Terceira Prova – ED1 – Sistemas de Informação

Turma

Número Sequencial

Prof. Bruno A. N. Travençolo – FACOM-UFU

NOME:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
| E1a |  |
| E1b |  |
| E1c |  |
| E1d |  |
| E2 |  |
|  |  |
| Total |  |
|  |  |

Matrícula:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Instruções. Leia com atenção

1. Esta prova tem folhas numeradas de 1 a 5.
2. Escreva o seu nome e número de matrícula nos campos acima.
3. A prova pode ser feita a lápis ou a caneta
4. Utilizar para todos os problemas a linguagem C
5. Desligue seu celular
6. Sobre a mesa somente caneta, lápis e borracha
7. Não retirar o grampo da prova
8. Os versos das folhas podem ser usados para escrever as respostas
9. Se a codificação não couber em uma linha, continue na linha de baixo
10. Diretivas #includes são opcionais, excetos para os TADs
11. Preencha sua turma no campo acima. Preencha seu número sequencial quando assinar a lista de presença
12. (7 ptos) Considere o TAD TDlinkedList que serve para manipular uma lista duplamente encadeada de números inteiros. Considere que as funções retornam -1 em caso de erro e 0 em caso de sucesso.

|  |  |
| --- | --- |
| TDLinkedList.c | TDLinkedList.h |
| #include "TDLinkedList.h"  typedef struct dlnode DLNode;  struct DlinkedList {  DLNode \*begin;  DLNode \*end;  int size;  };  struct dlnode {  int data;  DLNode \*next;  DLNode \*prev;  }; | typedef struct DlinkedList List;  List\* list\_create();  int list\_free(List \*li);  int list\_push\_front(List \*li, int a);  int list\_push\_back(List \*li, int a);  int list\_insert(List \*li, int pos, int a);  int list\_size(List \*li);  int list\_pop\_front(List \*li);  int list\_pop\_back(List \*li);  int list\_erase(List \*li, int pos);  int list\_front(List \*li, int \*a);  int list\_back(List \*li, int \*a);  int list\_get\_pos(List \*li, int nmat, int \*pos);  **int list\_splice(List \*dest, List \*source, int pos);** |

(a) Implemente a função int list\_erase\_even(List \*li), que retira da lista todos os elementos que são números pares. Pode-se utilizar qualquer função já implementada na lista (que estão no .h)

(b) Uma função comumente encontrada em TAD de listas é a função *splice* (que significa juntar, ligar, emendar). Essa função transfere elementos de uma lista para outra a partir de uma determinada posição. Seu cabeçalho é o seguinte:

int list\_splice(List \*dest, List \*source, int pos);

**dest** - lista de destino (lista que receberá a outra lista)

**source** – lista de origem (lista que será ligada à outra lista)

**pos** – posição na lista de destino (*dest*) em que a lista origem (*source*) será inserida. O primeiro elemento da lista de origem passará a ocupar a posição *pos* na lista destino.

**Retorno da função**: 0 para sucesso; -1 para qualquer tipo de erro

Na implementação do *splice*, nenhum elemento é copiado ou movido, somente os ponteiros internos das duas listas são rearranjados. Após o splice, a lista origem continua existindo, no entanto, ela ficará sem elementos (vazia).

Considerando que a lista em questão armazena números inteiros, desenhe como ficarão a listas após os seguintes comandos. No desenho indique todos os ponteiros.

List \*la = list\_create(); // lista a

List \*lb = list\_create(); // lista b

list\_push\_back(la, 4);

list\_push\_back(la, 9);

list\_push\_back(la, 20);

list\_push\_back(la, 2);

list\_push\_front(lb, 5);

list\_push\_front(lb, 7);

|  |
| --- |
| la |

|  |
| --- |
| lb |

Desenhe como ficarão as listas após o *splice*, que foi chamado utilizando o seguinte comando:

list\_splice(la, lb, 2);

|  |
| --- |
| la |

|  |
| --- |
| lb |

(c) Implemente a função splice conforme orientações do exercício (b)

int list\_splice(List \*dest, List \*source, int pos) {

1. (3 ptos) Considere o seguinte TAD pilha (Stack).

|  |  |
| --- | --- |
| TStack.c | TStack.h |
| #include "TStack.h"  struct stack {  char c[MAX];  int size;  }; | #define MAX 100  typedef struct stack Stack;  Stack \*stack\_create();  void stack\_free(Stack \*st);  int stack\_push(Stack \*st, char c);  int stack\_pop(Stack \*st);  int stack\_top(Stack \*st, char \*c);  int stack\_empty(Stack \*st);  int stack\_size(Stack \*st); |

(a) Considere o problema de decidir se uma dada sequência de parênteses e colchetes está bem-formada (ou seja, parênteses e colchetes são fechados na ordem inversa àquela em que foram abertos). Por exemplo, a sequência

( ( ) [ ( ) ] )

está bem-formada, enquanto ( [ ) ] está malformada. Suponha que a sequência de parênteses e colchetes está armazenada em uma string ASCII s. (Como é hábito em C, o último caractere da string é \0.). Implemente a função *verifica*, para verificar se a string é válida (retorna 1 se válida e 0 caso contrário). Utilizando o TAD Pilha (stack) na solução.

int main(){

char s[100];

// supor que já foi lida a string e que ela está armazenada em s

if (verifica(s))

printf(“válida”);

else print (“inválida”);

}

int verifica(char \*s){