EXERCÍCIOS DE REVISÃO DE TÓPICOS DA LINGUAGEM C (ANSI)
TADS

1. Responda:

   a) Você conhece as diretivas de compilação? Descreva as seguintes diretivas:

      #include, #define, #undef, #ifdef, #ifndef, #if , #else, #elif, #endif

   b) Qual é o valor de *p* na função *main(.)* abaixo? Qual a importância de inicializar apontadores? Na coluna esquerda, o bloco de código declara/aloca alguma variável na memória?

```
struct teste{
        int inteiro;
        float real;
        char nome[30];
};
```

```
main(void)
{
        struct teste *p, x = {321, 2.39, "Silva"};

}
```

   c) Para o código e representação de mapa de memória exibidos abaixo, responda:

      Qual é o valor final de *p*?
      Qual é o valor final de **p*?
      Qual é o valor final de *x*?
      Qual é o valor final de *&x*?
      Qual é o valor final de pp?
      Qual é o valor final de *pp?
      Qual é o valor final de **pp?
      Qual é o valor final de *y*?

```
main(void)
{
        int *p=NULL, **pp=NULL, x = 321, y=101;
        p = &x;
        pp=&p;
        *p= -3;
        y=**pp;

}
```

Campo	Endereço	Conteúdo
p	00007FFC127C92C8	
x	00007FFC127C92C0	
y	00007FFC127C92C4	
pp	00007FFC127C92D0	
*pp == p		
**pp == *p == x		

d) Para o código e representação de mapa de memória exibidos abaixo, responda:

Qual é o valor de *p*?
Qual é o valor de p→self?

```
struct teste2{      int inteiro;
                    float real;
                    char nome[30];
                    struct teste2 *self;
        };
```

```
        struct teste2 x = {115, 2.5, "Smith", NULL},*p=NULL;
        p = &x;
        p->self = &x;
```

Campo	Endereço	Conteúdo
x	00007FFE0F4FB570	
p	00007FFE0F4FB568	
p->self	00007FFE0F4FB598	

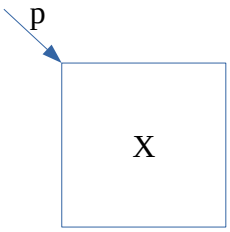
e) Para o código e representação de mapa de memória exibidos abaixo, responda:

Qual é o valor de *p*?
Qual é o valor de &(p→real)?
Qual é o valor de p→real?
Qual é o valor de &(p→apont)?
Qual é o valor de p→apont?
Qual é o valor de *(p→apont)?

```
struct teste{
        int inteiro;
        float real;
        char nome[30];
        char rua[30];
        int *apont;
        };
```

```
main(void)
{
        struct teste *p=NULL, x = {321, 2.39, "Silva", "Timbo", NULL};
        int y=101;
        p = &x;
        p→apont = &y;

}
```



Campo	Endereço	Conteúdo
p	00007FFDE307C4A8	
p->inteiro	00007FFDE307C4B0	
p->real	00007FFDE307C4B4	
p->nome	00007FFDE307C4B8	
p->rua	00007FFDE307C4D6	
p->apont	00007FFDE307C4F8	
...	...	
y	00007FFDE307C4A4	

2. Para a *struct* descrita abaixo, encontre e solucione os erros/equívocos nos exemplos abaixo:

```
struct teste{       int inteiro;
                    float real;
                   char nome[30];

      };
```

A)
```
main(void)
{       struct teste *p, x = {321, 2.39, "Silva"};
        printf("Campos da variável x: %i, %f, %s", p→inteiro, p→real, p->nome);

}
```

B)
```
main(void)
{       struct teste *p, x = {321, 2.39, "Silva"};
        p= (struct teste *) malloc(sizeof(struct teste));
        printf("Campos da variável x: %i, %f, %s", p→inteiro, p→real, p->nome);

}
```

C)
```
main(void)
{       struct teste *p, x = {321, 2.39, "Silva"};
        p = x;
        printf("Campos da variável x: %i, %f, %s", p→inteiro, p→real, p->nome);

}
```

