Nome: _____ Número USP: _____

Para as questões 1, 2, 4 e 6 cada valor preenchido incorretamente anulará um valor preenchido corretamente. Sugere-se deixar os campos que você tenha dúvida em branco.

Questão 1 - Dado o seguinte programa:

```
#include <stdlib.h>
#include <stdio.h>

typedef struct aux {
    int chave;
    struct aux* prox;
} REG, * PONT;

void trocar1(REG r1, REG r2){
    int temp = r1.chave;
    r1.chave = r2.chave;
    r2.chave = temp;
}

void trocar2(PONT r1, PONT r2)
{
    int temp = r1->chave;
    r1->chave = r2->chave;
    r2->chave = temp;
}

int main(){
    REG r1;
    REG r2;
    REG r3;
    REG r4;
    REG r5;

    r1.chave = 5;
    r2.chave = 4;
    r3.chave = 3;
    r4.chave = 2;
    r5.chave = 1;

    r1.prox = &r2;
    r2.prox = &r3;
    r3.prox = &r4;
    r4.prox = &r1;

    trocar1(r1,r2);
    trocar2(&r3,&r4);

    printf("r1.chave: %i\n",r1.chave);
    printf("r2.chave: %i\n",r2.chave);
    printf("r3.chave: %i\n",r3.chave);
    printf("r4.chave: %i\n",r4.chave);

    printf("&r1: %p\n",&r1);
    printf("r1.prox: %p\n",r1.prox);

    printf("r2.prox->chave: %i\n",r2.prox->chave);
    printf("r4.prox->chave: %i\n",r4.prox->chave);

    r5.prox = &r5;
    printf("r5.prox->chave: %i\n",r5.prox->chave);
    printf("(*(r5.prox)).chave: %i\n",
            (*(r5.prox)).chave);

    return 0;
}
```

Sabendo-se que (onde &x significa o endereço no qual a variável x foi criada):

&r1: 220 &r3: 240 &r5: 260
 &r2: 230 &r4: 250

Preencha as lacunas abaixo com o que seria impresso pela execução do programa:

r1.chave: _____

r1.prox: _____

r2.chave: _____

r2.prox->chave: _____

r3.chave: _____

r4.prox->chave: _____

r4.chave: _____

r5.prox->chave: _____

&r1: _____

(*(r5.prox)).chave: _____

Questão 2 - Dado o seguinte programa:

```
#include <malloc.h>
#include <stdio.h>

typedef int* pontInt;

int funcao(int a, int* b){
    int resultado = *b + a;
    *b = a;
    return resultado;
}

int main(){
    int a = 1;
    int b = 2;
    int c = 4;
    int d = 8;
    int e = 16;
    pontInt x;
    pontInt y;
    pontInt* z;
```

```
    a = funcao(b, &c);
    x = &b;
    y = &d;
    z = &y;
    x = &a;
    e = d + (*y);
    **z = 7;

    printf("a: %i\n", a);
    printf("b: %i\n", b);
    printf("c: %i\n", c);
    printf("d: %i\n", d);
    printf("e: %i\n", e);
    printf("x: %p\n", x);
    printf("y: %p\n", y);
    printf("z: %p\n", z);
    printf("*x: %i\n", *x);
    printf("*y: %i\n", *y);
    printf("*z: %p\n", *z);
    printf("**z: %i\n", **z);
    return 0;
}
```

Sabendo-se que (onde $\&x$ significa o endereço no qual a variável x foi criada):

&a: 756	&d: 768	&y: 780
&b: 760	&e: 772	&z: 784
&c: 764	&x: 776	

Preencha as lacunas abaixo com o que seria impresso pela execução do programa:

a: _____	e: _____	*x: _____
b: _____	x: _____	*y: _____
c: _____	y: _____	*z: _____
d: _____	z: _____	**z: _____

Questão 3 - Considere uma nova estrutura de dados que chamaremos de LISTAY e trata-se de uma lista duplamente ligada, circular, com nó-cabeça e ordenada de forma crescente. A estrutura possuirá um nó-cabeça, criado na inicialização, o qual encabeçará a lista (isto é, ficará sempre antes do primeiro elemento "válido" da lista) e nunca será apagado. A inserção de elementos deve ser feita de acordo com o valor da chave de seus registros (de forma crescente) e não será permitida a inserção de elementos com chaves repetidas. A estrutura LISTAY, propriamente dita, terá um único campo: *cabeça* que receberá o endereço do nó-cabeça. Lembre-se: a estrutura é circular e duplamente ligada (cada elemento possui um ponteiro para o endereço de seu anterior e de seu próximo, o último elemento terá o nó-cabeça como próximo e o nó-cabeça terá o último elemento da lista como seu anterior). Complete as duas funções abaixo (o código a ser completado está na próxima página):

