GABARITO
PONTEIROS

1.

```
int x, y, *p;y = 0;
p = &y;//*p = 0
x = *p;//x = 0
x = 4;//x = 4
(*p)++;//*p = 1, y = 1
--x;//x = 3
(*p) += x;
//*p = 4, y = 4
```

Ao final, temos:x = 3, y = 4, p apontando para y (*p = 4).

2. a)

```
void main() {
int x, *p;
x = 100;
p = x;//p deveria receber o endereço de x, já que p é um ponteiro (e x não).Ponteiros “armazenam” o endereço para o qual eles apontam! O código correto seria: p = &x;
printf(“Valor de p: %d.\n”, *p);}
```

b)

```
void troca (int *i, int *j) {
int *temp;
*temp = *i;
*i = *j;
*j = *temp;}
```

A variável “temp” não precisava ser um ponteiro, já que apenas precisaarmazenar um valor inteiro, sem precisar apontar para algum lugar. O código correto seria:void troca (int *i, int *j) {int temp;temp = *i;*i = *j;*j = temp;}

c)

```
char *a, *b;

a = "abacate";//o ponteiro “a” ainda não aponta para algum lugar nem possuimemória alocada!

b = "uva";//o ponteiro “b” ainda não aponta para algum lugar nem possuimemória alocada!
```

```
if (a < b)

printf ("%s vem antes de %s no dicionário", a, b);

else

printf ("%s vem depois de %s no dicionário", a, b);
```

O correto seria:char a[] = "abacate", b[] = "uva";if (a[0] < b[0]) {printf ("%s vem antes de %s no dicionário", a, b);}else {printf ("%s vem depois de %s no dicionário", a, b);}Nesse caso, verificar apenas a primeira letra das cadeias de caracteres funciona,pois temos “a” e “u”. Porém, o código acima não funcionaria para os casos“manga” e “mamão”, por exemplo. Fica como exercício para o aluno criar umafunção que faz a verificação de toda a cadeira de caracteres.

3.

Se v (ou o endereço de v[0]), que representa o primeiro item do vetor está nobyte de endereço 55000, logo o índice v[3] (ou v + 3) estará no byte 55000 +8*3 = 55024. Nota: em máquinas de 32 bits, inteiros ocupam 32 bits (4bytes).

4.

```
#include <stdio.h>

#include <stdlib.h>

void mm(int *v, int n, int *min, int *max) {

int i;*min = v[0];

*max = v[0];

for (i = 1; i < n; i++) {

if (v[i] > *max) {

*max = v[i];

} else if (v[i] < *min) {

*min = v[i];

} }

}
```

```
int main() {
int n, i, *vet, minimo, maximo;
printf("Quantos numeros voce deseja digitar? ");
scanf("%d", &n);
vet = malloc(n * sizeof(int));
for (i = 0; i < n; i++) {
printf("Digite o numero de indice %d: ", i);
scanf("%d", &vet[i]);
}
mm(vet, n, &minimo, &maximo);
printf("Minimo: %d. Maximo: %d.\n", minimo, maximo);
retum 0;
}
```

5.

Como “v” nos retorna o endereço do primeiro elemento de um vetor, “v + 3”nos retorna o endereço do quarto elemento. Porém, v[3] nos retorna o quartoelemento! A diferença é que em um caso temos o elemento e em outro oendereço do elemento.

6.

O primeiro laço “for” popula o vetor “a” com números decrescentes,começando de 98 (para o índice 0) até 0 (para o índice 98). Já o segundo,onde temos o vetor populado, troca os valores dos índices; ele é maiscomplicado: para cada índice “i”, ele troca o valor do elemento no vetor pelovalor do elemento que possui como índice o valor do elemento a[i]. Dessaforma, até a primeira metade do vetor (índice 49, pois 99 / 2 = 49), os valores são invertidos (de 0 a 49, para os índices de 0 a 49). Porém, para ospróximos itens, o vetor já está

invertido, então ele é sobrescrito com osnovos valores (da primeira metade do vetor), que, por coincidência, são osvalores que estavam anteriormente (quando populamos o vetor, no primeiro“for”, em ordem decrescente).

7.

```
void troca (float *a, float *b) {

float temp;

temp = *a;

*a = *b;

*b = temp;

}
```

8.

```
char *strcopy(char *str) {

int n, i;


char *nova;
//Primeiro vamos contar quantos caracteres a string tem:
for (n = 0; str[n] != '\0'; n++) {}

//Agora vamos copiar alocar memoria para a nova string:
nova = malloc(n * sizeof(char));

//Com a memoria alocada, podemos copiar:
for (i = 0; i <= n; i++) {
nova[i] = str[i];
}

//O "=" eh necessario para copiar tambem o \0 final de str
return nova;
}
```

9.

Idêntica à questão 4, trocando apenas “maximoMinimo” por “mm” (e sem necessidade da função main). Nota: na questão 4 não está explícito que precisamos passar como parâmetro o número de elementos do vetor (mas é necessário).