D)
```
main(void)
{       struct teste *p, x = {321, 2.39, "Silva"};
        p = &x;
        printf("Campos da variável x: %i, %f, %s", p.inteiro, p.real, p.nome);

}
```

E)
```
main(void)
{       struct teste *p, x = {321, 2.39, "Silva"};
        p= (struct teste *) malloc(sizeof(struct teste));
        p=&x;
        printf("Campos da variável x: %i, %f, %s", p→inteiro, p→real, p->nome);

}
```

F)
```
main(void)
{
        int a=10, vet[]={1,2,3,4,5}, *p=NULL;
        float b=35.75;
        info y = {31,"Wilson"};
        void *ptr; // Declaracao de um ponteiro para um tipo genérico (void)
        ptr=&a; // Atribuindo o endereço de um inteiro.
        printf("a = %d \n", * ( (int*) ptr) );
        ptr=&b; // Atribuindo o endereço de um float.
        printf("b = %f \n",*( (float*) ptr) );
        ptr= &y;
        printf("nome= %s, idade = %i \n\n", ((info*) ptr)->nome,((info*) ptr)->idade);
        printf("\n\n Acessando um vetor por aritmetica de ponteiro void\n");
        ptr=&vet[0];
        for (int i =0;i<6;i++,ptr++)
        printf("vet[%i] = %i \n", i, *( (int*) ptr) );
}
```

3. Escreva os comandos em linguagem C que levem: (i) da situação 'A' para a 'B' e (ii) da situação 'A' para a 'C', conforme está ilustrado na Figura 1, onde: *taminfo*, *dados*, *topo* e *abaixo* são campos de structs já criadas e representadas graficamente e o símbolo '→' indica um local referenciado por um apontador.

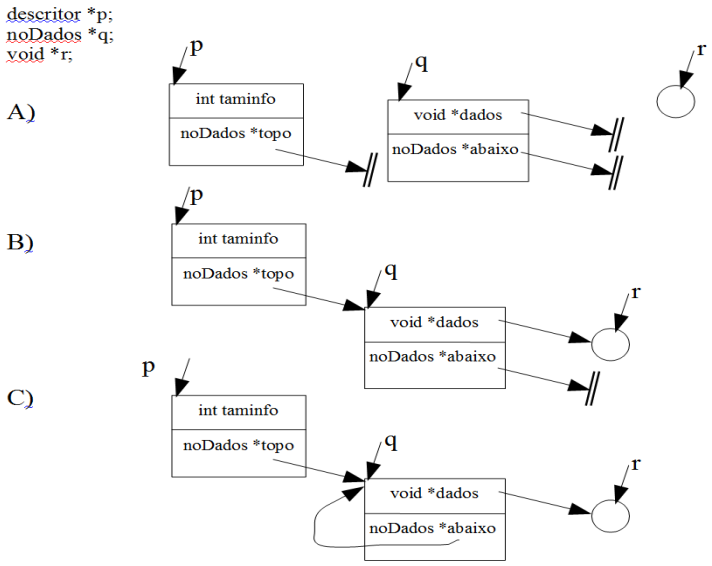


Figura 1: Configurações de nodos.

4. Na Figura 2 e na Figura 3, A e B são sequências encadeadas sofrendo uma inserção. Nas figuras, *pt1*, *pt2* e *pt3* são apontadores já declarados para as respectivas "structs" contendo os campos de ligação: 'a' (aponta para o vizinho esquerdo) e/ou 'b' (aponta para o vizinho direito).
   Nestas condições (sem declarar ou alocar qualquer nova variável) pede-se que você escreva o menor número de comandos, em linguagem *C*, para levar de A1 para A2 e de B1 para B2, conforme as figuras.

```
struct xxx {   int w;
               struct xxx *b;
    }*pt3,*pt2,*pt1;
```

