**Aluno: Rafael Kazuhito Vidal Beltrão**

**Análise e Desenvolvimento de Sistemas – Turma C**

**Repositório com os códigos para teste https://github.com/rkazuhito/Faculdade/tree/main/2o%20semestre/estrutura%20de%20dados/atividade%20Pilha%2014-03**

**ESTRUTURA DE UMA PILHA COMENTADA**

#include <stdio.h> // funções de entrada e saída padrão, como printf() e scanf()

#include <conio.h> // fornece funções para controle de console

#include <stdlib.h> // fornece acesso a funções como malloc() e exit()

struct Pilha { //estrutura da pilha. os elementos serão float

int topo; // Índice do topo da pilha

int capa; // Capacidade máxima da pilha

float \*pElem; // Ponteiro para o primeiro elemento da pilha

};

void criarpilha( struct Pilha \*p, int c ){ // inicializa a pilha

p->topo = -1; // Inicializa o topo como -1, indicando que a pilha está vazia

p->capa = c; // Define a capacidade da pilha

p->pElem = (float\*) malloc (c \* sizeof(float)); // Aloca memória para a pilha

}

//// função - testa se a pilha está vazia

int estavazia ( struct Pilha \*p ){ //verifica se a pilha está vazia. Retorna 1 se estiver vazia e 0 caso contrário.

if( p-> topo == -1 )

return 1; // true - vazio

else

return 0; // false - não vazia

}

/// função testa se a pilha está cheia

int estacheia ( struct Pilha \*p ){ //verifica se a pilha está cheia. Retorna 1 se estiver cheia e 0 caso contrário

if (p->topo == p->capa - 1)

return 1; // true - cheia

else

return 0; // false - não cheia

}

//// função emp

void empilhar ( struct Pilha \*p, float v){ //adiciona um elemento à pilha. Recebe um ponteiro para a pilha e um valor v a ser adicionado

p->topo++; // Incrementa o topo da pilha (muda a posição)

p->pElem [p->topo] = v; // Armazena o valor na nova posição do topo

}

float desempilhar ( struct Pilha \*p ){ // remove e retorna o elemento no topo da pilha.

float aux = p->pElem [p->topo]; // Salva o valor do topo

p->topo--; // Decrementa o topo, aponta para o penúltimo

return aux; // Retorna o valor removido para exibi-lo na tela

}

float retornatopo ( struct Pilha \*p ){ //mostra o que tem no topo da pilha

return p->pElem [p->topo]; // Retorna o valor do topo da pilha

}

int main(){

struct Pilha minhapilha; // variável para armazenar a pilha

int capacidade, op; // Declara variáveis para armazenar a capacidade da pilha e a opção do usuário

float valor; // Declara uma variável para armazenar valores a serem empilhados

printf( "\nCapacidade da pilha? " );

scanf( "%d", &capacidade ); // Obtém a capacidade da pilha a partir do usuário

criarpilha (&minhapilha, capacidade); // Inicializa a pilha com a capacidade especificada

while( 1 ){ /\* loop infinito \*/

// Menu de opções

printf("\n1- empilhar (push)\n");

printf("2- desempilhar (POP)\n");

printf("3- Mostrar o topo \n");

printf("4- sair\n");

printf("\nopcao? ");

scanf("%d", &op); // Obtém a opção do usuário

switch (op){

case 1: // Opção para empilhar um valor

if( estacheia( &minhapilha ) == 1 ) // Verifica se a pilha está cheia

printf("\nPILHA CHEIA! \n"); // Exibe mensagem de pilha cheia

else {

printf("\nVALOR? "); //caso não esteja cheia poderá adicionar um valor

scanf("%f", &valor); // Obtém o valor a ser empilhado

empilhar (&minhapilha, valor); // Empilha o valor

}

break;

case 2: // Opção para desempilhar um valor

if ( estavazia(&minhapilha) == 1 ) // Verifica se a pilha está vazia

printf( "\nPILHA VAZIA! \n" ); // Exibe mensagem de pilha vazia

else{ //caso não esteja vazia será possível desempilha um valor

valor = desempilhar (&minhapilha); // Desempilha um valor

printf ( "\n%.1f DESEMPILHADO!\n", valor ); // Exibe o valor desempilhado

}

break;

