EXERCÍCIOS DE REVISÃO DE TÓPICOS DA LINGUAGEM C (ANSI)

1. Responda:

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

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

b) Leia sobre os comandos memcpy e typedef.

c) Para o código e representação de mapa de memória exibidos abaixo, responda os valores finais das seguintes expressões?:

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

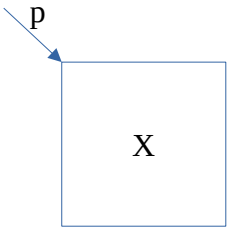
e) Para o código e representação de mapa de memória exibidos abaixo, responda quais serão os valores finais das seguintes expressões?

p
&(p->real)
p->real
&(p->apont)
p->apont
*(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:

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

        } informacao;
```

A)
```
main(void)
{
    informacao *p, x = {321, 2.39, "Silva"};
    if (p)
          printf("valores da struct X: %i, %f, %s", p->inteiro, p->real, p->nome);
     else
          printf("o ponteiro está anulado");
}
```

B)
```
main(void)
{       informacao *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)
{       informacao *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)
{       informacao *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)
{       informacao *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;
        informacao 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 da situação 'A' para a 'B' e da situação 'A' para a situação 'C', conforme está ilustrado abaixo, 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.

typedef struct nodados{ int tamInfo;
*struct nodados *pt*
}NoDados;

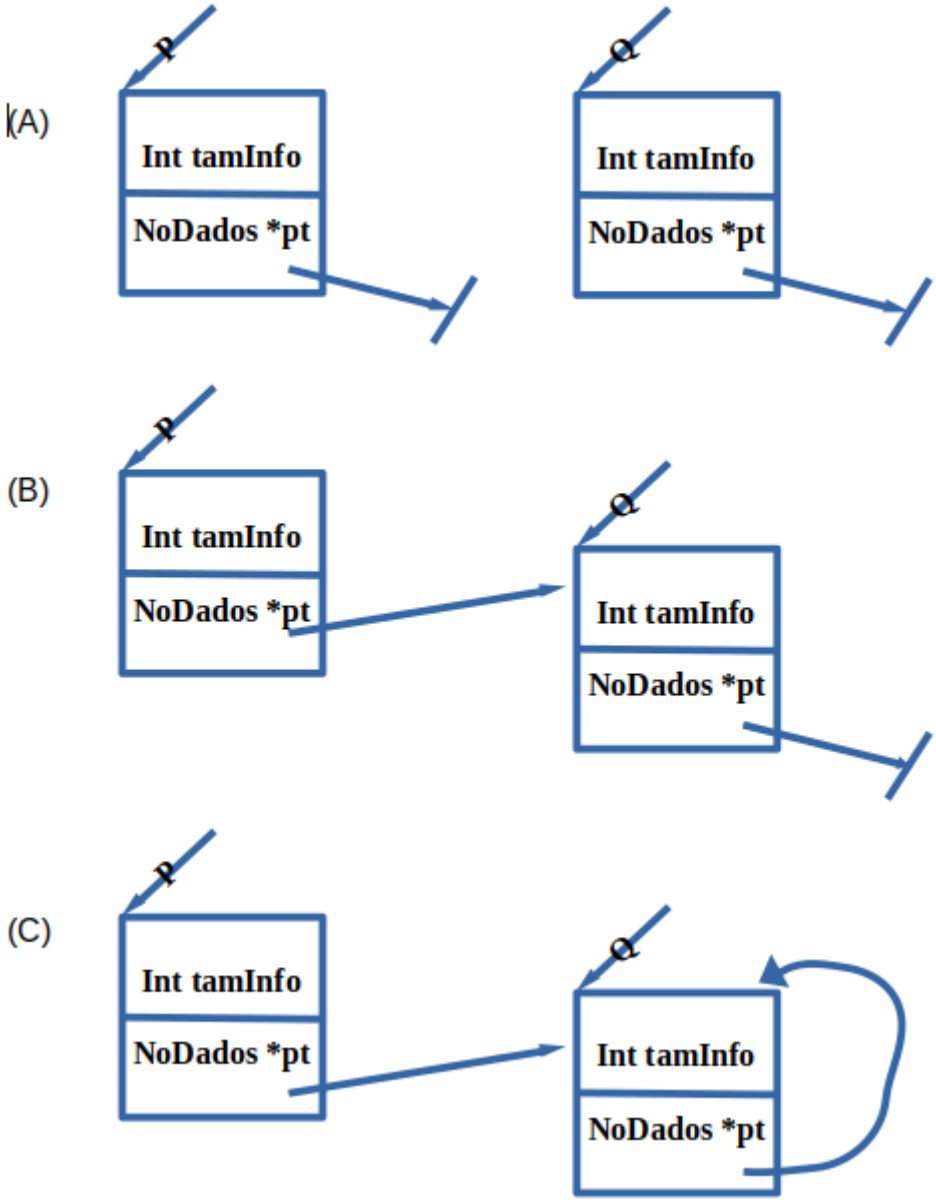


Figura 1: Inserção de um elemento em uma sequência simplesmente encadeada.

