1));
    printf("%d\n", (x+2));
    printf("%d\n", (x+3));
    printf("CHAR\n%d\n", (z+1));
    printf("%d\n", (z+2));
    printf("%d\n", (z+3));
    printf("DOUBLE\n%d\n", (w+1));
    printf("%d\n", (w+2));
    printf("%d\n", (w+3));
    printf("FLOAT\n%d\n", (y+1));
    printf("%d\n", (y+2));
    printf("%d\n", (y+3));
}
```



```
D:\Qt\
INT
6422176
6422180
6422184
CHAR
6422189
6422190
6422191
DOUBLE
6422128
6422136
6422144
FLOAT
6422160
6422164
6422168
Press <R>
```

Como pode perceber, o int do pc vale 4 bytes, e não 2.

(Questão 12)

aloha[2] = value; Resultado= Válido
scanf("%f", &aloha); Resultado= Válido
aloha = value; Resultado= Inválido
printf("%f", aloha); Resultado= Inválido
coisas[4][4] = aloha[3]; Resultado= Válido
coisas[5] = aloha; Resultado= Inválido
pf = value; Resultado= Inválido
pf = aloha; Resultado= Válido

(Questão 13)

Um ponteiro para uma função possibilita o acesso ao local de memória da função. O chamamento da função pode ser feita através desse ponteiro.

EXEMPLO:

```
#include <stdio.h>
```

```
int subtracao(int x, int y){  
    int k = x-y;  
    return k;  
}
```

```
int main(){  
  
    int a = 5, b=3, c;  
  
    int (*p)(int*, int*);  
    p = subtracao;  
    c = p(a,b);  
  
    printf("Subtracao entre %d e %o eh %i", a, b, c);  
  
    return 0;  
}
```

(Questão 14)

```
#include <stdio.h>
```

```
void ordenar(int n ,float *v[]){
```

```
    for(int i = 0; i<n-1; i++){
        for(int l = i+1; l<n; l++){
            if(v[i]>v[l]){
                float *j = v[i];
                v[i] = v[l];
                v[l] = j;
            }
        }
    }
}
```

```
}
```

```
int main(){
```

```
    int n;
```

```
    printf("Numero de elementos: ");
    scanf("%i", &n);
```

```
    float *v = malloc(n*sizeof(float));
```

```
    printf("Digite os elementos: ");
    for(int i = 0; i<n; i++){

        scanf("%f", &v[i]);
    }
```

```
    ordenar(n, v);
```

```
    for(int i = 0; i<n; i++){
```

```
    printf("%f ", v[i]);  
}  
  
free(v);  
  
return 0;  
}
```

(Questão 15)

```
#include <stdio.h>  
#include <stdlib.h>  
  
int teste(const void *a, const void *b){//função que vai entrar como parametro de  
comparação na função qsort  
    return (*(int*)a-*(int*)b); //se a for maior, retorna inteiro positivo, igual retorna 0, menor  
    retorna negativo  
}  
  
int main(){  
  
    int n;//quantidade de termos  
  
    printf("Numero de elementos: ");  
    scanf("%i", &n);//atribui à "n" o dado fornecido pelo usuario  
  
    float *v = malloc(n*sizeof(float));//alocacao do vetor  
  
    printf("Digite os elementos: ");  
    for(int i = 0; i<n; i++){//for responsavel em receber os elementos do vetor  
  
        scanf("%f", &v[i]);  
    }
```


qsort(v, n, sizeof(float), teste); //funcao QSORT(). Recebe o ponteiro, n° de elementos, tamanho dos elementos e a função que realizará a comparação

```
for(int i = 0; i<n; i++){  
  
    printf("%f ", v[i]);  
}  
  
free(v); //libera o espaço criado na memoria para o vetor  
  
return 0;  
}
```

(Questão 16)

```
#include <stdio.h>
```

```
int comparar (float x, float y){  
    if (x < y){  
        return 1;  
    }  
    else{  
        return 0;  
    }  
}
```

```
void ordenar(float *v[], int n, int (*comparar)(float *, float *)){
```

```
    for(int i=0; i<n; i++){  
  
        for(int j=0; j<i; j++){  
  
            if(comparar(v[i], v[j])){  
  
                float *k=v[i];  
                v[i]=v[j];  
                v[j]=k;  
            }  
        }  
    }  
}
```

```
    }  
  }  
}
```

```
int main(){  
  
    int n;  
  
    printf("Numero de elementos: ");  
    scanf("%i", &n);  
  
    float *v = malloc(n*sizeof(float));  
  
    printf("Digite os elementos: ");  
    for(int i = 0; i<n; i++){  
  
        scanf("%f", &v[i]);  
    }  
  
    float (*p) = comparar;  
  
    ordenar(v, n, p);  
  
    for(int i = 0; i<n; i++){  
  
        printf("%f ", v[i]);  
    }  
  
    free(v);  
  
    return 0;  
}
```

(Questão 17)

A biblioteca utilizada para tal foi a <time.h>

Para conseguir o tempo de cada função, foi feito para a questão 16:

```
float a = clock();
ordenar(v, n, p);
float b = clock();
float tempo = b-a;
printf("TEMPO = %f", tempo/CLOCKS_PER_SEC);
```

```
Numero de elementos: 15000
TEMPO = 0.556000
```

0.556 segundos para 15000 elementos

E usando qsort:

```
float a = clock();
qsort(v, n, sizeof(float), teste);
float b = clock();
float tempo = b-a;
printf("TEMPO = %f", tempo/CLOCKS_PER_SEC);
```