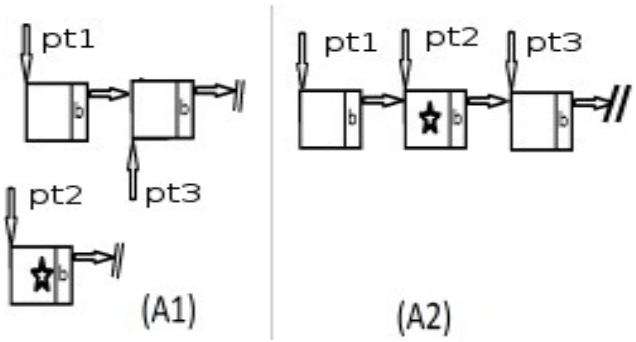


Figura 2: Sequência simplesmente encadeada.

```
struct yyy { int k;
             struct yyy *a;
             struct yyy *b;
      }*pt1,*pt2;
```

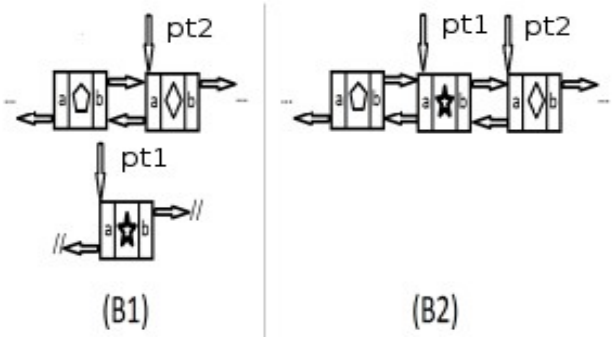


Figura 3: Sequência duplamente encadeada.

5. Escreva os comandos em linguagem *C* que, para cada caso, levem do estado 1 para o estado 2 na figura Figura 4:

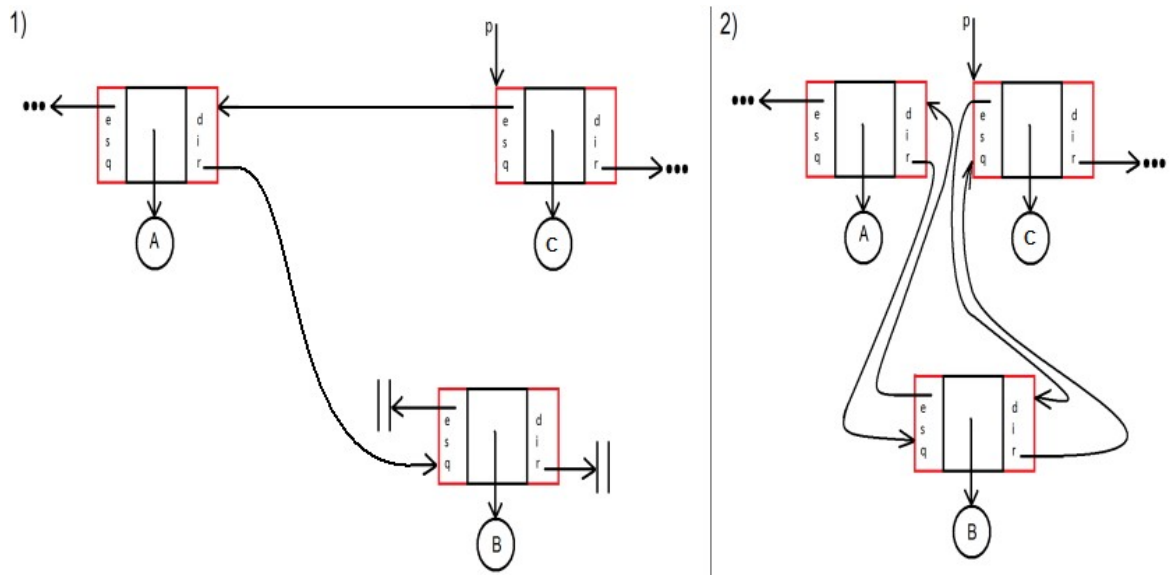


Figura 4: Inserção de elemento duplamente encadeado.

6. Construa a função *int contaNodo(struct nodo *p)* a qual retorna a contagem do número de nós atualmente inseridos em uma lista encadeada conforme a Figura 4 sequências encadeadas..

