Exercícios Teóricos – u00m: Linguagem C

Catarina F. M. Castro (803531) – AEDs II

1-

- Faça o quadro de memória e mostre a saída na tela:

```
int *x1;          int x2;          int *x3;

x1 = (int *) malloc (sizeof(int));
printf("\nx1(%p)(%i)(%p) x2(%i)(%p) x3(%p)(%i)(%p)", x1, *x1, &x1, x2, &x2, x3, *x3, &x3);

*x1 = 20;
printf("\nx1(%p)(%i)(%p) x2(%i)(%p) x3(%p)(%i)(%p)", x1, *x1, &x1, x2, &x2, x3, *x3, &x3);

x2 = *x1;
printf("\nx1(%p)(%i)(%p) x2(%i)(%p) x3(%p)(%i)(%p)", x1, *x1, &x1, x2, &x2, x3, *x3, &x3);

*x3 = x2 * *x1;
printf("\nx1(%p)(%i)(%p) x2(%i)(%p) x3(%p)(%i)(%p)", x1, *x1, &x1, x2, &x2, x3, *x3, &x3);

x3 = &x2;
printf("\nx1(%p)(%i)(%p) x2(%i)(%p) x3(%p)(%i)(%p)", x1, *x1, &x1, x2, &x2, x3, *x3, &x3);

x2 = 15;
printf("\nx1(%p)(%i)(%p) x2(%i)(%p) x3(%p)(%i)(%p)", x1, *x1, &x1, x2, &x2, x3, *x3, &x3);
```

*x1 (89Bh)	x2 (89Ch)	*x3 (89Dh)	Tela
20	89Bh	20*89Bh	x1(89Bh)(null)(89Bh) x2(null)(89Ch) x3(89Dh)(null)(89Dh)
	15	89Ch	x1(89Bh)(20)(89Bh) x2(null)(89Ch) x3(89Dh)(null)(89Dh)
			x1(89Bh)(20)(89Bh) x2(20)(89Ch) x3(89Dh)(null)(89Dh)
			x1(89Bh)(20)(89Bh) x2(20)(89Ch) x3(89Dh)(20*89Bh)(89Dh)
			x1(89Bh)(15)(89Bh) x2(20)(89Ch) x3(89Dh)(89Ch)(89Dh)

2-

- Faça o quadro de memória:

```
double M [3][3];
double *p = M[0];
for (int i = 0; i < pow(MAXTAM, 2); i++, p++){
    *p=0.0;
}
```

M[0][0]	M[0][1]	M[0][2]	M[1][0]	M[1][1]	M[1][2]	M[2][0]	M[2][1]	M[2][2]	*p
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	Endereço de M[0]

3-

- Mostre a saída na tela

```
double a;
double *p, *q;
a = 3.14;
printf("%f\n", a);
p = &a;
*p = 2.718;
printf("%f\n", a);
a = 5;
printf("%f\n", *p);
```

```
p = NULL;
p = (double*) malloc(sizeof(double));
*p = 20;
q = p;
printf("%f\n", *p);
printf("%f\n", a);
free(p);
printf("%f\n", *q);
```

Tela
3.14
77h
5
20
5
NULL

4-

- Mostre o quadro de memória

```
int a[10], *b;
b = a;
b[5] = 100;
printf("\n%d -- %d", a[5], b[5]);

b = (int*) malloc(10*sizeof(int));
b[7] = 100;
printf("\n%d -- %d", a[7], b[7]);

//O comando a = b gera um erro de compilação
```

b	a	Tela
b[5] = 100	a[5] = 100	100 – 100
b[7] = 100		77h – 100

5-

- Mostre o quadro de memória

```
int *x1;          int x2;          int *x3;
x1 = (int*) malloc(sizeof(int));
*x1 = 20;
x2 = *x1;
*x3 = x2 * *x1;
x3 = &x2;
x2 = 15;
x2 = 13 & 3;
x2 = 13 | 3;
x2 = 13 >> 1;
x2 = 13 << 1;
```

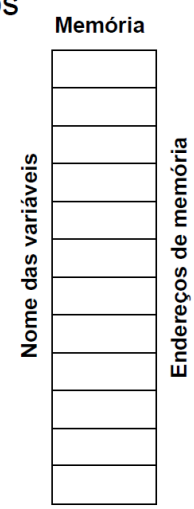
***x1**	77h	7Ah		
x2	78h	20	20*77h	15
***x3**	79h	78h		
endereço4	7Ah	20		

6-

- Execute o programa abaixo, supondo os atributos código (int) e salário (double) para cada Cliente

```
Cliente c;
c.codigo = 5;
Cliente *p = NULL;
p = (Cliente*) malloc (sizeof(Cliente));
p->codigo = 6;
Cliente *p2 = &c;
p2->codigo = 7;
```

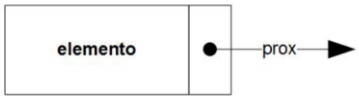
Representação gráfica



c	77h	5 / ?	7 / ?	
***p**	78h	NULL	89h	
***p2**	79h	77h		
end1	89h	6 / ?		