```
Numero de elementos: 15000
TEMPO = 0.001000
```

0.001 segundos para 15000 elementos. Ou seja, a utilização da função QSORT() foi **extremamente** mais eficiente.

(Questão 18)

```
#include <stdio.h>
#include <stdlib.h>
```

```
void soma(int *a,int *b,int *c, int n){

    for(int i = 0; i<n; i++){
        c[i] = a[i] + b[i];
    }
}
```

```
}
```

```
int main()
{
    srand(time(NULL));
    int *a, *b, *c, n;

    printf("Numero de elementos: ");
    scanf("%d",&n);

    a=(int*)malloc(n*sizeof(int));
    b=(int*)malloc(n*sizeof(int));
    c=(int*)malloc(n*sizeof(int));
    for(int i=0;i<n;i++){
        b[i]=rand()%20;
        a[i]=rand()%10;
    }
    printf("B= ");
    for(int i=0; i<n; i++){
        printf("%d ", b[i]);
    }
    printf("\nA= ");
    for(int i=0; i<n; i++){
        printf("%d ", a[i]);
    }

    soma(a, b, c, n);

    printf("\nC= ");
    for(int i=0; i<n; i++){
        printf("%d ", c[i]);
    }
    printf("\n");
    free(a);
    free(b);
    free(c);
    return 0;
}
```

(Questão 19)

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
void multiplica(int **A,int **B,int **C,int la,int cb,int ca)
```

```
{
    for (int i = 0; i < la; ++i) {
        for (int j= 0; j < cb; ++j) {
            for (int k = 0; k < ca; ++k) {
                C[i][j] = C[i][j] + A[i][k]*(B[k][j]);
            }
        }
    }
}
```

```
int main(void){
```

```
    int la, ca, lb=ca, cb;
```

```
    printf("\nLinhas de A: ");
```

```
    scanf("%d", &la);
```

```
    printf("\nColunas de B: ");
```

```
    scanf("%d", &cb);
```

```
    printf("\nColunas de A: ");
```

```
    scanf("%d", &ca);
```

```
    int **A = (int**) malloc(la*sizeof(int*));
```

```
    A[0] = malloc(la*ca*sizeof(int));
```

```
    int **B = (int**) malloc(lb*sizeof(int*));
```

```
    B[0] = malloc(lb*cb*sizeof(int));
```

```
int **C = (int**) malloc(la*sizeof(int*));  
C[0] = malloc(la*cb*sizeof(int));
```

```
for (int i = 0; i < la; ++i) {  
    for (int j= 0; j < ca; ++j) {  
        A[i][j] = rand()%10;
```

```
    }  
}
```

```
for (int i = 0; i < lb; ++i) {  
    for (int j= 0; j < cb; ++j) {  
        B[i][j] = rand()%20;
```

```
    }  
}
```

```
for (int i = 0; i < la; ++i) {  
    for (int j= 0; j < cb; ++j) {  
        C[i][j] = 0;
```

```
    }  
}
```

```
printf("\nA: \n");
```

```
for (int i = 0; i < la; ++i) {  
    for (int j= 0; j < ca; ++j) {  
        printf(" %d ", A[i][j]);  
    }  
    printf("\n");
```

```
}
```

```
printf("\n B: \n");
```

```
for (int i = 0; i < lb; ++i) {  
    for (int j= 0; j < cb; ++j) {  
        printf(" %d ", B[i][j]);
```

```
    }  
    printf("\n");  
}
```

```
multiplica(A,B,C, la, cb,ca);
```

```
printf("\nC: \n");  
for (int i = 0; i < la; ++i) {  
    for (int j= 0; j < cb; ++j) {  
        printf(" %d ", C[i][j]);  
    }  
    printf("\n");  
}
```

```
free(A[0]);  
free(A);  
free(B[0]);  
free(B);  
free(C[0]);  
free(C);
```

```
return 0;  
}
```

(Questão 20)

```
#include <stdio.h>
#include <stdlib.h>
#include "gc.h"

int main(void)
{
    int n=1000;
    GC_INIT();
    int *x= GC_MALLOC(n*sizeof(int*));
    return 0;
}

return 0;
}
```

(Questão 21)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(void)
{
    clock_t tempo;
    int n=10000, **x;

    GC_INIT();

    tempo=clock();

    x=(int**)malloc(n*sizeof(int*));
    for(int i=0;i<n;i++){
        x[i]=(int*)malloc(n*sizeof(int));
        for(int j=0;j<n;j++){
            x[i][j]=222;
        }
    }
}
```



```
}  
}
```

```
for(int i=0;i<n;i++){  
    free(x[i]);  
}  
free(x);
```

```
tempo=clock()-tempo;  
printf("\n %f segundos usando malloc\n", ((float)tempo)/CLOCKS_PER_SEC);
```

```
tempo=clock();  
x=(int**) GC_MALLOC(n*sizeof(int*));  
for(int i=0;i<n;i++){  
    x[i]=(int*) GC_MALLOC(n*sizeof(int));  
    for(int j=0;j<n;j++){  
        x[i][j]=222;  
    }  
}  
tempo=clock()-tempo;  
printf("%f segundos usando GC_MALLOC\n",((float)tempo)/CLOCKS_PER_SEC);
```

```
return 0;  
}
```

```
0.466000 segundos usando malloc  
0.0772000 segundos usando GC_MALLOC
```

0.0466 segundos usando malloc

0.0772000 segundos usando GC_MALLOC, bem mais eficiente