PF – LP1 - 2013-2
Prof. Leandro Marzulo

Nome:

Instruções: Esta prova é composta de duas questões totalizando 10 (dez) pontos, sendo a nota máxima 10 (dez). Responda as questões de forma sucinta e clara e usando indentação para as questões que envolvem programação. As questões devem respondidas a caneta. BOA PROVA!

Questão 1) (2,0) **Alocação dinâmica**: Explique as diferenças e semelhanças entre arrays unidimensionais estáticos e dinâmicos. Não esqueça de abordar questões de declaração, acesso aos elementos e a forma como os elementos são armazenados em memória.

A alocação de arrays estáticos é preparada automaticamente em tempo de compilação, enquanto que para arrays dinâmicos é feita em tempo de execução. Os arrays unidimensionais tanto estáticos quanto dinâmicos ocupam uma área contígua na memória. Entretanto, no caso dos arrays dinâmicos temos uma variável do tipo ponteiro que aponta para essa região e esse ponteiro é declarado explicitamente. O acesso aos elementos de ambos os tipos de array podem ser feitos usando []. No caso de arrays dinâmicos, podemos também acessar os elementos usando aritmética de ponteiros.

Questão 2) (2,0) **Escopo de variáveis e ponteiros**: O que é impresso na tela depois da execução do programa abaixo?

```
#include <stdio.h>
void func(int, int *);
int main()
{
        int i; int k=0;
        for (i=0; i<5;i++,k+=2)
                func(i,&k);
}
void func(int z, int * w)
{
        static int x=10;
        int y = 5;
        printf("%d - %d - %d - %d\n",x,y,z,*w);
        x++; y++; z++; (*w)++;
}
```

```
10 - 5 - 0 - 0
11 - 5 - 1 - 3
12 - 5 - 2 - 6
13 - 5 - 3 - 9
14 - 5 - 4 - 12
```

Questão 3) (6,0) **Lista encadeada**: Considere uma lista encadeada simples contendo números de ponto flutuante de dupla precisão. A definição de um elemento da lista segue abaixo:

```
struct elem
{
        double num;
        struct elem * prox;
};
```

Responda:

a) (2,0) Escreva o código completo (incluindo a declaração) da função insereFim, que recebe como parâmetros um ponteiro para o elemento a ser inserido e um ponteiro para o ponteiro da cabeça da lista e insere o elemento no final da lista. A função não é recursiva. O cabeçalho da função é:

void insereFim(struct elem * e, struct elem ** p);

```
void insereFim (struct elem * e, struct elem ** p)
{
    // como e vai ser o último da lista, sabemos que ele aponta para NULL
    e->prox = NULL;
    // se a lista está vazia, substitui a cabeça
    if (!(*p))
        *p = e;
    else
    {
        struct elem * aux = *p; // auxiliar para percorrer a lista
        while(aux->prox) //enquanto existir um próximo elemento
            aux = aux->prox; // avança
        aux->prox = e; //próximo do último = e
    }
}
```

b) (2,0) Escreva o código completo (incluindo a declaração) da função buscaQtdFaixa que recebe dois inteiros A e B e o ponteiro para a cabeça da lista e retorna o número de elementos E da lista tal que A<E<B. Você deve usar recursão. O cabeçalho da função é:

int buscaQtdFaixa(int a, int b, struct elem * p);

```
int buscaQtdFaixa(int a, int b, struct elem * p)
{
    //se o elemento existe
    if (p)
    {
        //se o elemento corrente está dentro da faixa:
        if ((p->num > a) && (p->num <b))
            return 1+buscaQtdFaixa(a, b, p->prox); // soma 1 com o que
encontrar na sublista restante;
        else //c.c.
            return buscaQtdFaixa(a, b, p->prox); // apenas retorna a sublista
restante;
    }
    else
        return 0;

}
```

c) (2,0) Escreva o código completo (incluindo a declaração) da função buscaFaixa que recebe dois inteiros A e B, o ponteiro para a cabeça da lista e um inteiro MAX e retorna um vetor V com MAX elementos. V é um vetor de ponteiros para a struct elem e deve ser alocado dinamicamente dentro da função buscaFaixa. Para cada E da lista tal que A<E<B, E deve ser incluído em V. A função retorna V quando a lista chega ao fim ou quando MAX elementos foram encontrados. Você não deve usar recursão. O cabeçalho da função é:

struct elem ** buscaFaixa(int a, int b, struct elem * p, int max);

```
struct elem ** buscaFaixa(int a, int b, struct elem * p, int max)
{
    //aloca o vetor resultado (todo zerado)
    struct elem ** v = (struct elem **) calloc(max, sizeof(struct elem *));

    int i=0;

    while ((p) && (i<max)) //enquanto não cheguei ao final da lista ou no
máximo de resultados
    {
        //se o elemento corrente está dentro da faixa:
        if ((p->num > a) && (p->num <b))
        {
            v[i]=p; //insere no vetor de resultados
            i++; //ajusta i
        }
        //avança para o próximo da lista
        p = p->prox;
    }
    return v;
}
```