7-

· Crie um registro célula contendo os atributos elemento (inteiro) e prox (apontador para outra célula)



```
typedef struct Celula {
        int elemento;
        struct Celula *prox;
        } Celula;

Celula *novaCelula(int elemento) {
        Celula *nova = (Celula*) malloc(sizeof(Celula));
        nova->elemento = elemento;
            nova->prox = NULL;
}
```

8-

· Mostre o que acontece se um método tiver o comando *Celula *tmp = novaCelula(3)*.

```
typedef struct Celula {
    int elemento;
    struct Celula *prox;
} Celula;

Celula *novaCelula(int elemento) {
    Celula *nova = (Celula*) malloc(sizeof(Celula));
    nova->elemento = elemento;
    nova->prox = NULL;
    return nova;
}
```

O ponteiro *tmp irá receber um ponteiro que aponta para um espaço de memória com o valor 3 salvo em elemento, e o valor NULL salvo em prox.

9-

· Represente graficamente o código Java abaixo

```
Elemento e1;
```

e1 → ?

10-

• Represente graficamente o código Java abaixo

```
Elemento e1 = new Elemento();
```

e1 → [?]

11-

• Represente graficamente o código Java abaixo

```
Elemento[] e2 = new Elemento [3];
```

e2 → [?][?][?]

12-

• Represente graficamente o código Java abaixo

```
Elemento[] e2 = new Elemento [3];

for (int i = 0; i < 3; i ++){
    e2[i] = new Elemento();
}
```

e2 → [ ] → [ ]
[ ] → [ ]
[ ] → [ ]

13-

• Represente graficamente o código C abaixo

```
Elemento e1;
```

e1[ ]

14-

• Represente graficamente o código C abaixo

```
Elemento* e2;
```

e2 → ?

15-

- Represente graficamente o código C abaixo

```
Elemento* e2 = (Elemento*) malloc(sizeof(Elemento));
```

e2 → [?]

16-

- Represente graficamente o código C abaixo

```
Elemento* e2 = (Elemento*) malloc(3*sizeof(Elemento));
```

e2 → [ ] → [ ]
[ ] → [ ]
[ ] → [ ]

17-

- Represente graficamente o código C abaixo

```
Elemento e3[3];
```

e3[ ][ ][ ]

18-

- Represente graficamente o código C abaixo

```
Elemento** e4;
```

e4 → ?

19-

- Represente graficamente o código C abaixo

```
Elemento** e4 = (Elemento**) malloc(3*sizeof(Elemento*));
```

e4 → [ ] → [ ]
[ ] → [ ]
[ ] → [ ]

20-

- Represente graficamente o código C abaixo

```
Elemento** e4 = (Elemento**) malloc(3*sizeof(Elemento*));
e4[0] = (Elemento*) malloc(sizeof(Elemento*));
e4[2] = (Elemento*) malloc(sizeof(Elemento*));
```

e4 → [ ] → [ ]
[ ] → ?
[ ] → [ ]

21-

- Represente graficamente o código C++ abaixo

```
Elemento e1;
```

e1[ ]

22-

- Represente graficamente o código C++ abaixo

```
Elemento* e2;
```

e2 → ?

23-

- Represente graficamente o código C++ abaixo

```
Elemento* e2 = new Elemento;
```

e2 → [ ]

24-

- Represente graficamente o código C++ abaixo

```
Elemento* e2 = new Elemento[3];
```

e2 → [ ][ ][ ]

25-

- Represente graficamente o código C++ abaixo

```
Elemento e3[3];
```

e3[ ][ ][ ]

26-

- Represente graficamente o código C++ abaixo

```
Elemento** e4;
```

e4 → ?

27-

- Represente graficamente o código C++ abaixo

```
Elemento** e4 = new Elemento*[3];
```

e4 → [ ] → ?
[ ] → ?
[ ] → ?

28-

- Represente graficamente o código C++ abaixo

```
Elemento** e4 = new Elemento*[3];
e4[0] = new Elemento;
e4[2] = new Elemento;
```

e4 → [ ] → [ ]
[ ] → ?
[ ] → [ ]