UFJF – ICE – Departamento de Ciência da Computação

Disciplina: Laboratório de Programação II

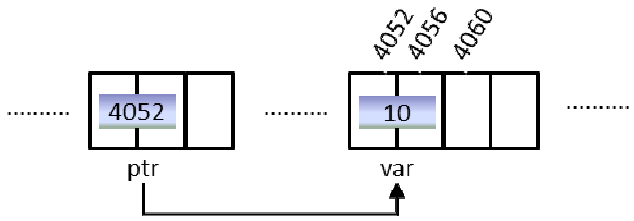
Professores: Itamar/Saulo/Stênio

Lista de Exercícios 1 (ponteiros)

1) Fazer um esquema representando a memória do computador ao se executar a sequencia de comandos:

```
int var;
int *ptr;
var = 10;
ptr = &var;
```

Resposta:



2) Um ponteiro é:

a) o endereço de uma variável;
b) uma variável que armazena endereços;
c) o valor de uma variável;
d) um indicador da próxima ariável a ser acessada.

3) Escreva uma instrução que imprima o endereço da variável `var`.

4) Indique: (1) operador de endereços
(2) operador de referência (de indireção)

a) `p = &i;`
b) `int &r = j;`
c) `printf("%p",&i);`
d) `int *p = &i;`

5) O que significa o operador asterisco em cada um dos seguintes casos:

a) `int *p;`
b) `printf("*d",*p);`
c) `*p = x*5;`
d) `printf("*d",*(p + 1));`

6) Quais das seguintes instruções declaram um ponteiro para uma variável `float`:

a) `float *p;`
b) `*float p;`
c) `float* p;`
d) `float *p = &f;`
e) `*p;`
f) `float& p = q;`

7) Se o endereço de `var` foi atribuído a um ponteiro variável `pvar`, quais das seguintes expressões são verdadeiras?

a) `var == &pvar;`
b) `var == *pvar;`
c) `pvar == *var;`
d) `pvar == &var;`

8) Considere as declarações

```
int i = 3, j = 5;
int *p = &i, *q = &j;
```

Indique qual é o valor das seguintes expressões:

a) `p == &i;`
b) `*p - *q;`
c) `**&p;`
d) `3*-*p/*q+7;`
e) `*p - *q;`

9) Qual é a saída deste programa?

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
  int i = 5, *p;
  p = &i;
  printf("%p\t%d\t%d\t%d\t%d\t",p ,(*p+2),**&p,(3**p),(**&p + 4));
  return 0;
}
```

10) Se `i` e `j` são variáveis inteiras e `p` e `q` ponteiros para `int`, quais das seguintes expressões de atribuição são incorretas?

a) `p == &i;`
b) `*q = &j;`
c) `P = &*&i;`
d) `i = (*&)j;`
e) `i = *&*&j;`
f) `q = &p;`
g) `i = (*p)++ + *q;`
h) `if(p == i) i++;`

10) O seguinte programa está correto? Justifique.

```
#include <stdio.h>
#include <stdlib.h>
#define VAL 987
int main()
{
  int *p = VAL;
  printf("%d", *p);
  return 0;
}
```

Obs: o erro será em tempo de execução, quando se usa o Code::Blocks.

11) O seguinte programa está correto? Justifique.

```
#include <stdio.h>
#include <stdlib.h>
#define VAL 987
int main()
{
  int i = VAL;
  int *p;
  printf("%d", *p);
  return 0;
}
```

12) Qual a diferença entre: `mat[3]` e `*(mat + 3)`?

13) Admitindo a declaração: `int mat[8];` por que a instrução `mat++;` é incorreta?

R. Está incorreta, pois o nome de um vetor (matriz) é um ponteiro constante. Não se pode alterar (neste caso, incrementar) uma constante. O comando, por exemplo, `x = 3++` também está errado.

14) Admitindo a declaração: `int mat[8];`. Quais das seguintes expressões referenciam o valor do terceiro elemento da matriz?

a) `*(mat + 2);`
b) `*(mat + 3);`
c) `mat + 2;`
d) `mat + 3;`

15) O que faz o programa seguinte (o que será impresso?):

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  int mat[] = {4, 5, 6};
  for(int i=0; i<3; i++)
    printf("%d\n", *(mat + i));
  return 0;
}
```

