**Exercicios ED – Listas 04 e 06**

**Participantes: Alicia Abade Muniz**

**Gustavo Balbino de Sousa**

**Lista 04**

**Ex 13**

// Considere uma lista contendo números inteiros positivos. Faça uma função que retorne a média da lista.

#include <stdio.h>

#include <stdlib.h>

typedef struct *no* {

    int dado;

    struct *no*\* prox;

} *t\_no*;

void adicionar(*t\_no*\*\* *cab*, int *novo\_dado*) {

*t\_no*\* novoNo = (*t\_no*\*)malloc(sizeof(*t\_no*)); //aloca memória para um novo nó

*t\_no*\* ultimo = \**cab*; //ponteiro para percorrer a lista até o último elemento

    novoNo->dado = *novo\_dado*; //atribui o novo dado ao campo 'dado' do novo nó

    novoNo->prox = NULL; //define o próximo nó como NULL, pois será o último da lista

    if (\**cab* == NULL) { //verifica se a lista está vazia

        \**cab* = novoNo; //se estiver vazia, o novo nó se torna o primeiro da lista

        return;

    }

    while (ultimo->prox != NULL) // Percorre a lista até encontrar o último nó

        ultimo = ultimo->prox;

    ultimo->prox = novoNo; // Adiciona o novo nó ao final da lista

}

float calcularMedia(*t\_no*\* *cab*) {

    if (*cab* == NULL)

        return 0.0; // Se a lista estiver vazia, retorna média zero

    int soma = 0;

    int contador = 0;

*t\_no*\* atual = *cab*; // Ponteiro para percorrer a lista

    while (atual != NULL) {

        soma += atual->dado; //soma o valor do nó atual à variável 'soma'

        contador++; //incrementa o contador de elementos

        atual = atual->prox; //avança para o próximo nó na lista

    }

    //retorna a média como um número de ponto flutuante

    return (float)soma / contador;

}

void imprimirLista(*t\_no*\* *no*) {

    while (*no* != NULL) { //enquanto houver nós na lista

        printf("%d ", *no*->dado);

*no* = *no*->prox; //avança para o próximo nó na lista

    }

    printf("\n");

}

int main() {

*t\_no*\* cab = NULL;

    adicionar(&cab, 3);

    adicionar(&cab, 7);

    adicionar(&cab, 5);

    printf("Lista: ");

    imprimirLista(cab);

    printf("Media da lista: %.2f\n", calcularMedia(cab));

    return 0;

}

**Ex 16**

// Fazer uma func¸ao que copia uma lista L1 em uma outra lista L2.

#include <stdio.h>

#include <stdlib.h>

typedef struct *no* {

    int dado;

    struct *no*\* prox;

} *t\_no*;

void adicionar(*t\_no*\*\* *cab*, int *novo\_dado*) {

*t\_no*\* novoNo = (*t\_no*\*)malloc(sizeof(*t\_no*));

*t\_no*\* ultimo = \**cab*;

    novoNo->dado = *novo\_dado*;

    novoNo->prox = NULL;

    if (\**cab* == NULL) {

        \**cab* = novoNo;

        return; // Sai da função

    }

    while (ultimo->prox != NULL) {

        ultimo = ultimo->prox;

    }

    ultimo->prox = novoNo;

}

void copiarLista(*t\_no*\* *L1*, *t\_no*\*\* *L2*) {

*t\_no*\* atual = *L1*;

    while (atual != NULL) {

        adicionar(*L2*, atual->dado); //adiciona o valor do nó atual de L1 na lista L2

        atual = atual->prox;

    }

}

void imprimirLista(*t\_no*\* *no*) {

    while (*no* != NULL) {

        printf("%d ", *no*->dado);

*no* = *no*->prox;

    }

    printf("\n");

}

int main() {

*t\_no*\* L1 = 0;

*t\_no*\* L2 = 0;

    // Exemplo de adição de elementos na lista L1

    adicionar(&L1, 3);

    adicionar(&L1, 7);

    adicionar(&L1, 5);

    printf("Lista L1: ");

    imprimirLista(L1);

    copiarLista(L1, &L2);

    printf("Lista L2: ");

    imprimirLista(L2);

    return 0;

}

**Ex 21**

//  Fac¸a um programa que receba do usuario dois vetores, ´ A e B, com 10 numeros inteiros ´

// cada. Crie um novo vetor denominado C calculando C = A - B. Mostre na tela os infos

// do vetor C.