PONT buscaSeq(LISTAY* l, TIPOCHAVE ch), função que recebe o endereço de uma LISTAY e o valor de uma chave e deverá retornar *NULL* caso nenhum elemento válido (isto é, um elemento exceto o nó-cabeça) possua um registro com essa chave. Caso contrário, deverá retornar o endereço do elemento que possui em seu registro o valor do campo *chave* igual a *ch*. A busca realizada deve ser uma busca sequencial considerando que a lista está ordenada de forma crescente pelo valor das chaves e utilizando o nó-cabeça como sentinela.

bool excluirElemLista(LISTAY* l, TIPOCHAVE ch, REGISTRO* reg), que deve retornar *false* caso não exista nenhum elemento válido na lista que possua um registro com chave igual ao valor da chave passado como parâmetro (*ch*). Caso contrário, deverá copiar o registro do elemento a ser excluído (que possui chave igual a *ch*) para a memória apontada pelo parâmetro *reg*, acertar todos os ponteiros (e qualquer outro campo) necessários, liberar a memória do elemento que está sendo apagado e retornar *true*.

As seguintes definições e funções já foram dadas:

```
#include <stdio.h>
#include <malloc.h>
#define true 1
#define false 0

typedef int bool;
typedef int TIPOCHAVE;

typedef struct {
    TIPOCHAVE chave;
    // outros campos...
} REGISTRO;

typedef struct auxElem {
    REGISTRO reg;
    struct auxElem* ant;
    struct auxElem* prox;
} ELEMENTO;

typedef ELEMENTO* PONT;

typedef struct {
    PONT cabeca;
} LISTAY;
```

```
void inicializarLista(LISTAY* l){
    l->cabeca = (PONT) malloc(sizeof(ELEMENTO));
    l->cabeca->prox = l->cabeca;
    l->cabeca->ant = l->cabeca;
}

int tamanho(LISTAY* l) {
    PONT end = l->cabeca->prox;
    int tam = 0;
    while (end != l->cabeca){
        tam++;
        end = end->prox;
    }
    return tam;
}
```

O código a ser completado está a seguir (se desejar, ao invés de completar, você pode fazer uma nova implementação, respeitando a assinatura da função). Caso ache que há linhas desnecessárias no código, basta riscá-las:

```
PONT buscaSeq(LISTAY* l, TIPOCHAVE ch){
    PONT atual = l->cabeca->prox;
    _____ = ch;
while (atual->reg.chave < _____) atual = _____;
if (atual != _____ && atual->reg.chave == ch) return _____;
return NULL;
}
```

```

bool excluirElemLista(LISTAY* l, TIPOCHAVE ch, REGISTRO* reg){
    PONT apagar = buscaSeq(l, ch);
    if (apagar == NULL) return _____;
    _____ = apagar->reg;
    apagar->prox->ant = _____;
    apagar->ant->prox = _____;
    free(_____);
    return _____;
}

```

Questão 4 - Todas as funções da questão anterior recebiam como parâmetro um ponteiro para uma LISTAY. Assinale com um V (verdadeiro) as funções que poderiam ser reescritas (adaptando o código da função, mas mantendo sua funcionalidade) da seguinte forma: substituindo o ponteiro para LISTAY (LISTAY* l) por uma variável do tipo LISTAY (LISTAY l); caso contrário assinale com F (falso: a função não poderia ser reescrita). Caso não saiba, sugere-se deixar em branco, pois cada alternativa assinalada incorretamente descontará uma assinalada corretamente.

Dica: preste atenção em quais são os campos que compõe a estrutura LISTAY propriamente dita, adicionalmente, lembre-se que o nó-cabeça é criado pela função inicializarLista e nunca é apagado.

```

[ ] int tamanho(LISTAY l)
[ ] void inicializarLista(LISTAY l)
[ ] PONT buscaSeq(LISTAY l, TIPOCHAVE ch)
[ ] bool excluirElemLista(LISTAY l, TIPOCHAVE ch, REGISTRO* reg)

```

Questão 5 - Dada uma **fila**, implementação estática, que use as estruturas definidas a seguir e onde definimos que a fila está vazia quando numElementos = 0, escreva o código da função de inserção conforme visto em aula e descrito a seguir:

bool inserirElementoFila(FILA* f, REGISTRO reg), que insere o REGISTRO reg (isto é, copia o conteúdo do registro) no final da fila (após o último elemento válido) caso haja espaço e retorna *true*; ou retorna *false* se a fila estiver lotada. **Não esqueça de acertar todos os campos/índices necessários.**

```

#include <stdio.h>
#define true 1
#define false 0
#define MAX 50
typedef int bool;
typedef int TIPOCHAVE;

typedef struct {
    TIPOCHAVE chave;
} REGISTRO;

typedef struct {
    int inicio;
    int numElementos;
    REGISTRO A[MAX];
} FILA;

void inicializarFila(FILA *f){
    f->inicio=0;
    f->numElementos=0;
}

```

Questão 6 – O código a seguir apresenta a função `AtribuiMatriz`, da implementação de matrizes esparsas vista ao longo do semestre.

