Universidade Federal Fluminense
Campus de Rio das Ostras
Instituto de Ciência e Tecnologia

Programação de Computadores II
Prova P2 - 28/06/2018
Prof. Alessandro Copetti

Nome: _____**GABARITO**_____

Obs.: A prova pode ser feita a lápis! As funções não devem conter valores fixos.

1. (1pt) O que será impresso na tela? Mostre o valor que cada variável assume durante a execução e também circule a resposta; não é para explicar o que cada linha faz. Toda questão deve estar correta.

```
int *funcao(int *v, int n, int *x) {
    int i=0, cont=0,j=0;
    int *vet2=(int *) malloc(n*sizeof(int));
    for (j=n-1,i=0;i<n;i++) {
        if ((v[i]%2)!=0)
            cont++;
        vet2[j--]=v[i];
    }
    *x = cont;
    return vet2;
}

main(){
    int vet[5]={1,2,3,4,5};
    int *p, i, cont;
    p = funcao(vet, 5, &cont);
    for (i=0;i<5;i++)
        printf("%d ",p[i]);
    printf("\nResultado=%d", cont);
}
```

5 4 3 2 1
3

2. (2pts) Desenvolva uma função com o protótipo: `Lista* lst_crescente (int valorMax);` A função deve retornar uma lista encadeada, a qual fica inserindo novos elementos na lista somente se o valor gerado aleatoriamente entre 1 e valorMax é maior que o maior valor inserido até o momento. Quando for gerado o número valorMax então a função deve retornar o endereço do primeiro elemento da lista. Por exemplo, a execução para `lst_crescente(1000)` poderia dar como resposta após imprimir a lista: `996, 963, 725, 501, 468, 42`

```
typedef struct lista {
    int info;
    struct lista *prox;
} Lista;
Lista* lst_insere (Lista* lst, int val) {
    Lista* novo = (Lista*) malloc(sizeof(Lista));
    novo->info = val;
    novo->prox = lst;
    return novo;
}
int main() {
    Lista *lst=NULL;
    lst = lst_crescente(1000);
    // lst_imprime(lst);
    return 0;
}
```

```
Lista* lst_crescente (int valorMax)
{
   int num, maior = 0;
   Lista *lst = lst_cria();
   while(1)  {
      num = (rand() % valorMax) + 1;
```

```
      if (num == valorMax)
         break;
      if (num > maior) {
         maior = num;
         lst = lst_insere(lst, num);
      }
   }
   return lst;
}
```

3. Tendo o código abaixo:

```
typedef struct data {
   int dd, mm, aa;         /*  Dia, mes e ano              */
} Data;

typedef struct compromisso {
   char descricao[81];     /*  Descricao do compromisso    */
   Data dta;               /*  Data do compromisso         */
} Compromisso;

typedef struct elemento {
   Compromisso compr;          /*  Compromisso armazenado             */
   struct elemento *prox;    /*  Ponteiro para o próximo elemento     */
} Elemento;

int busca(Elemento* lst, int mes);
Elemento* lista_insere (Elemento* lst, char *descricao, int dia, int mes, int ano);
void gravaLista(Elemento *lista, char *nomeArq);
Elemento *leDoArquivo(Elemento *lista, char *nomeArq);
Compromisso ** vetorPtr(Elemento *lista, int *numCompromissos);
void imprimeVetPtr(Compromisso **vet, int n);

main(void) {
   Elemento *lista=NULL;
   Compromisso **vet;
   int numCompromissos=0;
   lista = listaInsere(lista, "compromisso um",25,11,10);
   lista = listaInsere(lista, "compromisso dois",30,11,10);
   lista = listaInsere(lista, "compromisso tres",01,12,10);
   printf("\nNumero de compromissos no mes 11=%d\n",busca(lista,11));
   gravaLista(lista, "compromissos.bin");
   lista = leDoArquivo(lista, "compromissos.bin");
   vet = vetorPtr(lista, &numCompromissos);
   imprime(lista);
   imprimeVetPtr(vet, numCompromissos);
}
```

a. (2pts) Desenvolva a função busca, a qual retorna o número de eventos em um determinado mês.

b. (1,5pts) Desenvolva a função gravaLista, a qual grava em um arquivo binário (com o nome passado como parâmetro), todos os compromissos que constam na lista encadeada.

c. (1,5pts) Desenvolva a função leDoArquivo, a qual lê do arquivo (com o nome passado como parâmetro), todos os compromissos e usa a função lista_insere para adicionar os compromissos a uma lista encadeada que deve ser retornada pela função. A função lista_insere não precisa ser desenvolvida, somente será feita a chamada (conforme protótipo apresentado no início dessa questão) dentro da função leDoArquivo.

