ComprimentoDaLista

Escreva uma função que receba um ponteiro para o primeiro nó de uma lista ligada e retorne o comprimento da lista, isto é, o número de nós da lista.

Utilize a seguinte declaração para a função:
int comprimento_lista(Node *lst)

Obs.: Implemente apenas a função comprimento_lista. O ponteiro recebido pode ser NULL (lista vazia). Lembre-se de liberar a lista no término do programa (veja os slides da Aula 4).

Exemplo de teste (-1 para sair da leitura de dados):

Entrada:
1 2 3 -1
Saída
3

UltimoNoDaLista

Escreva uma função que receba um ponteiro para o primeiro nó de uma lista ligada e retorne um ponteiro para o último nó desta lista.

Utilize a seguinte declaração para a função:
Node *ultimo_no(Node *lst)

Obs.: Implemente apenas a função ultimo_no. O ponteiro recebido pode ser NULL (lista vazia).

Entrada:
3
7 5 2
Saída (campo info do nó):
2

Na função:
lst: 7->5->2->NULL
Retorna o endereço do nó 2 (algo como):
0x128e030

RemoveNoLista

Escreva uma função que receba um ponteiro para o primeiro nó de uma lista encadeada e um inteiro x e faça a remoção do nó que contém o valor x, caso exista. A função deve retornar um ponteiro para o início da lista.

Utilize a seguinte declaração para a função:
Node *remove_no(Node *lst, int x)

Obs.: Implemente apenas a função remove_no. O ponteiro recebido pode ser NULL (lista vazia).

A função deve remover apenas a primeira ocorrência do referido nó, caso exista mais de um. Dica: implemente uma função para buscar um nó na lista (a função devolve um ponteiro para o nó, caso exista ou NULL, caso contrário).

Entrada:
3
7 6 9
7
Saída
6 9

InsereOrdenado

Escreva uma função que receba um ponteiro para o primeiro nó de uma lista encadeada (ordenada em ordem crescente) e um inteiro k. A função deve criar um novo nó com o valor k e inserir este nó na lista mantendo a ordenação dos elementos da lista. A função deve retornar um ponteiro para o início da lista.

Utilize a seguinte declaração para a função:
Node *insere_ordenado(Node *lst, int k)

Obs.: Implemente apenas a função insere_ordenado. O ponteiro recebido pode ser NULL (lista vazia).

Entrada:
3
3 6 9
7
Saída
3 6 7 9

ComparaListas

Dados dois ponteiros para os primeiros nós de duas listas ligadas, crie uma função que permita comparar os dados nos nós das listas para verificar se eles são iguais.
As listas são iguais apenas se tiverem o mesmo número de nós e os nós correspondentes tiverem os mesmos dados.

Utilize a seguinte declaração para a função:
int *compara_listas(Node *lst1, Node *lst2)

Obs.: Implemente apenas a função compara_listas. O ponteiro recebido pode ser NULL (lista vazia).

Entrada:
3

1 2 3
3
1 2 3
Saída
1

IntercalaListas

Dados dois ponteiros para os primeiros nós de duas listas ligadas, crie uma função que permita intercalar os dados nos nós das listas de tal forma a criar uma (nova) terceira lista ligada mas organizada na forma crescente.
Os dados em ambas as listas iniciais estão organizados em ordem crescente.

Utilize a seguinte declaração para a função:
Node *intercala_listas(Node *lst1, Node *lst2)

Obs.: Implemente apenas a função intercala_listas. O ponteiro recebido pode ser NULL (lista vazia).

Entrada:
3
1 3 5
4
2 3 8 11
Saída
1 2 3 3 5 8 11

InverteLista

Dado um ponteiro para o primeiro nó de uma lista ligada, crie uma função que permita inverter a ordem na lista.
Isto é, as ligações devem ser invertidas. O último elemento da lista deve apontar para NULL. A função deve devolver um ponteiro ao primeiro elemento da lista.

Utilize a seguinte declaração para a função:
void *inverte_lista(Node *lst1)

Obs.: Implemente apenas a função inverte_lista. O ponteiro recebido pode ser NULL (lista vazia).

Não crie uma nova lista, a ideia é fazer a inversão na própria lista.