10.

int x = 100, *p, **pp; //x recebe o valor 100, p é um ponteiro para inteiro e pp é umponteiro para ponteiro para inteiro

p = &x; //p passa a apontar para o endereço de x (logo, *p tem o valor 100)

pp = &p; //pp passa a apontar para o endereço de p, logo *pp é o valor de pp, que éo mesmo que p e **pp é o mesmo que *p, que é o mesmo que x (que é igual a 100)

printf(“Valor de pp: %d\n”, **pp); //imprime o valor de **pp, que é igual ao valor de x, como mencionado na linha acima

11.

```
#include <stdio.h>

#include <stdlib.h>

int *acha_caractere(char *str, char c, int *pn) {

int i = 0, n = 0, *indices = 0;

for (i = 0; str[i] != '\0'; i++) {

if (str[i] == c) {

n++;

}

}

indices = (int *) malloc(n* sizeof(int));

for (i = 0, n = 0; str[i] != '\0'; i++) {

if (str[i] == c) {

indices[n] = i;n++;
```

```
}
}
*pn = n;
return indices;
}

int main() {
int *indices = 0, n = 0, i;
char *frase = "teste";
indices = acha_caractere(frase, 'e', &n);
for (i = 0; i < n; i++) {
printf("%d ", indices[i]);
}
return 0;
}
```

12.

a) int *p; //o asterisco mostra que é uma declaração de um ponteiro

b) cout << *p; //o asterisco faz com que o conteúdo do endereço a ser apontado por p seja mostrado

c) *p = x*5; //o asterisco faz com que o conteúdo do endereço apontado por p se torne x*5

d) cout << *(p+1); //o asterisco faz com que o conteúdo do endereço a ser apontado por (p+1) seja mostrado

13.

//A saída será:

p= O endereço para onde p aponta

(*p+2) = O valor do conteúdo para onde p aponta + 2, que, no caso, dá 7

**&p = O valor do conteúdo para onde p aponta. No caso, 5

(3**p) = O valor do conteúdo para onde p aponta multiplicado por 3, ou seja, 15.

(**&p+4) = O valor do conteúdo para onde p aponta somado a 4. No caso, dá 9.

STRUCTS

1.

```
#include <stdio.h>

typedef struct {
	int matricula;
	char nome[100];
	float nota1;
	float nota2;
} Aluno;


#define QUANTIDADE_DE_ALUNOS 3
```

```
int main(){
	Aluno alunos[QUANTIDADE_DE_ALUNOS];

	printf("Dados: nome(sem espacos), matricula, nota1, nota2\n");
	for(int i=0; (i < QUANTIDADE_DE_ALUNOS); i++){
		printf("\nInforme os dados do aluno(%i): ",i+1);
```

```
            scanf("%s %i %f %f",alunos[i].nome,
&alunos[i].matricula,
                    &alunos[i].nota1, &alunos[i].nota2);
        }

        printf("\nMatricula\tNome\tMedia\n");
        for(int i=0; (i < QUANTIDADE_DE_ALUNOS); i++){

printf("%i\t%s\t%1.2f\n",alunos[i].matricula,alunos[i].nome,
                    (alunos[i].nota1 + alunos[i].nota2)/2);
        }

        getchar();
        return 0;
}
```

2.

```
#include <stdio.h>

typedef struct{
        char  nome[100];
        char  sexo; // 'm': masculino, 'f': femino
        float peso;
        float altura;
        long long cpf;
} Pessoa;

#define QUANTIDADE_DE_PESSOAS 3

int main(){
        Pessoa pessoas[QUANTIDADE_DE_PESSOAS];

        printf("Campos: nome, altura, peso, cpf, sexo\n");
```

```
        for(int i=0; (i < QUANTIDADE_DE_PESSOAS); i++){
                printf("\nInforme os dados da pessoa(%i): ",i+1);
                scanf("%s %f %f %Lu %c",pessoas[i].nome,
&pessoas[i].altura,
                        &pessoas[i].peso, &pessoas[i].cpf,
&pessoas[i].sexo);
        }

        printf("\nInforme o CPF da pessoa: ");
        long long cpf_localizador;
        scanf("%Lu",&cpf_localizador); //

        printf("\nSexo\tNome\tIMC");
        for(int i=0; (i < QUANTIDADE_DE_PESSOAS); i++){ //
                if (cpf_localizador == pessoas[i].cpf){ //
                        float imc = pessoas[i].peso /
(pessoas[i].altura *
                                pessoas[i].altura);
                        printf("\n%c\t%s\t%1.2f\n",pessoas[i].sexo,
                                pessoas[i].nome, imc);
                        break;
                }
        }

        getchar();
        return 0;
}
```