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 abaixo.

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

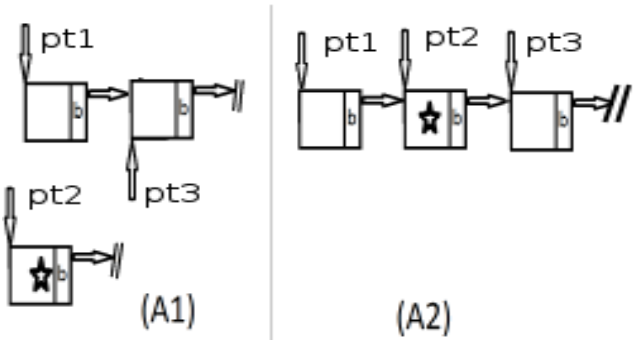


Figura 2: Sequência simplesmente encadeada.

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

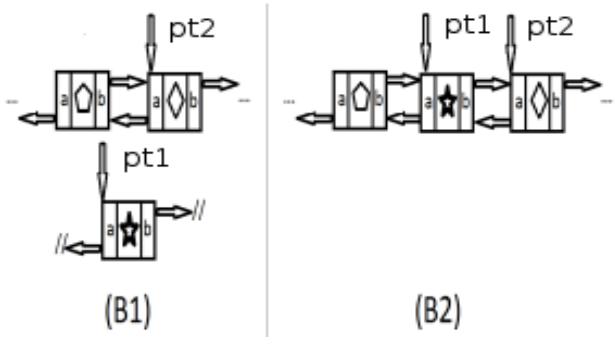


Figura 3: Sequência duplamente encadeada.

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

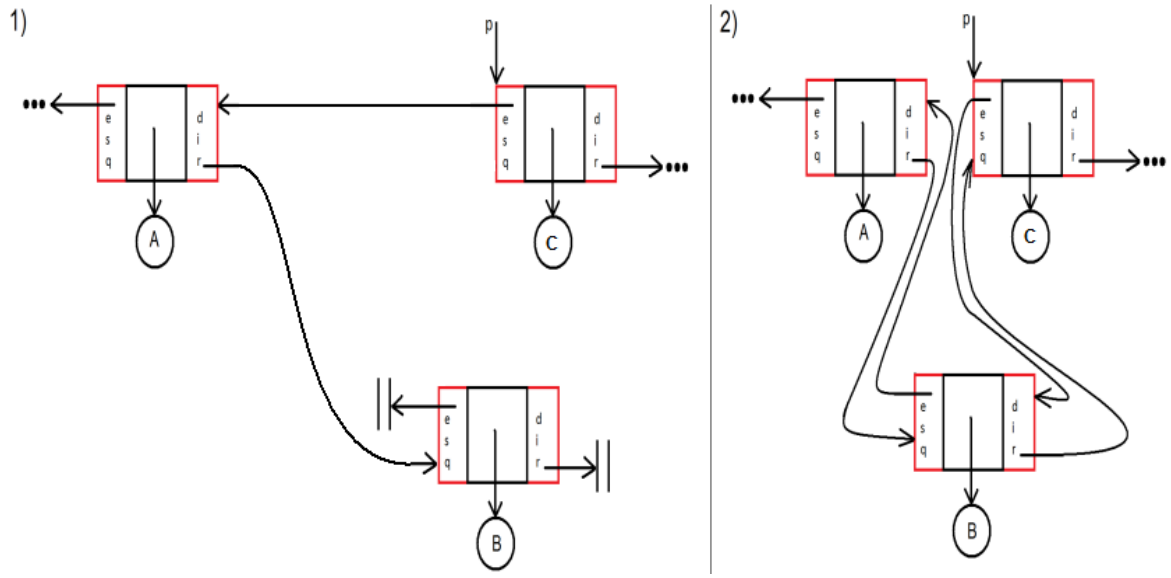


Figura 4: Inserção de elemento duplamente encadeado.