   Ao final do processo o ponteiro “p” deverá continuar referenciando o nó mais à esquerda.

```
struct nodo{  int x;
              struct nodo *link;
      };
```

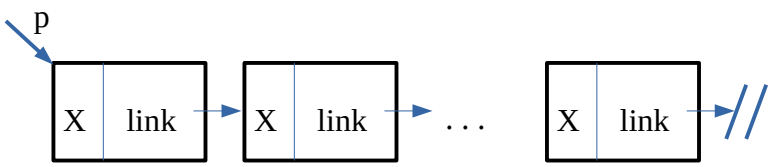


Figura 5: Sequência simplesmente encadeada, finalizada com o Null.

7. Refaça a questão anterior considerando a mudança exibida na Figura 6.

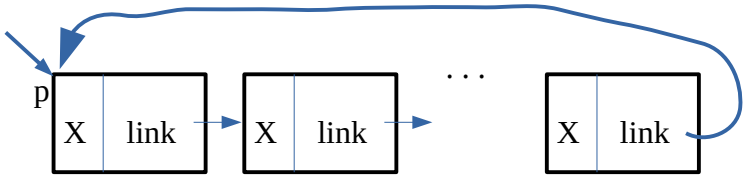


Figura 6: Sequência encadeada circular.

8. Escreva (em 'C') o laço *for(...)* que leva do estado A (Figura 7-A) ao estado B (Figura 7-B). Os dados de inicialização são fornecidos pelo usuário. No descritor, *tamVet* se refere ao tamanho do vetor.

```
typedef struct nodo{          int matricula;
                              char *nome;
                    }data;

typedef struct desc{  int tamVet;
                        data *vet;
                    }descritor;
. . .
p→vet = (data *) malloc(p→tamVet * sizeof(data));

for(int i=0; _____________________; _____________________)

                  ____________________________________________________;
```

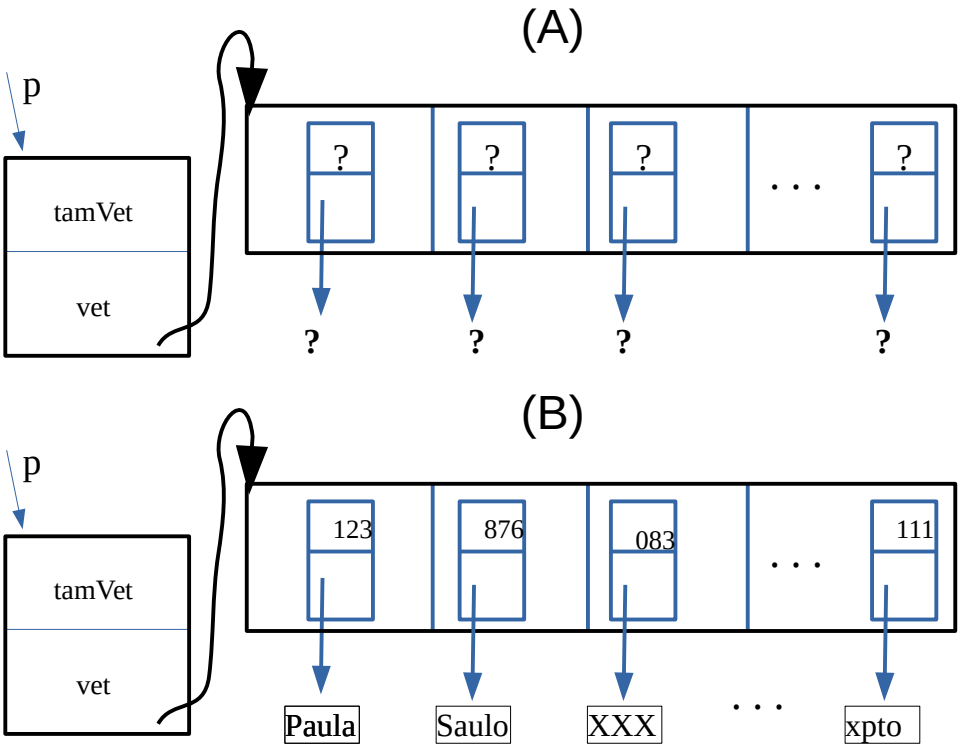


Figura 7: A) vetor de apontadores a serem inicializados, B) vetor inicializado.