d. (2pts) Desenvolva a função vetorPtr(lista), a qual retorna um vetor de ponteiros contendo todos os elementos da lista encadeada recebida como parâmetro. O parâmetro passado como referência, numCompromissos, deve ser atualizado com o número de compromissos encontrado na lista encadeada.

```
#include <stdio.h>

typedef struct data {
  int dd, mm, aa;       /*  Dia, mes e ano            */
} Data;

typedef struct compromisso {
```

```
  char descricao[81];    /*  Descricao do compromisso     */
  Data dta;              /*  Data do compromisso          */
} Compromisso;

typedef struct elemento {
  Compromisso compr;        /*  Compromisso armazenado             */
  struct elemento *prox;    /*  Ponteiro para o próximo elemento   */
} Elemento;

int busca(Elemento* lst, int mes);
Elemento* listaInsere (Elemento* lst, char *descricao, int dia, int mes, int ano);
void gravaLista(Elemento *lista, char *nomeArq);
Elemento *leDoArquivo(Elemento *lista, char *nomeArq);
Compromisso ** vetorPtr(Elemento *lista, int *numCompromissos);
void imprimeVetPtr(Compromisso **vet, int n);

main(void) {
  Elemento *lista=NULL;
  Compromisso **vet;
  int numCompromissos=0;
  lista = listaInsere(lista, "compromisso um",25,11,10);
  lista = listaInsere(lista, "compromisso dois",30,11,10);
  lista = listaInsere(lista, "compromisso tres",01,12,10);
  printf("\nNumero de compromissos no mes 11=%d\n",busca(lista,11));
  gravaLista(lista, "compromissos.bin");
  lista = leDoArquivo(lista, "compromissos.bin");
  vet = vetorPtr(lista, &numCompromissos);
  imprime(lista);
  imprimeVetPtr(vet, numCompromissos);
}

Elemento* listaInsere (Elemento* lst, char *descricao, int dia, int mes, int ano) {
  Elemento* novo = (Elemento*) malloc(sizeof(Elemento));
  strcpy(novo->compr.descricao, descricao);
  novo->compr.dta.dd = dia;
  novo->compr.dta.mm = mes;
  novo->compr.dta.aa = ano;
  novo->prox = lst;
  return novo;
}

int busca(Elemento* lst, int mes){
   Elemento *ptr;
   int cont=0;

   for (ptr = lst; ptr != NULL; ptr = ptr->prox)
      if (ptr->compr.dta.mm == mes)
         cont++;
   return cont;
}

void imprime(Elemento* lst){
   Elemento *ptr;
   for (ptr = lst; ptr != NULL; ptr = ptr->prox)
      printf("\nCompromisso=%s          %d/%d/%d",ptr->compr.descricao,ptr->compr.dta.dd,ptr-
>compr.dta.mm,ptr->compr.dta.aa);
}
```

```
void gravaLista(Elemento *lista, char *nomeArq) {
   FILE *arq=fopen(nomeArq,"wb");
   if(arq==NULL){
      printf("erro na abertura do arquivo");
      exit(1);
   }
   Elemento *p = lista;
   printf("\ngravando...\n");
   for(;p!=NULL;p=p->prox){
      fwrite(&p->compr,sizeof(Compromisso),1,arq);
   }
   fclose(arq);
}

Elemento *leDoArquivo(Elemento *lista, char *nomeArq) {
   FILE *arq=fopen(nomeArq,"rb");
   if(arq==NULL){
      printf("erro na abertura do arquivo");
      exit(1);
   }
   Elemento *lst=NULL;
   // ou: Elemento *lst=lista; caso queira adicionar à lista existente
   Compromisso comp;
   printf("\nlendo...\n");
   while (fread(&comp,sizeof(Compromisso),1,arq)) {
      lst = listaInsere(lst, comp.descricao, comp.dta.dd, comp.dta.mm, comp.dta.aa);
   }
   fclose(arq);
   return lst;
}

void imprimeVetPtr(Compromisso **vet, int n) {
   int i=0;
   for(;i<n;i++)
      printf("\nDescricao=%s,  %d//%d//%d\n",  vet[i]->descricao,  vet[i]->dta.dd,  vet[i]->dta.mm,
vet[i]->dta.aa);
}

Compromisso ** vetorPtr(Elemento *lista, int *cont) {
   Elemento *ptr;
   Compromisso **vet;
   int i=0;
   *cont=0;

   // verifica quantos nodos tem na lista:
   for (ptr = lista; ptr != NULL; ptr = ptr->prox,*cont++);

   vet = (Compromisso **) malloc(*cont*sizeof(Compromisso *));

   for (i=0,ptr = lista; ptr != NULL; ptr = ptr->prox,i++) {
      vet[i] = (Compromisso *) malloc(sizeof(Compromisso));
      strcpy(vet[i]->descricao, ptr->compr.descricao);
      vet[i]->dta.dd = ptr->compr.dta.dd;
      vet[i]->dta.mm = ptr->compr.dta.mm;
      vet[i]->dta.aa = ptr->compr.dta.aa;
   }
   return vet;
}
```