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 as sequências encadeadas exibidas abaixo:

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

A)

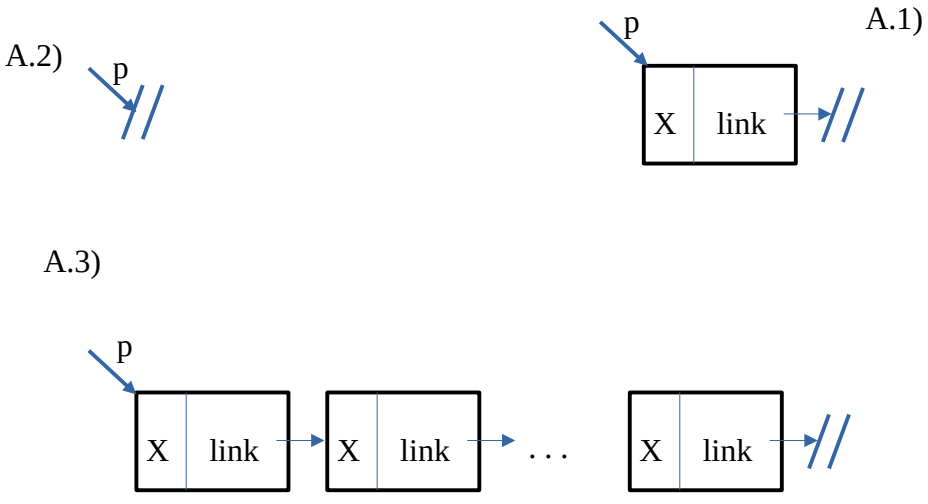
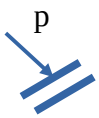


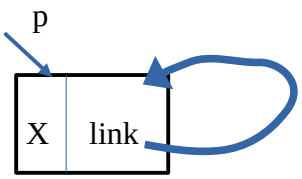
Figura 5: A1) Sequência vazia, A2) Sequência com uma unidade e A3) Sequência simplesmente encadeada com 'n' unidades, finalizada com o Null.

B)

B1)



B2)



B3)

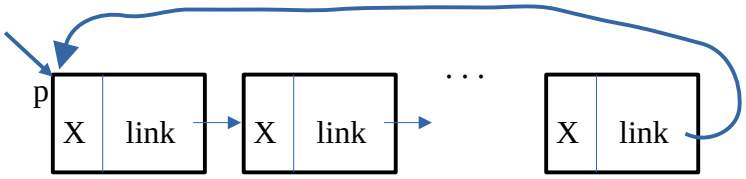
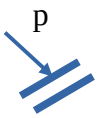


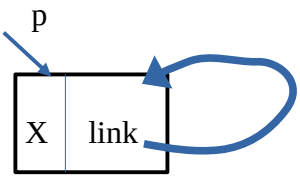
Figura 6: Sequência encadeada circular: B1) Sequência vazia, B2) Sequência com uma unidade e B3) Sequência com 'n' unidades.

C)

C1)



C2)



C3)

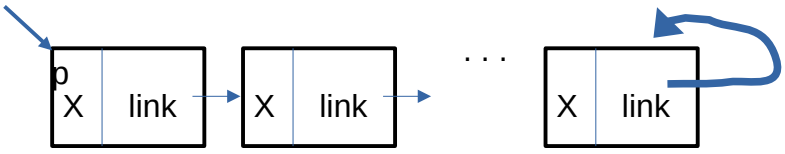
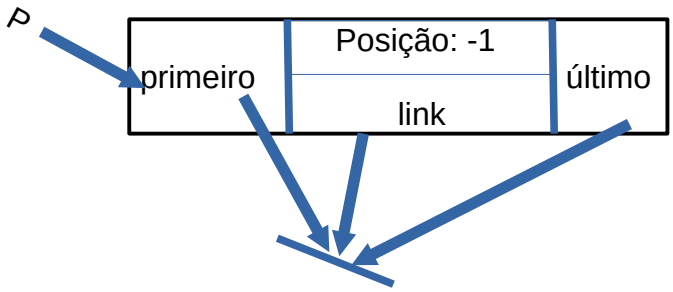


Figura 7: Sequência encadeada: C1) Sequência vazia, C2) Sequência com uma unidade e C3) Sequência com 'n' unidades.