9. Construa a função *int insere(Descritor *p, Nodo *novo, int pos)* a qual insere o *novo* item na posição *pos* em uma sequência simplesmente encadeada (exemplo na Figura 4) não vazia. A função deve retornar zero ou um a depender da operação falhar ou ter sucesso. Um requisito adicional é que a posição *pos* já exista na sequência. Escreva um teste de mesa (execução do algoritmo em papel).

```
typedef struct nodo{struct data dt;
                    struct nodo *link;
                    }Nodo;
```

```
typedef struct descritor{   int tamanho;
                            Nodo *inicio;
                    }Descritor;
```

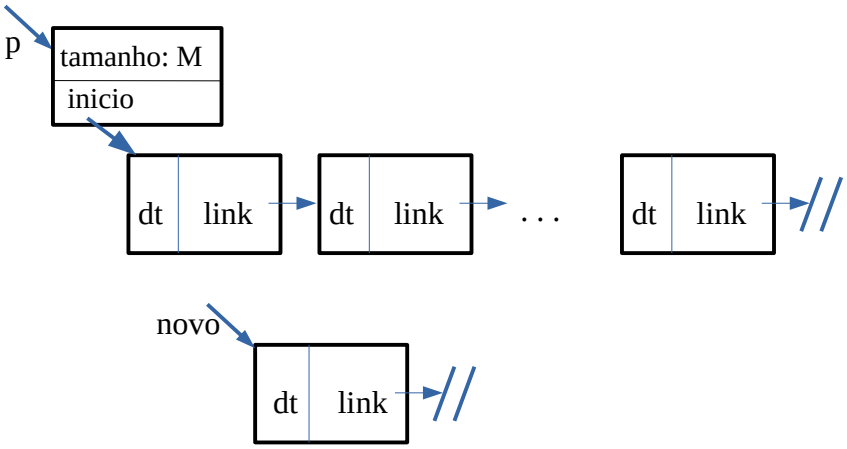


Figura 8: Sequência simplesmente encadeada com descritor.

10. Construa a função *int insereNovoItemAtivo(Descritor *p, Info *novo)* a qual insere o *novo* item na posição *itensAtivos+1* e atualiza o descritor apontado por *p* (exemplo na Figura 9). A função deve retornar zero ou um a depender da operação falhar ou ter sucesso. O campo *itensAtivos* corresponde ao número de itens inseridos a partir do índice zero. O que acontece se *tamVet==itensAtivos*?

```
typedef struct ponto{ float x;
                      float y;
                    }Info;
```

```
typedef struct descritor
                  { int tamVet;
                    int itensAtivos;
                    info *vet;
                  }Descritor;
```

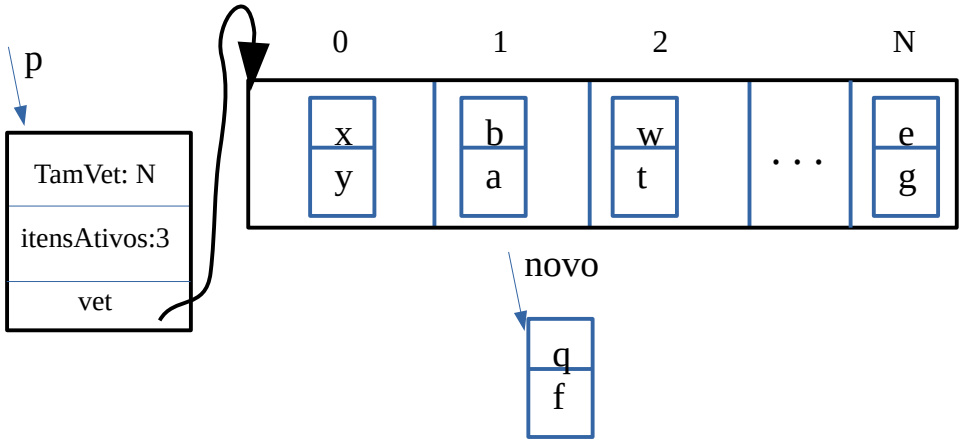


Figura 9: sequência de itens sobre um vetor.