Entrada:
3
1 2 3
Saída
3 2 1

SomaListas

Calculadora "ilimitada"

Crie funções para calcular a soma e o produto de números inteiros positivos com número "ilimitado" de dígitos. Por exemplo, o programa deve calcular expressões como: 123456789 x 987654321, e apresentar todos os dígitos do resultado do cálculo (121932631112635269).

Considere uma lista duplamente ligada para representar cada número, onde cada nó da lista contém um dígito.
Considere a seguinte estrutura para cada nó da lista:

```
typedef struct node Node;
struct node {
   int info;
   Node *ant;
   Node *prox;
};
```

Escreva a seguinte função:
(a) soma_listas: esta função recebe duas listas ligadas (ponteiros para o primeiro nó de cada lista) representando dois números inteiros positivos e calcula a soma desses números. O resultado deve ser armazenado em uma outra lista ligada, utilizando o mesmo formato (um dígito em cada nó).
A função deve retornar um ponteiro para o início da lista resultante.

Utilize a seguinte declaração para a função:
Node *soma_listas(Node *lst1, Node *lst2)

Obs.: Implemente apenas a função soma_listas. O ponteiro recebido pode ser NULL (lista vazia).

Entrada:
3
1 1 1
3
2 2 2
Saída
3 3 3

Extra: MultiplicaListas (opcional)

Calculadora "ilimitada"

Estendendo o exercício anterior, escreva a seguinte função:
(b) multiplica_listas: esta função recebe duas listas ligadas (ponteiros para o primeiro nó de cada lista) representando dois números inteiros positivos e calcula o produto desses números. O resultado deve ser armazenado em uma outra lista ligada, utilizando o mesmo formato (um dígito em cada nó).

A função deve retornar um ponteiro para o início da lista resultante.

Utilize a seguinte declaração para a função:
Node *multiplica_listas(Node *lst1, Node *lst2)

Obs.: Implemente apenas a função multiplica_listas. O ponteiro recebido pode ser NULL (lista vazia).

Dica: para o cálculo do produto, procure aproveitar a função soma já implementada.

Entrada:
2
1 2
2
1 2
Saída
1 4 4

PilhaDinamica

Considere uma pilha dinâmica de números reais (utilizando lista ligada) definida pelas seguintes estruturas:

```
typedef struct node Node;
struct node {
  float valor;
  Node* prox;
};
typedef struct pilha Pilha;
struct pilha {
  Node* topo;
};
```

Implemente as seguintes funções:

Pilha *cria_pilha ();
int pilha_vazia(Pilha *p);
void empilha(Pilha *pilha, float valor);
float desempilha(Pilha *p);
void libera_pilha(Pilha *p);

Obs.1: Escreva apenas as funções indicadas, não é necessário escrever a função main. Também não é necessário declarar as estruturas da pilha.
Obs.2: Na operação desempilha, trate o caso em que a pilha está vazia, apresentando a mensagem: “Pilha vazia” e finalize a execução. Com exceção deste caso, não apresente mensagens (com printf) a partir das funções.

Exemplo de teste:

Entrada (1-empilha, 2-desempilha, 3-sair):
1 2 (empilha 2)
1 5 (empilha 5)
2 (desempilha)

3
Saída (conteúdo da pilha):
2

TopoETamanhoDaPilha

Considere uma pilha dinâmica de inteiros (utilizando lista ligada) definida pelas seguintes estruturas:

```
typedef struct node Node;
struct node {
  int valor;
  Node* prox;
};
typedef struct pilha Pilha;
struct pilha {
  Node* topo;
};
```

Escreva as seguintes funções:

int elemento_topo(Pilha *p): recebe um ponteiro para uma pilha e retorna o valor armazenado no topo da pilha (preservando o elemento no topo da pilha).
int tamanho(Pilha *p): recebe um ponteiro para uma pilha e retorna o tamanho da pilha (preservando os elementos da pilha).