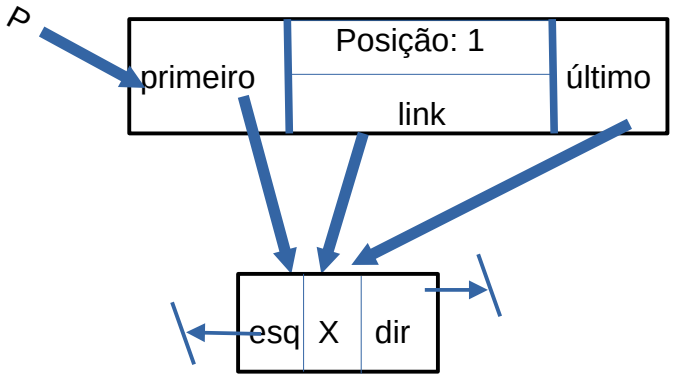
D) Neste caso o protótipo da função será *int contaNodo(struct descritor *p)*, onde:

```
struct descritor{ unsigned  int posicao;
                  struct nodo *primeiro, *ultimo,*link;
        };
```

C1)



C2)



C3)

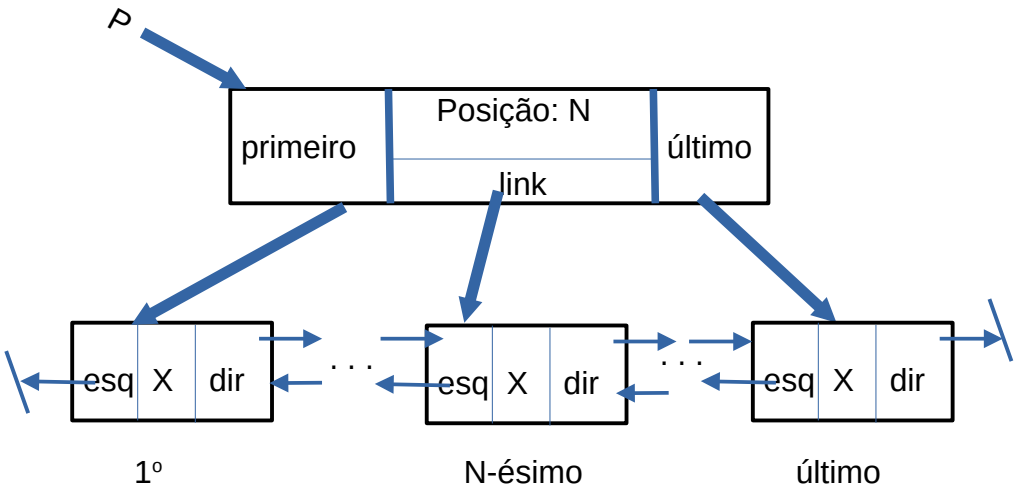


Figura 8: Sequência encadeada com "líbero": C1) Sequência vazia, C2) Sequência com uma unidade e C3) Sequência com 'k' unidades.

7. Um descritor é uma “struct” com metadados sobre a estrutura de dados propriamente dita (uma lista encadeada, por exemplo). Um exemplo de descritor pode ser visto na Figura 9 contendo os campos: *tamanho* (quantidade atual de elementos na lista) e *inicio* (apontador para o nó inicial da lista). Para uma lista simplesmente encadeada com descritor, construa as funções:

   a) *int reinicia(descritor *p)*, a qual promove um “reset” da lista removendo todos os nós, retornando a lista ao estado “vazia” (*inicio* == *Null*). Sugestão: utilize ponteiro auxiliar como um “farejador” de final de lista dentro de um laço;

   b) *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 9). Um requisito adicional é que a posição *pos* já exista na sequência de nós de dados da lista.

As funções devem retornar zero ou um (1) a depender da respectiva operação falhar ou ter sucesso.

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

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

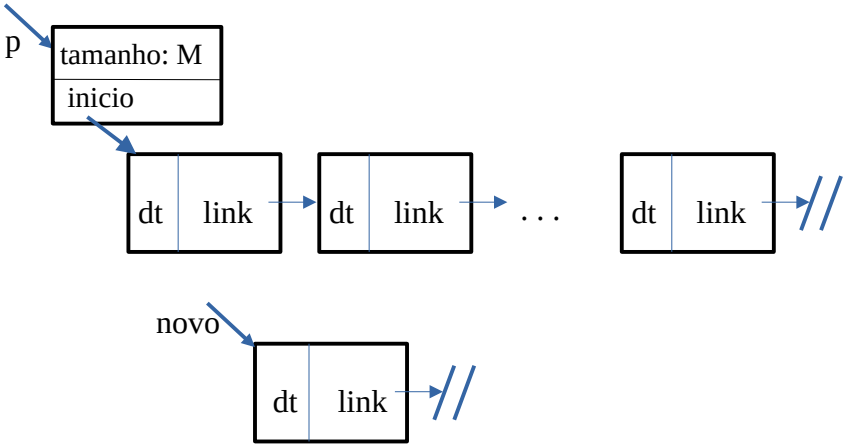


Figura 9: Sequência simplesmente encadeada com descritor.