Questão 6a – Complete a parte do código responsável por procurar pelo elemento da linha *lin* e coluna *col* (parte **A** do código)

<pre>bool AtribuiMatriz(MATRIZ* m,int lin, int col, float val) { if (lin<0 lin >= m->linhas col<0 col >= m->colunas) return false; PONT ant = _____; PONT atual = _____; while (_____ != NULL && atual->coluna _____ col) { ant = _____; atual = _____; } }</pre>	A
<pre>if (atual != NULL && atual->coluna == col) { if (val == 0) { if (ant==NULL) m->A[lin] = atual->prox; else ant->prox = atual->prox; free(atual); } }</pre>	B
<pre>else atual->valor = val;</pre>	C
<pre>}else if (val != 0){ PONT novo = (PONT) malloc(sizeof(NO)); novo->coluna = col; novo->valor = val; novo->prox = atual; if (ant == NULL) m->A[lin] = novo; else ant->prox = novo; }</pre>	D
<pre>return true; }</pre>	

Questão 6b – Assinale V para verdadeiro e F para falso em relação a o que é feito por esse código. Cada alternativa assinalada incorretamente anulará uma assinalada corretamente. Na dúvida, sugere-se deixar a alternativa em branco.

- O trecho **B** do código é responsável por excluir da estrutura o elemento correspondente a linha *lin* e coluna *col*.
- O trecho **C** do código é responsável por criar um novo elemento.
- O trecho **D** do código é responsável por inserir o elemento correspondente a linha *lin* e coluna *col*.
- Se o usuário solicitar para inserir o valor 0 (zero) para um elemento que ainda não existe na estrutura, o código irá retornar *true*, sem modificar a estrutura.

Questão 7 – O código a seguir apresenta parte do código da Lista Ligada Ordenada, Circular e com Nó Cabeça implementada ao longo do semestre.

```
typedef struct tempRegistro{
    REGISTRO reg;
    struct tempRegistro* prox;
} ELEMENTO;

typedef ELEMENTO* PONT;

typedef struct {
    PONT cabeca;
} LISTA;

void inicializarLista(LISTA* l){
    l->cabeca = (PONT) malloc(sizeof(ELEMENTO));
    l->cabeca->prox = l->cabeca; // faz com que a lista seja circular
}

int tamanho(LISTA* l) {
    PONT end = l->cabeca->prox;
    int tam = 0;
    while (end != l->cabeca){
        tam++;
        end = end->prox;
    }
    return tam;
}

PONT buscaSentinela(LISTA* l, TIPOCHAVE ch){
    PONT pos = l->cabeca->prox;
    l->cabeca->reg.chave = ch;
    while (pos->reg.chave != ch) pos = pos->prox;
    if (pos != l->cabeca) return pos;
    return NULL;
}

PONT buscaSeqExc(LISTA* l, TIPOCHAVE ch, PONT* ant){
    *ant = l->cabeca;
    PONT pos = l->cabeca->prox;
    l->cabeca->reg.chave = ch; // utiliza o cabeça como sentinela
    while (pos->reg.chave < ch){
        *ant = pos;
        pos = pos->prox;
    }
    if ((pos != l->cabeca) && (pos->reg.chave == ch)) return pos;
    return NULL;
}

bool excluirElemLista(LISTA* l, TIPOCHAVE ch){
    PONT ant, i;
    i = buscaSeqExc(l, ch, &ant);
    if (i == NULL) return false;
    ant->prox = i->prox;
    free(i);
    return true;
}

bool inserirElemListaOrd(LISTA* l, REGISTRO reg) {
    PONT ant, i;
    i = buscaSeqExc(l, reg.chave, &ant);
    if (i != NULL) return false;
    i = (PONT) malloc(sizeof(ELEMENTO));
    i->reg = reg;
    i->prox = ant->prox;
    ant->prox = i;
    return true;
}
```

Questão 7a – Reescreva o código das funções *inicializarLista*, *tamanho*, *excluirElemLista* e *inserirElemListaOrd* considerando que a estrutura foi modificada para conter um campo chamado *numElem* que armazena o número de elementos válidos (sem contar o nó cabeça) da estrutura.

```
typedef struct {  
    PONT cabeca;  
    int numElem;  
} LISTA;
```

Questão 7b – Todas as funções originais dessa estrutura podiam receber como parâmetro uma cópia da LISTA (ao invés de uma referência da LISTA) e com as devidas adaptações dos códigos teriam sua funcionalidade mantida, exceto a função *inicializarLista* que de fato precisa receber uma referência como parâmetro. Assinale com V (para verdadeiro) e F (para falso) quais funções poderiam receber como parâmetro uma cópia da LISTA e manter sua funcionalidade após a modificação da estrutura (isto é, após criarmos o campo *numElem*). Cada alternativa assinalada incorretamente anulará uma assinalada corretamente. Na dúvida, sugere-se deixar a alternativa em branco.

- [] *inicializarLista*
- [] *tamanho*
- [] *excluirElemLista*
- [] *inserirElemListaOrd*