11. Para a questão anterior, se *tamVet==itensAtivos* o vetor não aceitará mais inserções. Reimplemente a função *int insereNovoItemAtivo(Descritor *p, Info *novo)* de maneira que, se ocorrer a condição de vetor cheio, o mesmo seja acrescido de mais 10 posições por meio o uso da função *realloc( ) e a inserção prossiga.*

12. Abaixo temos dois apontadores do tipo "*struct teste"* e apontando para regiões diferentes na memória, complete o comando "memcpy" que executa a cópia da instância de "*struct teste"* apontada por *Q* para a instância de "struct teste" apontada por *P*.

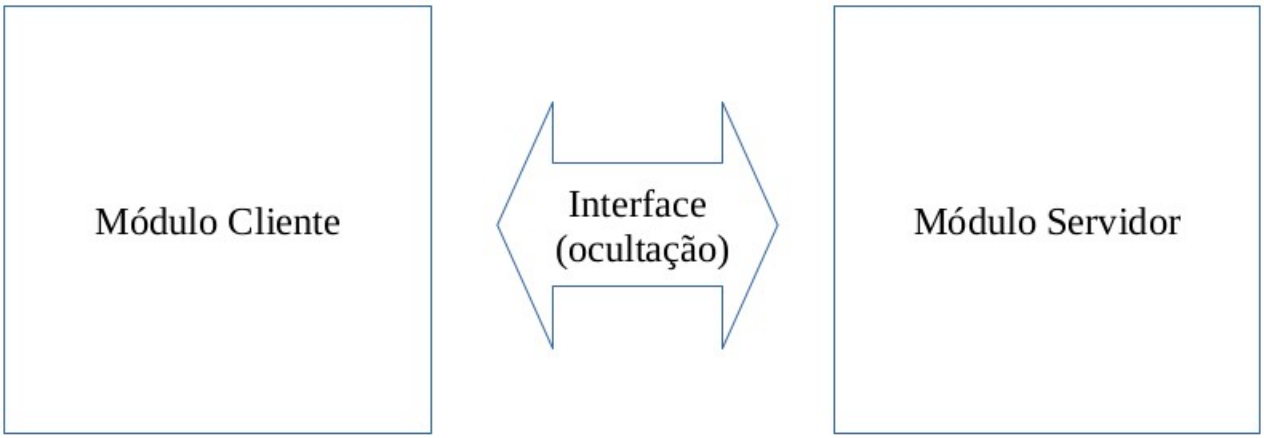
memcpy*(_____, _____ , _____)*

```
typedef struct teste{       char nome[30];
                            char telefone[20];
                            char RG[20];
                    }tipoTeste;

void main(void)
{       tipoTeste *P, Q={"NOME", "1234567", "9876541"};
        P= (tipoTeste *) malloc(sizeof(tipoTeste));

}
```

13. Discuta a utilização da diretiva include para separar um sistema em arquivos diferentes visando a ocultação de implementação e dados (encapsulamento).



Respostas dos itens

1.c)

Campo	Endereço	Conteúdo
p	00007FFC127C92C8	00007FFC127C92C0
x	00007FFC127C92C0	-3
y	00007FFC127C92C4	-3
pp	00007FFC127C92D0	00007FFC127C92C8
*pp == p	&p	00007FFC127C92C0
**pp == *p == x	&x	-3

1.d)

Campo	Endereço	Conteúdo
&x	00007FFE0F4FB570	
p	00007FFE0F4FB568	00007FFE0F4FB570
p->self	00007FFE0F4FB598	00007FFE0F4FB570

1.e)

Campo	Endereço	Conteúdo
p	00007FFDE307C4A8	00007FFDE307C4B0
p->inteiro	00007FFDE307C4B0	321
p->real	00007FFDE307C4B4	2.39
p->nome	00007FFDE307C4B8	Silva
p->rua	00007FFDE307C4D6	Timbo
p->apont	00007FFDE307C4F8	00007FFDE307C4A4
...	...	
y	00007FFDE307C4A4	101