8. Construa a função completa *int contaNodo(struct descritor *p)* a qual retorna a contagem do número de nós (caixas brancas, na figura) atualmente inseridos em um conjunto de listas, conforme a Figura 10.

   Um exemplo pode ser visto na figura abaixo. Lembre-se que uma ou mais listas podem eventualmente estar vazias (por exemplo: p→vet[2]==Null).

   Perceba que:
   - *p->vet[0]* é um apontador que faz referência ao início da lista-zero,
   - *p->vet[1]* ao início da lista-1 e assim sucessivamente até...
   - *p ->vet[p->Tamvet-1]* que faz referência à última lista.

   Execute teste de mesa para conferir sua solução.

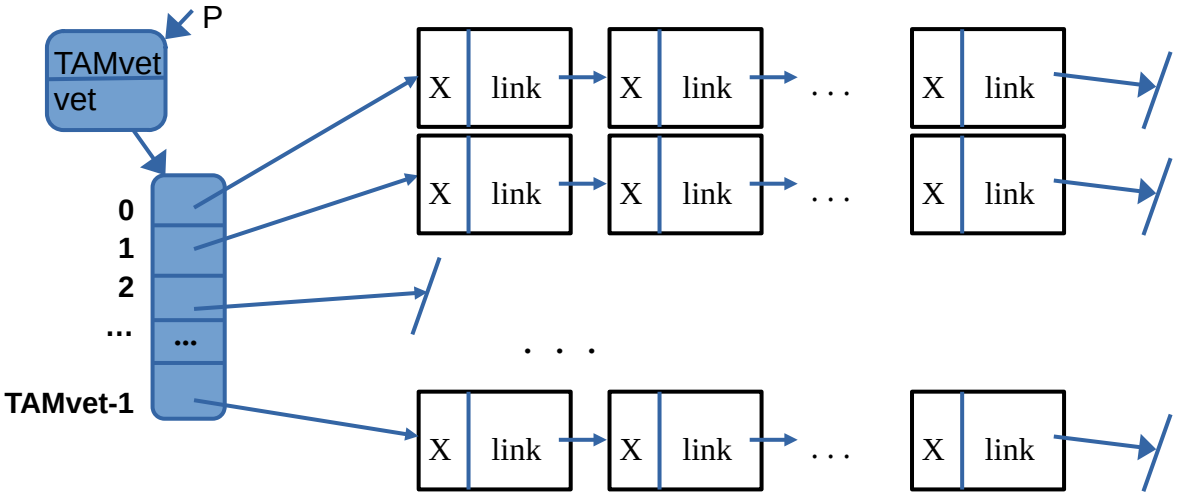
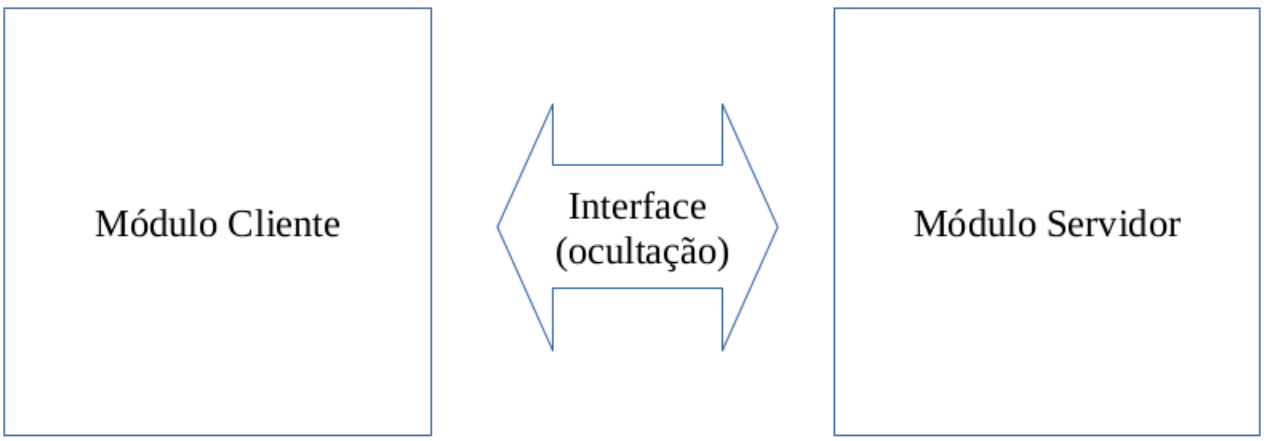


Figura 10: Exemplo de uma lista de listas.

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



10. Sobre encapsulamento e construção de interfaces, escreva em C (em um arquivo "interface.h") a especificação de uma interface para a estrutura proposta na questão 8.

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