#include <stdio.h>

#include <stdlib.h>

// Definição da estrutura do nó da lista

typedef struct *no*{

    int info;

    struct *no*\* proximo;

} *t\_no*;

*t\_no*\* criarno(int *info*) {

*t\_no*\* novoNo = (*t\_no*\*)malloc(sizeof(*t\_no*));

    novoNo->info = *info*;

    novoNo->proximo = NULL;

    return novoNo;

}

void adicionarNo(*t\_no*\*\* *cab\_ref*, int *info*) {

*t\_no*\* novoNo = criarno(*info*);

*t\_no*\* ultimo = \**cab\_ref*;

    if (\**cab\_ref* == NULL) {

        \**cab\_ref* = novoNo;

        return;

    }

    while (ultimo->proximo != NULL) {

        ultimo = ultimo->proximo;

    }

    ultimo->proximo = novoNo;

}

//função para criar um novo vetor C a partir dos vetores A e B

*t\_no*\* criarVetorC(*t\_no*\* *cabA*, *t\_no*\* *cabB*) {

*t\_no*\* cabC = NULL; //criação do vetor C

*t\_no*\* atualA = *cabA*;//ponteiro para percorrer o vetor A

*t\_no*\* atualB = *cabB*;

    while (atualA != NULL && atualB != NULL) { //enquanto houver elementos em ambos os vetores

        int valorC = atualA->info - atualB->info; //Calcula o elemento de C como A - B

        adicionarNo(&cabC, valorC); //Adiciona o elemento calculado ao vetor C

        atualA = atualA->proximo;

        atualB = atualB->proximo;

    }

    return cabC; // Retorna o vetor C

}

void mostrarLista(*t\_no*\* *cab*) {

*t\_no*\* atual = *cab*; //ponteiro para percorrer a lista, começa do primeiro nó

    while (atual != NULL) { //Enquanto houver nós na lista

        printf("%d ", atual->info);

        atual = atual->proximo;

    }

    printf("\n");

}

int main() {

*t\_no*\* cabA = 0;

*t\_no*\* cabB = 0;

*t\_no*\* cabC = 0;

    int valor;

    printf("Digite 10 valores inteiros para o vetor A:\n");

    for (int i = 0; i < 10; i++) {

        printf("A[%d]: ", i);

        scanf("%d", &valor);

        adicionarNo(&cabA, valor); // Adiciona o valor ao vetor A

    }

    printf("Digite 10 valores inteiros para o vetor B:\n");

    for (int i = 0; i < 10; i++) {

        printf("B[%d]: ", i);

        scanf("%d", &valor);

        adicionarNo(&cabB, valor); // Adiciona o valor ao vetor B

    }

    // Criação do vetor C a partir dos vetores A e B

    cabC = criarVetorC(cabA, cabB);

    // Exibição dos elementos do vetor C

    printf("\nVetor C (C = A - B):\n");

    mostrarLista(cabC);

    return 0;

}

**Ex 24**

// 24. Escreva uma func¸ao que verifica se duas listas dadas s ˜ ao iguais (mesmo conte ˜ udo

#include <stdio.h>

#include <stdlib.h>

typedef struct *no* {

    int dado;

    struct *no*\* prox;

} *t\_no*;

void adicionar(*t\_no*\*\* *cab*, int *novo\_dado*) {

*t\_no*\* novoNo = (*t\_no*\*)malloc(sizeof(*t\_no*));

*t\_no*\* ultimo = \**cab*;

    novoNo->dado = *novo\_dado*;

    novoNo->prox = NULL;

    if (\**cab* == NULL) {

        \**cab* = novoNo;

        return;

    }

    while (ultimo->prox != NULL) {

        ultimo = ultimo->prox;

    }

    ultimo->prox = novoNo;

}

int saoIguais(*t\_no*\* *L1*, *t\_no*\* *L2*) {

    //ponteiros para percorrer as duas listas

*t\_no*\* p1 = *L1*;

*t\_no*\* p2 = *L2*;

    while (p1 != NULL && p2 != NULL) {

        if (p1->dado != p2->dado) { //compara os dados dos nós atuais

            return 0;

        }

        p1 = p1->prox;

        p2 = p2->prox;

    }

    //se ambas as listas terminaram ao mesmo tempo, elas são iguais

    return (p1 == NULL && p2 == NULL);

}

void imprimirLista(*t\_no*\* *no*) {

    while (*no* != NULL) {

        printf("%d ", *no*->dado); //imprime o valor do nó atual

*no* = *no*->prox; //avança para o próximo nó na lista

    }

    printf("\n");

}

int main() {

*t\_no*\* L1 = 0;

*t\_no*\* L2 = 0;

    //adição de elementos na lista L1

    adicionar(&L1, 1);

    adicionar(&L1, 2);

    adicionar(&L1, 3);

    //eadição de elementos na lista L2

    adicionar(&L2, 3);

    adicionar(&L2, 2);

    adicionar(&L2, 3);

    printf("Lista L1: ");

    imprimirLista(L1);

    printf("Lista L2: ");

    imprimirLista(L2);

    //verifica se as listas L1 e L2 são iguais

    if (saoIguais(L1, L2)) {

        printf("As listas sao iguais.\n");