3.

```
#include <stdio.h>

typedef struct {
        long  codigo;
```

```
        char  nome[100];
        float preco;
} Produto;

#define QUANTIDADE_DE_PRODUTOS 5

int main(){
        Produto produtos[QUANTIDADE_DE_PRODUTOS];

        printf("Campos: codigo-do-produto nome preco\n");
        for(int i=0; (i < QUANTIDADE_DE_PRODUTOS); i++){
                printf("\nInforme os dados do produto(%i): ",i+1);
                scanf("%ld %s
%f",&produtos[i].codigo,produtos[i].nome,
                        &produtos[i].preco);
        }

        for(int i=0; (i < QUANTIDADE_DE_PRODUTOS); i++){
                printf("\n%ld\t%s R$ %1.2f", produtos[i].codigo,
                        produtos[i].nome,produtos[i].preco);
        }

        long codigo_digitado;
        printf("\nInforme o codigo do produto: ");
        scanf("%ld", &codigo_digitado);

        for(int i=1; (i < QUANTIDADE_DE_PRODUTOS); i++){
                if (produtos[i].codigo == codigo_digitado) {
                        printf("\nPreço: R$ %1.2f\n",
produtos[i].preco);
                }
        }

        getchar();
        return 0;
```

```
}
```

4.

```
#include <stdio.h>

typedef struct  {
        char nome[256];
        long long cpf;
} Cliente;

typedef struct {
        long    numero_da_conta;
        long    cpf_do_cliente;
        double  saldo;
} Conta;

#define QUANTIDADE_DE_CLIENTES 3
#define OPERACAO_SAQUE 1
#define OPERACAO_DEPOSITO 2

int main(){
  Cliente clientes[QUANTIDADE_DE_CLIENTES];
  Conta   contas[QUANTIDADE_DE_CLIENTES];

  printf("Campos: cpf nome deposito-inicial\n");
  for(long i=0; (i < QUANTIDADE_DE_CLIENTES); i++){
    printf("\nDados para abertura da conta(%ld): ",i+1);
    scanf("%Ld %s %lf",&clientes[i].cpf,clientes[i].nome,
      &contas[i].saldo);

    contas[i].numero_da_conta = i;
    contas[i].cpf_do_cliente = clientes[i].cpf;
```

```
    printf("\nCliente: %s Conta: %ld Saldo inicial: %1.2lf\n",
      clientes[i].nome, contas[i].numero_da_conta,
contas[i].saldo);
  }

  int operacao; // como ainda não aprendemos a comparar
strings,
              // vamos usar 'operação' como numérico.
  long num_conta;
  double valor;
  int sair=0; // FALSE

  while (!sair){
    printf("\nInforme a operação: 1-Saque 2-Deposito 3-Sair:
");
    scanf("%d", &operacao);

    if (operacao == OPERACAO_SAQUE || operacao ==
OPERACAO_DEPOSITO){
      printf("\nInforme numero-da-conta e valor: ");
      scanf("%ld %lf", &num_conta, &valor);
      for(int i=0; (i < QUANTIDADE_DE_CLIENTES); i++){
        if (contas[i].numero_da_conta == num_conta) {
          if (operacao == OPERACAO_SAQUE){
            contas[i].saldo -= valor;
            printf("\nSAQUE: %1.2lf", valor);
          }
          if (operacao == OPERACAO_DEPOSITO){
            contas[i].saldo += valor;
            printf("\nDEPOSITO: %1.2lf", valor);
          }
          for(int j=0; j < QUANTIDADE_DE_CLIENTES; j++){
            if (clientes[j].cpf == contas[i].cpf_do_cliente)
              printf("\nCliente: %s Saldo atual: %1.2lf",
                clientes[j].nome, contas[i].saldo);
```

```
        }
       }
     }
   }else{
     sair = 1; // TRUE
   }
 }

 getchar();
 return 0;
}
```

5.

```
Lista* retira_ultimo (Lista* l)
{
if(l == NULL) return NULL;
else    {
        Lista* ant = NULL;
        Lista* p = l;
        while(p->prox !=NULL)      {
                                   ant = p;
                                   p = p->prox;
                                   }
        free (p);
        if (ant!=NULL)          {
                                ant -> prox = NULL;
                                return l;
                                }
        else return NULL;
```

```
        }
}
```

6.

```
Lista* constroi(int n, int* v)
{
  int i;
  Lista* head=NULL;
  for (i=0;i<n;i++) {
    Lista* novo=(Lista*) malloc(sizeof(Lista));
    novo->info=v[i];
    novo->prox=head;
    head=novo;
  }
  return head;
}
```

7.

```
struct lista {
   char nome[81];
   float nota;
   struct lista* prox
};
typedef struct lista Lista;
Lista* insere (Lista* lst, char* nome, float nota)
{
        Lista* novo=(Lista*) malloc(sizeof(Lista));
```

```
        strcpy(novo->nome,nome);
        novo->nota=nota;
        novo->prox=lst;
        return novo;
}
```

8.

a) O nó acessado sera o x.

b) O nó acessado sera o z.

c) O nó acessado sera o b.

9.

a) O valor de x será (55 * 17) – 3 = 932.

b) O valor de x será (3*17 – 40) = 11.

c) O valor de x será (40 / 8 == 5) = TRUE

10. Como os nós tem sempre valores diferentes, a lista pode ter 1, 2 ou 4 nós para que a função retorne verdadeiro pois, assim, pri->prox->prox e pri->ant->ant referenciarão o mesmo nó, satisfazendo a equação.