case 3: // Opção para mostrar o topo da pilha

if ( estavazia (&minhapilha) == 1 ) // Verifica se a pilha está vazia

printf( "\nPILHA VAZIA!\n" ); // Exibe mensagem de pilha vazia

else { //caso não esteja vazia

valor = retornatopo (&minhapilha); // Obtém o valor do topo

printf ( "\nTOPO: %.1f\n", valor ); // Exibe o valor do topo

}

break;

case 4: // Opção para sair do programa

exit(0); // Encerra o programa

default: // Opção inválida

printf( "\nOPCAO INVALIDA! \n" ); // Exibe mensagem de opção inválida

}

}

}

**EXEMPLO DE PILHA PRÁTICO E COMENTADO**

Um exemplo prático de uma pilha pode ser a função de desfazer (undo) em um editor de texto. Vamos supor que você está digitando um texto em um editor de texto simples. Cada vez que você digita uma palavra ou uma frase e pressiona "Enter", o texto é adicionado à pilha de ações. Se você cometer um erro ou decidir desfazer sua última ação, o editor de texto simplesmente desempilha a última ação (ou seja, a última palavra ou frase digitada) e a remove do texto.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_TEXT\_LENGTH 100 // Define o comprimento máximo do texto

struct Pilha {

char texto[MAX\_TEXT\_LENGTH]; // Array para armazenar o texto

struct Pilha \*prox; // Ponteiro para o próximo item na pilha

};

struct Pilha \*topo = NULL; // Ponteiro para o topo da pilha, inicializado como NULL indicando pilha vazia

void push(char \*texto) {

struct Pilha \*novo = (struct Pilha\*)malloc(sizeof(struct Pilha)); // Aloca memória para um novo item da pilha

if (novo == NULL) { // Verifica se a alocação de memória foi bem sucedida

printf("Erro: não foi possível alocar memória.\n"); // Imprime uma mensagem de erro

exit(1); // Sai do programa com código de erro

}

strcpy(novo->texto, texto); // Copia o texto fornecido para o novo item da pilha

novo->prox = topo; // Define o próximo item da pilha como o antigo topo

topo = novo; // Atualiza o topo da pilha para apontar para o novo item

}

void pop() {

if (topo == NULL) { // Verifica se a pilha está vazia

printf("Erro: pilha vazia, nada para desfazer.\n"); // Imprime uma mensagem de erro

return; // Retorna da função

}

struct Pilha \*temp = topo; // Cria um ponteiro temporário apontando para o topo da pilha

topo = topo->prox; // Atualiza o topo da pilha para apontar para o próximo item

free(temp); // Libera a memória do item removido

}

void mostrarTextoAtual() {

if (topo == NULL) { // Verifica se a pilha está vazia

printf("Texto atual: [vazio]\n"); // Imprime uma mensagem indicando que o texto está vazio

return; // Retorna da função

}

printf("Texto atual: %s\n", topo->texto); // Imprime o texto atual na pilha

}

int main() {

int opcao; // Variável para armazenar a opção do usuário

char textoDigitado[MAX\_TEXT\_LENGTH]; // Array para armazenar o texto digitado pelo usuário

while (1) { // Loop infinito para exibir o menu repetidamente

printf("\n1- Digitar texto\n"); // Opção para digitar texto

printf("2- Desfazer (undo)\n"); // Opção para desfazer a última ação

printf("3- Mostrar texto atual\n"); // Opção para mostrar o texto atual na pilha

printf("4- Sair\n"); // Opção para sair do programa

printf("Opcao? "); // Solicita ao usuário que escolha uma opção

scanf("%d", &opcao); // Lê a opção escolhida pelo usuário

switch (opcao) { // Inicia o bloco switch para lidar com diferentes opções

case 1:

printf("Digite o texto: "); // Solicita ao usuário que digite o texto

scanf(" %[^\n]", textoDigitado); // Lê uma linha de texto incluindo espaços

push(textoDigitado); // Adiciona o texto à pilha

break;

case 2:

pop(); // Desfaz a última ação

break;

case 3:

mostrarTextoAtual(); // Mostra o texto atual na pilha

break;

case 4:

printf("Saindo...\n"); // Mensagem indicando que o programa está sendo encerrado

exit(0); // Sai do programa sem erros

default:

printf("Opcao invalida!\n"); // Mensagem indicando que a opção escolhida é inválida

}

}

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

}

**Print da execução** -> Nesta execução do código foi digitado uma vez o ‘txt’ e depois o ‘txtnovo’. Foi mostrado na tela o topo atual da pilha ‘txtnovo’, retirado esse topo e mostrado o novo topo que foi a primeira string digitada ‘txt’