    } else {

        printf("As listas nao sao iguais.\n");

    }

    return 0;

}

**Lista 06**

**Ex 05**

// 5. Considere uma pilha que armazene caracteres. Fac¸a uma func¸ao para determinar se ˜

// uma string e da forma XY, onde X ´ e uma cadeia formada por caracteres arbitr ´ arios e Y ´

// e o reverso de X. Por exemplo, se x = ABCD, ent ´ ao y = DCBA. Considere que x e y s ˜ ao˜

// duas strings distintas.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct *no* {

    char dado;

    struct *no*\* prox;

} *t\_no*;

void empilhar(*t\_no*\*\* *topo*, char *valor*) {

*t\_no*\* novoNo = (*t\_no*\*)malloc(sizeof(*t\_no*));

    novoNo->dado = *valor*; //atribui o valor ao campo 'dado' do novo nó

    novoNo->prox = \**topo*; //define o próximo nó como o nó atual do topo

    \**topo* = novoNo;

}

char desempilhar(*t\_no*\*\* *topo*) {

    if (\**topo* == NULL) {

        return '\0';

    }

*t\_no*\* temp = \**topo*;

    char valor = temp->dado;

    \**topo* = (\**topo*)->prox;

    free(temp);

    return valor;

}

int saoXy(const char\* *str1*, const char\* *str2*) {

    int n1 = strlen(*str1*);

    int n2 = strlen(*str2*);

    if (n1 != n2) {

        return 0; //se as strings tiverem comprimentos diferentes, não podem ser reversas uma da outra

    }

    //verifica caractere por caractere

    for (int i = 0; i < n1; i++) {

        if (*str1*[i] != *str2*[n1 - i - 1]) {

            return 0; //etorna falso se houver uma diferença

        }

    }

    return 1; //retorna verdadeiro se forem reversas uma da outra

}

int main() {

    char str1[100], str2[100];

    printf("Digite a primeira string: ");

    scanf("%s", str1);

    printf("Digite a segunda string: ");

    scanf("%s", str2);

    //verifica se sao xy

    if (saoXy(str1, str2)) {

        printf("eh XY!.\n", str1, str2);

    } else {

        printf("Nao eh XY.\n", str1, str2);

    }

    return 0;

}

**Ex 08**

// Escreva um programa que utilize uma pilha para verificar se expressoes aritm ˜ eticas est ´ ao˜

// com a parentizac¸ao correta. O programa deve verificar express ˜ oes para ver se cada “abre ˜

// parenteses” tem um “fecha par ˆ enteses” correspondente. Ex.: ˆ

// Correto: ( ( ) ) – ( ( )( ) ) – ( ) ( )

// Incorreto: )( – ( ( ) ( – ) ) ( (

#include <stdio.h>

#include <stdlib.h>

typedef struct *no* {

    char dado;

    struct *no*\* prox;

} *t\_no*;

void empilhar(*t\_no*\*\* *topo*, char *valor*) {

*t\_no*\* novoNo = (*t\_no*\*)malloc(sizeof(*t\_no*));

    novoNo->dado = *valor*;

    novoNo->prox = \**topo*;

    \**topo* = novoNo;

}

char desempilhar(*t\_no*\*\* *topo*) {

    if (\**topo* == NULL) {

        return '\0';

    }

*t\_no*\* temp = \**topo*;

    char valor = temp->dado;

    \**topo* = (\**topo*)->prox;

    free(temp);

    return valor;

}

int verificarExpressao(const char\* *expressao*) {

*t\_no*\* pilha = NULL;

    for (int i = 0; *expressao*[i] != '\0'; i++) {

        if (*expressao*[i] == '(') {

            empilhar(&pilha, '('); //Empilha parênteses de abertura

        } else if (*expressao*[i] == ')') {

            if (pilha == NULL) {

                return 0; //retorna falso se tentar desempilhar uma pilha vazia

            }

            desempilhar(&pilha); //desempilha parênteses de fechamento

        }

    }

    //verifica se a pilha está vazia após processar a expressão

    return pilha == NULL;

}

int main() {

    const char\* expressoes[] = {"((()))","(()(()))","()()",")(","((()(",")()("};

    int n = sizeof(expressoes) / sizeof(expressoes[0]); //calcula o número de expressões

    for (int i = 0; i < n; i++) {

        if (verificarExpressao(expressoes[i])) {

            printf("Correto: %s\n", expressoes[i]);

        } else {

            printf("Incorreto: %s\n", expressoes[i]);

        }

    }

    return 0; // Retorna 0 para indicar que o programa foi executado com sucesso

}

**Ex 13**

// 13. Desenvolva uma func¸ao para testar se duas pilhas P1 e P2 s ˜ ao iguais.

