Giullio Emmanuel da Cruz Di Gerolamo

RA: 790965

Frequência F4

**Arquivo Pilha.cpp**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

typedef struct

{

char letra;

} Item;

typedef struct st\_celula

{

Item\* item;

struct st\_celula\* proximo;

} Celula;

typedef struct

{

Celula\* topo;

int tamanho;

} Pilha;

int compara(Pilha\*, Pilha\*);

Pilha\* cria();

Item\* POP(Pilha\*);

int PUSH(Item\*, Pilha\*);

int exibe(Pilha\*);

Pilha\* inverte(Pilha\*);

int main(void)

{

Pilha\* p = cria();

Item item;

exibe(p); // vazia

// um loop para carregar uns itens

printf("Insere de 'A' a 'Z'\n");

for (item.letra = 'A'; item.letra <= 'Z'; item.letra += 1) PUSH(&item, p);

exibe(p);

int n = p->tamanho / 2;

printf("Desempilhando %d elementos\n", n);

for (int i = 0; i < n; i += 1)

printf("%c ", (POP(p))->letra);

printf("\n");

printf("Pilha restante\n");

exibe(p);

printf("Inverte a Pilha\n");

Pilha\* inv\_p = inverte(p);

exibe(inv\_p);

printf("Pilha original\n");

exibe(p);

// cria um apilha igual a inv\_p pra comparar

printf("\n\nComparando pilhas:\n\n");

Pilha\* outra = cria();

exibe(outra);

exibe(inv\_p);

if (compara(inv\_p, outra))

printf("As pilhas sao iguais\n");

else

printf("As pilhas sao diferentes\n");

printf("\nCarregando a pilha que estava vazia: \

Insere de 'B' a 'M'\n");

for (item.letra = 'M'; item.letra >= 'B'; item.letra -= 1) PUSH(&item, outra);

printf("\nCompara de novo:\n");

exibe(outra);

exibe(inv\_p);

if (compara(inv\_p, outra))

printf("As pilhas sao iguais\n");

else

printf("As pilhas sao diferentes\n");

printf("\nInsere a letra que faltava e compara de novo:\n");

item.letra = 'A';

PUSH(&item, outra);

exibe(outra);

exibe(inv\_p);

if (compara(inv\_p, outra))

printf("As pilhas sao iguais\n");

else

printf("As pilhas sao diferentes\n");

return 0;

}

int compara(Pilha\* uma, Pilha\* outra)

{ // retorna zero se 'uma' difere de 'outra', ou 1

if (uma->tamanho != outra->tamanho) return 0; // claro

Celula\* pA = uma->topo;

Celula\* pB = outra->topo;

// pA e pB apontam para o topo de cada pilha

// olha um por um e se for diferente ja era

for (int i = 0; i < uma->tamanho; i += 1)

{

if (pA->item->letra != pB->item->letra) return 0;

pA = pA->proximo; pB = pB->proximo; // avanca

}

return 1;

}

Pilha\* cria()

{

Pilha\* nova = (Pilha\*)malloc(sizeof(Pilha));

nova->tamanho = 0;

nova->topo = NULL;

return nova;

};

Item\* POP(Pilha\* pilha)

{

if (pilha->tamanho < 1) return NULL; // vazia

Item\* valor = (Item\*)malloc(sizeof(Item));

\*valor = \*(pilha->topo->item);

Celula\* topo = pilha->topo; // salva para nao perder

pilha->topo = topo->proximo;

pilha->tamanho -= 1; // um a menos;

free(topo->item); // adeus item

free(topo); // adeus celula

return valor;

}

int PUSH(Item\* item, Pilha\* pilha)

{

Celula\* nova = (Celula\*)malloc(sizeof(Celula));

Item\* pItem = (Item\*)malloc(sizeof(Item));

\*(pItem) = \*item; // copia tudo

nova->item = pItem; // o ponteiro

nova->proximo = pilha->topo;

pilha->topo = nova;

pilha->tamanho += 1;

return 0;

} // empilha()

int exibe(Pilha\* pilha)

{

printf("Pilha tem %d elementos\n[", pilha->tamanho);

Celula\* p = pilha->topo;

while (p != NULL)

{

printf("'%c' ", p->item->letra);

p = p->proximo;

};

printf("]\n");

return 0;

};

Pilha\* inverte(Pilha\* origem)

{

Pilha\* destino = cria();

// enquanto tem algo na origem insere na outra

Item\* item = POP(origem);

while (item != NULL)

{

PUSH(item, destino);

item = POP(origem);

}

return destino;

};

**Prints da execução**

Graphical user interface, text

Description automatically generated

A picture containing text, screenshot, computer, indoor

Description automatically generated