16) O que faz o programa seguinte (o que será impresso?):

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  int mat[] = {4, 5, 6};
  for(int i=0; i<3; i++)
    printf("%d\n", mat + i);
  return 0;
}
```

17) O que faz o programa seguinte (o que será impresso?):

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  int mat[] = {4, 5, 6};
```

```
  int *p = mat;
  for(int i=0; i<3; i++)
    printf("%d\n", *p++);
  return 0;
}
```

18) Qual a diferença entre as duas instruções seguintes?

```
char s[] = “Brasil”;
char *s = “Brasil”;
```

19) Assumindo a declaração:

```
char *s = “Eu não vou sepultar Cesar”;
```

O que será impresso em cada uma das instruções:

a) `printf("%s",s);`
b) `printf("%p",&s[0]);;`
c) `printf("%p",s + 11);`
d) `printf("%c",s[0]);`
e) `printf("%p",s);`

20) Escreva a expressão `mat[i][j]` em notação de ponteiro.

21) Qual é a diferença entre os seguintes protótipos de funções:

```
void func(char *p);
void func(char p[]);
```

22) Assumindo a declaração:

```
char *items[5] = { "Abrir",
                   "Fechar",
                   "Salvar",
                   "Imprimir",
                   "Sair"
                 };
```

Para poder escrever a instrução `p = items;` a variável `p` deve ser declarada como:

a) `char p;`
b) `char *p;`
c) `char **p;`
d) `char ***p;`

23) Será feito uma aplicação passo a passo para um programa de restaurante.

a) Crie uma estrutura para descrever restaurantes. Os membros devem armazenar o nome, o endereço, o preço médio e o tipo de comida.

```
struct rest {
                char nome[15];
                char endereco[15];
                float preco;
                char tipo[10];  //italiana, francesa, mineira,...
              };
typedef struct rest Trest; //T representa tipo
```

b) Considerando que existem vários restaurantes, definir um vetor que contenha os dados de todos os restaurantes. Este vetor deve armazenar um ponteiro para cada estrutura `Trest` (restaurante). Não se sabe, de antemão, o número de restaurantes.

```
  Trest **nRest;
```

c) Fazer uma função para alocar uma estrutura `Trest` e retornar o endereço desta estrutura. Além disso, essa função deve ler os dados de um restaurante.

```
Trest *LeiaRest(void)
//aloca memória e lê os dados para um restaurante.
//Não esquecer de desalocar a memória após o uso da estrutura
{
  Trest *r; //armazena dados de um restaurante
  r = (Trest*) malloc(sizeof(Trest));
  printf("\nNome do restaurante:");
  gets(r->nome);
  printf("\nEndereco:");
  gets(r->endereco);
  printf("\nTipo de restaurante:");
  gets(r->tipo);
  printf("\nPreco medio:");
  scanf("%f",&r->preco);
  return(r);
}
```

d) Fazer uma função para ler os dados, em um vetor, de n restaurantes. Esta função terá como parâmetro o número inteiro n (tamanho do vetor) e deve alocar memória suficiente para armazenar os dados dos restaurantes, além, obviamente, de ler todos os dados. O retorno da função deve ser um ponteiro para o vetor lido.

```
Trest **LeiaVariosRest(int n)
//aloca memória e lê os dados para n restaurantes.
//Não esquecer de desalocar a memória após o uso do vetor
{
  Trest **nRest;
  int i;
  nRest = (Trest**) malloc(n*sizeof(Trest));
  for(i=0; i<n; i++)
    nRest[i] = LeiaRest();
  return(nRest);
}
```

d) Fazer um procedimento para desalocar toda a memória ocupada pelo vetor de restaurantes, dados o vetor o seu tamanho n.

```
void desaloca(Trest **vetRest, int n)
//desaloca o vetor vetRest
{
  int i;
  //desaloca a memória de cada struct (restaurante)
  for(i=0; i<n; i++)
    free(vetRest[i]);
  //desaloca o vetor
  free(vetRest);
}
```

f) Fazer a função `main()` para 5 restaurantes e imprimir o preco de cada um.

```
int main()
{
  Trest **r;
  int n = 5,i;
  r = LeiaVariosRest(n);
  for(i=0; i<n; i++)
    printf("\n%f", r[i]->preco);
  desaloca(r,n);
  return 0;
}
```