#include <stdio.h>

#include <stdlib.h>

typedef struct *no* {

    int dado;

    struct *no*\* prox;

} *t\_no*;

void empilhar(*t\_no*\*\* *topo*, int *valor*) {

*t\_no*\* novoNo = (*t\_no*\*)malloc(sizeof(*t\_no*));

    novoNo->dado = *valor*;

    novoNo->prox = \**topo*;

    \**topo* = novoNo;

}

//função para desempilhar um elemento

int desempilhar(*t\_no*\*\* *topo*) {

    if (\**topo* == NULL) {

        return 0;

    }

*t\_no*\* temp = \**topo*;

    int valor = temp->dado;

    \**topo* = (\**topo*)->prox;

    free(temp);

    return valor;

}

//função para verificar se duas pilhas são iguais

int pilhasIguais(*t\_no*\* *P1*, *t\_no*\* *P2*) {

*t\_no* \*temp1 = *P1*, \*temp2 = *P2*;

    //percorre as duas pilhas comparando os elementos

    while (temp1 != NULL && temp2 != NULL) {

        if (temp1->dado != temp2->dado) {

            return 0; // Retorna falso se encontrar elementos diferentes

        }

        temp1 = temp1->prox;

        temp2 = temp2->prox;

    }

    //verifica se ambas as pilhas chegaram ao fim, têm o mesmo tamanho

    return (temp1 == NULL && temp2 == NULL);

}

//função para imprimir a pilha

void imprimirPilha(*t\_no*\* *topo*) {

*t\_no*\* temp = *topo*;

    while (temp != NULL) {

        printf("%d ", temp->dado);

        temp = temp->prox;

    }

    printf("\n");

}

int main() {

*t\_no*\* P1 = NULL;

*t\_no*\* P2 = NULL;

    //exemplo de adição de elementos nas pilhas

    empilhar(&P1, 1);

    empilhar(&P1, 2);

    empilhar(&P1, 3);

    empilhar(&P2, 1);

    empilhar(&P2, 2);

    empilhar(&P2, 3);

    printf("Pilha P1: ");

    imprimirPilha(P1);

    printf("Pilha P2: ");

    imprimirPilha(P2);

    if (pilhasIguais(P1, P2)) {

        printf("As pilhas P1 e P2 sao iguais.\n");

    } else {

        printf("As pilhas P1 e P2 nao sao iguais.\n");

    }

    while (P1 != NULL) {

        desempilhar(&P1);

    }

    while (P2 != NULL) {

        desempilhar(&P2);

    }

    return 0;

}

**Ex 14**

// 14. Desenvolva uma operac¸ao para transferir elementos de uma pilha P1 para uma pilha P2 ˜

// (copia).

#include <stdio.h>

#include <stdlib.h>

typedef struct *no* {

    int dado;

    struct *no*\* prox;

} *t\_no*;

void empilhar(*t\_no*\*\* *topo*, int *valor*) {

*t\_no*\* novoNo = (*t\_no*\*)malloc(sizeof(*t\_no*));

    novoNo->dado = *valor*;

    novoNo->prox = \**topo*;

    \**topo* = novoNo;

}

// Função para desempilhar um elemento

int desempilhar(*t\_no*\*\* *topo*) {

    if (\**topo* == NULL) {

        return 0;

    }

*t\_no*\* temp = \**topo*;

    int valor = temp->dado;

    \**topo* = (\**topo*)->prox;

    free(temp);

    return valor;

}

//função para copiar elementos de P1 para P2

void copiarPilha(*t\_no*\*\* *P1*, *t\_no*\*\* *P2*) {

*t\_no*\* temp = NULL;

    //transfere elementos de P1 para uma pilha temporária

    while (\**P1* != NULL) {

        empilhar(&temp, desempilhar(*P1*));

    }

    //transfere elementos da pilha temporária para P2

    while (temp != NULL) {

        empilhar(*P2*, desempilhar(&temp));

    }

}

void imprimirPilha(*t\_no*\* *topo*) {

*t\_no*\* temp = *topo*;

    while (temp != NULL) {

        printf("%d ", temp->dado);

        temp = temp->prox;

    }

    printf("\n");

}

int main() {

*t\_no*\* P1 = 0;

*t\_no*\* P2 = 0;

    empilhar(&P1, 1);

    empilhar(&P1, 2);

    empilhar(&P1, 3);

    //imprime a pilha P1 antes da cópia

    printf("Pilha P1 antes da copia: ");

    imprimirPilha(P1);

    //cpia os elementos de P1 para P2

    copiarPilha(&P1, &P2);

    printf("Pilha P1 apos a copia: ");

    imprimirPilha(P1);

    printf("Pilha P2 apos a copia: ");

    imprimirPilha(P2);

    while (P1 != NULL) {

        desempilhar(&P1);

    }

    while (P2 != NULL) {

        desempilhar(&P2);

    }

    return 0;

}