Considere as seguintes funções de pilha (já implementadas):

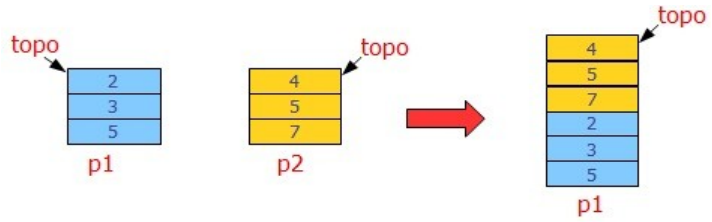
```
Pilha* cria_pilha();
int pilha_vazia(Pilha *pilha);
void empilha(Pilha *pilha, int valor);
int desempilha(Pilha *pilha);
void libera_pilha(Pilha *p);
```

Obs.1: Implemente as funções indicadas (elemento_topo e tamanho), não é necessário escrever a função main.
Obs.2: Também não é necessário declarar as estruturas da pilha e nem implementar as funções da pilha, estas já estão presentes no código de verificação. Utilize apenas as funções de pilha acima para manipular a pilha.
Exemplo de teste:

Entrada: 1 2 5 0
Saída (tamanho, topo, conteúdo da pilha):
4
0
0 5 2 1

ConcatenaPilhas

Escreva uma função que receba dois ponteiros para duas pilhas p1 e p2 e copie todos os elementos da pilha p2 para o topo da pilha p1, conforme o exemplo:



Uitlize a seguinte declaração:

void concatena_pilhas(Pilha *p1, Pilha *p2)
Considere uma pilha dinâmica de inteiros (utilizando lista ligada) definida pelas seguintes estruturas:

```
typedef struct node Node;
struct node {
  int valor;
  Node* prox;
};
```

```
typedef struct pilha Pilha;
struct pilha {
  Node* topo;
};
```

Considere as seguintes funções para a pilha (já implementadas):
Pilha* cria_pilha();
int pilha_vazia(Pilha *pilha);
void empilha(Pilha *pilha, int valor);
int desempilha(Pilha *pilha);
void libera_pilha(Pilha *p);

Obs.1: Escreva apenas a função concatena_pilhas, não é necessário escrever a função main.
Obs.2: Também não é necessário declarar as estruturas da pilha e nem implementar as funções da pilha, estas já estão presentes no código de verificação. Utilize apenas as funções de pilha acima para manipular a pilha.

Exemplo de teste:

Entrada:
3
135
4
2 3 8 11
Saída:
11 8 3 2 5 3 1

CopiaPilha

Escreva uma função que receba um ponteiro para uma pilha p e retorne uma cópia da pilha p, mantendo a pilha p com o seu conteúdo original.
Utilize a seguinte declaração:
Pilha *copia_pilha(Pilha *p)

Considere uma pilha dinâmica de inteiros (utilizando lista ligada) definida pelas seguintes estruturas:

```c
typedef struct node Node;
struct node {
  int valor;
  Node* prox;
};
```

```c
typedef struct pilha Pilha;
struct pilha {
  Node* topo;
};
```

Considere as seguintes funções para a pilha (já implementadas):

```
Pilha* cria_pilha();
int pilha_vazia(Pilha *pilha);
void empilha(Pilha *pilha, int valor);
int desempilha(Pilha *pilha);
void libera_pilha(Pilha *p);
```

Obs.1: Escreva apenas a função copia_pilha, não é necessário escrever a função main.
Obs.2: Também não é necessário declarar as estruturas da pilha e nem implementar as funções da pilha, estas já estão presentes no código de verificação. Utilize apenas as funções de pilha acima para manipular a pilha.

Entrada:
5
-1 4 7 9 0
Saída: (pilhas original e cópia; copia != original ?)
0 9 7 4 -1
0 9 7 4 -1
1

PalindromoComPilha

Considere as seguintes estruturas para uma pilha dinâmica de caracteres (utilizando lista ligada):

```c
typedef struct node Node;
   struct node {
   char valor;
   Node *prox;
};
typedef struct pilha Pilha;
struct pilha {
   Node *topo;
};
```

Escreva uma função que receba um ponteiro para uma pilha de caracteres e verifique se a palavra ou frase contida na pilha é um palíndromo. Um palíndromo é uma palavra ou frase que pode ser lida da esquerda para a direita ou da direita para a esquerda, ignorando espaços em branco, pontuações e acentos. Por exemplo as palavras: “osso”, “radar, e as frases: “arara rara” e “a grama é amarga” são palíndromos.

