P2 – LP1 - 2013-2
Prof. Leandro Marzulo
GABARITO

Nome:

Instruções: Esta prova é composta de três questões totalizando 12 (doze) pontos, sendo a nota máxima 10 (dez). Responda as questões de forma sucinta e clara e usando indentação para as questões que envolvem programação. O uso de lápis é permitido, entretanto, pedidos de revisão serão considerados apenas para questões respondidas a caneta. BOA PROVA!

Questão 1) (2,0) Uma variável do tipo ponteiro armazena um endereço de memória. Quando declaramos um ponteiro usamos um tipo base acompanhando do caractere * (`char * c`, `struct teste * m`, entre outros). Para que serve então esse tipo base, se todo endereço tem o mesmo tamanho?

Resp.: O tipo base na declaração de um ponteiro serve para que o compilador saiba tamanho dos elementos referenciados pelo ponteiro. Essa informação é usada para calcular os deslocamentos em operações aritméticas com ponteiros. Se temos, por exemplo, a declaração **`int * v`** e fazemos um malloc para 10 inteiros, armazenando esse ponteiro em **v**, ao acessar **v[2]**, considerando que um inteiro ocupa 4 bytes, o compilador irá somar 8 ao base de **v** para obter o endereço de **v[2]**.

Questão 2) (2,0) Escreva o código em C da função somaVetorEscalar, que recebe um vetor de inteiros V, o tamanho do vetor e um escalar E e executa a operação V+E. Você deverá usar aritmética de ponteiros para acessar os elementos de V.

Resp.:
```
void somaVetorEscalar(int * v, int tam, int e)
{
     int i;
     for (i=0; i<tam; i++)
          (*(v+i))+=e;
}
```

Questão 3) (8,0) Considere um projeto de uma aplicação que leia uma quantidade indefinida de números inteiros positivos e imprima, ao final, a lista dos números lidos em ordem crescente. O programa para de ler novos números quando o usuário digita um número <= 0. Cada número lido deve ser inserido em uma lista encadeada simples e ordenada.

Este projeto de software está dividido em 3 arquivos:

- principal.c – arquivo que contém a implementação da função main. Esta função implementa o loop de leitura dos números. Dentro do loop devem ser alocados dinamicamente os elementos da lista e deve ser chamada a função insereOrdenada. Depois do término do loop deve ser chamada a função imprimeLista.
- auxiliar.c – arquivo que contém a implementação das funções insereOrdenada, que insere um elemento na lista em ordem crescente, e imprimeLista, que imprime a lista ordenada. Estas duas funções são recursivas, como no exemplo visto em sala de aula.
- auxiliar.h – arquivo que contém o protótipo das funções insereOrdenada e imprimeLista, além da declaração da struct de um elemento da lista.

a) (3,0) Escreva o código contido no arquivo principal.c

Resp.:

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

int main(void)
{
        struct elem * phead = NULL;
        struct elem * e;
        int tmp;
        do
        {
                printf("Digite um número inteiro: ");
                scanf("%d", &tmp);
                getchar(); // para consumir a quebra de linha
                if (tmp > 0)
                {
                        e = (struct elem *)malloc(sizeof(struct elem));
                        e->valor = tmp;
                        insereOrdenada(e, &phead);
                }
        } while (tmp > 0);

        imprimeLista(phead);

        // Não precisava disso, mas estou incluindo para saberem como se faz o
free na lista
        limpaLista(&phead);

        return 0;
}
```

b) (3,0) Escreva o código contido no arquivo auxiliar.c

Resp.:

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

// só inclui isso por causa do limpaLista
#include <stdlib.h>

/* Criei essa função pro código ficar mais legível.
Não é obrigatório usar essa solução.
Você substituir as chamadas dessa função pelo seu respectivo código*/
void insereInicio(struct elem * e, struct elem ** l)
{
        e->next = *l;
        *l=e;
}

void insereOrdenada(struct elem * e, struct elem ** l)
{
        if (*l) //se tem coisa na lista
        {
                //se o elemento é menor que a cabeça
                if (e->valor < (*l)->valor)
                        insereInicio(e,l);
                else
                        insereOrdenada(e, &((*l)->next));
        }
        else
                insereInicio(e,l);
}

void imprimeLista(struct elem * l)
{
        if (l) //se a lista não acabou
        {
                //imprime o primeiro elemento
                printf("%d\n",l->valor);
                //imprime a sublista que começa no elemento seguinte
                imprimeLista(l->next);
        }
        else
                printf("Fim da Lista!\n");
}

// Não precisava disso, mas estou incluindo para saberem como se faz o free na
lista
void limpaLista(struct elem ** l)
{
        if (*l)
        {
                limpaLista(&((*l)->next));
                free (*l);
        }
        else
                printf("Lista Limpa!\n");
}
```

c) (2,0) Escreva o código contido no arquivo auxiliar.h

Resp.:

```
struct elem
{
        int valor;
        struct elem * next;
};

void insereOrdenada(struct elem *, struct elem **);
void imprimeLista(struct elem *);

// Não pedi essa função, mas estou incluindo para que saibam como fazer o free
na lista
void limpaLista(struct elem **);
```

DICA: Não esqueça de incluir os headers das bibliotecas usadas em cada arquivo.