

# ACH2023 - Algoritmos e Estrutura de Dados

## Prova 1 – 2018 – Turma 94

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

Questão 1 (1,5) Dado o seguinte programa:

```
#include <malloc.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      &r2: 230      &r3: 240      &r4: 250      &r5: 260

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

r1.chave: \_\_\_\_\_  
r2.chave: \_\_\_\_\_  
r3.chave: \_\_\_\_\_  
r4.chave: \_\_\_\_\_  
&r1: \_\_\_\_\_

r1.prox: \_\_\_\_\_  
r2.prox → chave: \_\_\_\_\_  
r4.prox → chave: \_\_\_\_\_  
r5.prox → chave: \_\_\_\_\_  
(\*r5.prox).chave: \_\_\_\_\_

Questão 2 (1,5) 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      &b: 760      &c: 764      &d: 768      &e: 772      &x: 776      &y: 78

&z: 784

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 (4.0) 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 cabeça;
} 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 (1,0)** 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\* I) por uma variável do tipo LISTAY( LISTAY I); caso contrário assinale com F (falso: a função não pode 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 I)
- [        ] void inicializarLista(LISTAY I)
- [        ] PONT buscaSeq(LISTAY I, TIPOCHAVE ch)
- [        ] bool excluirElemLista(LISTAY I, TIPOCHAVE ch, REGISTRO\* reg)

**Questão 5 (2,0)** 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;
}
```