Utilize a seguinte declaração:
int palindromo(Pilha *p)
Considere as seguintes funções para a pilha (já implementadas):
Pilha* cria_pilha();
int pilha_vazia(Pilha *pilha);
void empilha(Pilha *pilha, int valor);
int desempilha(Pilha *pilha);
void libera_pilha(Pilha *p);

Obs.1: Escreva apenas a função palindromo, não é necessário escrever a função main (mas é possível acrescentar funções auxiliares se necessário).
Obs.2: Também não é necessário declarar as estruturas da pilha e nem implementar as funções da pilha, estas já estão presentes no código de verificação. Utilize apenas as funções de pilha acima para manipular a pilha.

Exemplo:

Entrada:
9
a r a r a r a r a
Saída:
1

FilaDinamica

Considere uma fila dinâmica de reais (utilizando lista ligada) definida pelas seguintes estruturas:

```
typedef struct node Node;
struct node {
    float valor;
    Node *prox;
};
typedef struct fila Fila;
struct fila {
    Node *inicio, *fim;
};
```

Implemente as seguintes funções:

Fila* cria_fila();
int fila_vazia(Fila *f);
void insere(Fila *f, float valor);
float retira(Fila *f);
void libera_fila(Fila *f);

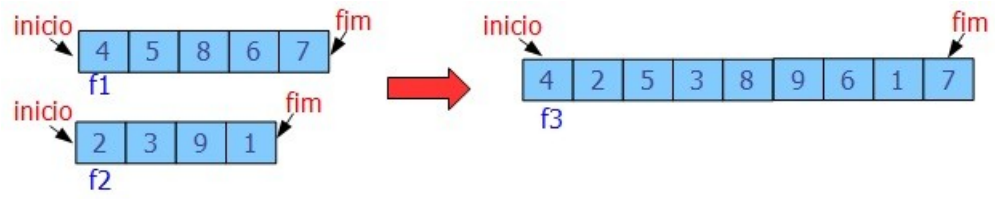
Obs.1: Escreva apenas as funções indicadas, não é necessário escrever a função main. Também não é necessário declarar as estruturas da fila.
Obs.2: Na operação retira, trate o caso em que a fila está vazia, apresentando a mensagem: “Fila vazia” e finalize a execução. Com exceção deste caso, não apresente mensagens (com printf) a partir das funções.

Exemplo de teste:

Entrada (1-insere, 2-retira, 3-sair):
1 2 (insere 2)
1 5 (insere 5)
2 (retira)
3
Saída (conteúdo da fila):
5

CombinaFilas

Escreva uma função que receba três ponteiros para três filas, f1, f2 e f3, e transfira para f3 os elementos de f1 e f2 de forma alternada, conforme o exemplo a seguir:



Considere uma fila dinâmica de reais (utilizando lista ligada) definida pelas seguintes estruturas:

```
typedef struct node Node;
struct node {
    float valor;
    Node *prox;
};
```

```
typedef struct fila Fila;
struct fila {
    Node *inicio, *fim;
};
```

Considere as seguintes funções para a pilha (já implementadas):

```
Fila* cria_fila();
int fila_vazia(Fila *f);
void insere(Fila *f, float valor);
float retira(Fila *f);
void libera_fila(Fila *f);
```

Obs.1: Escreva apenas a função combina_filas, não é necessário escrever a função main.
Obs.2: Também não é necessário declarar as estruturas da fila e nem implementar as funções da fila, estas já estão presentes no código de verificação. Utilize apenas as funções de fila acima para manipular a fila.

Exemplo de teste:

Entrada:
3
135
4
2 3 8 11
Saída:
1 2 3 3 5 8 11

VerificaExpressao

Escreva um programa que lê uma expressão (com no máximo 100 caracteres) e verifica se todos os parênteses, colchetes e chaves são abertos e fechados na ordem correta. O programa imprime "SIM" (caso os parênteses, colchetes e chaves tenham sido abertos e fechados corretamente) e "NAO" caso contrário.

Por exemplo, a expressão "(10+60-[40+x]-{80})" retorna SIM, mas a expressão "[10+60-(40+x]-{80})" retorna NAO.

Entrada:
15+(30*[x+2]+{y/5}+{z/9})/(8+1)
Saída:
SIM