código prof

filas_vetor.h

```c
typedef struct {
    int *dados;
    int N, p, u;
} fila;

fila *cria_fila ();
int enfileira (fila *f, int x);
int desenfileira (fila *f, int *y);
void destroi_fila (fila *f);
void imprime_fila (fila *f);
```

filas_vetor.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "filas_vetor.h"

fila *cria_fila () {
    fila *f = malloc (sizeof (fila));
    f->dados = calloc (5, sizeof (int));
    f->p = f->u = 0; // fila vazia
    f->N = 5;
    return f;
}

int redimensiona (fila *f) {
    return 1;
}

int enfileira (fila *f, int x) {
    if ((f->u+1)%f->N == f->p) // fila cheia
        if (redimensiona (f) != 0) return 1;
    f->dados[f->u] = x;
    f->u = (f->u + 1)%f->N;
    return 0;
}

int desenfileira (fila *f, int *y) {
    if (f->p == f->u) return 1;
    *y = f->dados[f->p];
    f->p = (f->p + 1)%f->N;
    return 0;
}

void destroi_fila (fila *f) {
    free (f->dados);
    free (f);
}

void imprime_fila (fila *f) {
    int i;
/*
 ----- -----
| 10 | -30 |
 ----- -----
  p u
 */
    for (i = 0; i < f->N; i++)
        printf (" -----");
    printf ("\n");

    for (i = 0; i < f->N; i++)
        printf ("| %3d ", f->dados[i]);
    printf ("\n");

    for (i = 0; i < f->N; i++)
        printf (" -----");
    printf ("\n");

    if (f->p < f->u) {
        for (i = 0; i < f->p; i++)
            printf ("      ");
        printf ("   p  ");
        for (i = f->p+1; i < f->u; i++)
            printf ("      ");
        printf ("   u\n");
    }
    else if (f->p > f->u) {
        for (i = 0; i < f->u; i++)
            printf ("      ");
        printf ("   u  ");
        for (i = f->u+1; i < f->p; i++)
            printf ("      ");
        printf ("   p\n");
    }
    else {
        for (i = 0; i < f->u; i++)
            printf ("      ");
        printf ("  p u\n");
    }
}
```

filas_lista.h

```c
typedef struct no {
    int dado;
    struct no *prox;
} no;

typedef struct {
    no *le;
} fila;

fila *cria_fila ();
int enfileira (fila *f, int x);
int desenfileira (fila *f, int *y);
void destroi_fila (fila *f);
void imprime_fila (fila *f);
```

filas_lista.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "filas_lista.h"

fila *cria_fila () {
    fila *f = malloc (sizeof (fila));
    f->le = malloc (sizeof (no));
    f->le->prox = f->le;
    return f;
}

int enfileira (fila *f, int x) {
    no *novo = malloc (sizeof (no));
    novo->prox = f->le->prox;
    f->le->prox = novo;
    f->le->dado = x;
    f->le = novo;
    return 0;
}

int desenfileira (fila *f, int *y) {
    no *lixo = f->le->prox;
    if (lixo == f->le) return 1;
    *y = lixo->dado;
    f->le->prox = lixo->prox;
    free (lixo);
    return 0;
}

void destroi_fila (fila *f) {
    int dummy;
    while (desenfileira (f, &dummy) == 0);
    free (f->le);
    free (f);
}

void imprime_fila (fila *f) {
    no *p;
    for (p = f->le->prox; p != f->le; p = p->prox)
        printf (" -----");
    printf ("\n");

    for (p = f->le->prox; p != f->le; p = p->prox)
        printf ("| %3d ", p->dado);
    printf ("|\n");

    for (p = f->le->prox; p != f->le; p = p->prox)
        printf (" -----");
    printf ("\n");

    printf ("   p  ");

    for (p = f->le->prox; p->prox != f->le; p = p->prox)
        printf ("      ");
    printf ("   u\n");
}
```

mainVetor.c
mainLista.c

filas_lista.h

```c
#include <stdio.h>
#include "filas_vetor.h"

int main () {
    int op, valor;
    fila *f = cria_fila ();

    do {
        printf ("MENU\n");
        printf ("1 - Enfileira\n");
        printf ("2 - Desenfileira\n");
        printf ("3 - Imprimir\n");
        printf ("4 - Sair\n");
        printf ("Escolha uma opcao: ");
        scanf ("%d", &op);

        switch (op) {
        case 1:
            printf ("Digite o valor a ser enfileirado: ");
            scanf ("%d", &valor);
            if (enfileira (f, valor) == 0)
                printf ("Elemento enfileirado com sucesso.\n\n");
            else
                printf ("Nao foi possivel enfileirar o elemento.\n\n");
            break;
        case 2:
            if (desenfileira (f, &valor) == 0)
                printf ("Valor desenfileirado: %d\n\n", valor);
            else
                printf ("Fila vazia.\n\n");
            break;
        case 3:
            imprime_fila (f);
            break;
        }
    } while (op != 4);

    destroi_fila (f);

    return 0;
}
```

pilha_vetor.h

```c
typedef struct {
    int *dado;
    int topo, N;
} pilha;

void cria_pilha (pilha *p);
int empilha (pilha *p, int x);
int desempilha (pilha *p, int *y);
void destroi_pilha (pilha *p);
void imprime_pilha (pilha p);
```

pilha_vetor.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "pilha_vetor.h"

void cria_pilha (pilha *p) {
    p->N = 10;
    p->dado = malloc (p->N * sizeof (int));
    p->topo = -1;
}

int empilha (pilha *p, int x) {
    p->topo++;
    if (p->topo == p->N) {
        p->dado = realloc (p->dado, 2*p->N*sizeof (int));
        if (p->dado != NULL) p->N *= 2;
        else return 1;
    }
    p->dado[p->topo] = x;
    return 0;
}

int desempilha (pilha *p, int *y) {
    if (p->topo == -1) return 1;
    *y = p->dado[p->topo];
    p->topo--;
    return 0;
}

void destroi_pilha (pilha *p) {
    free (p->dado);
}

void imprime_pilha (pilha p) {
    printf (" ---\n");
    for (int i = p.N-1; i >= 0; i--)
        if (i <= p.topo)
            printf("|%3d|\n ---\n", p.dado[i]);
        else
            printf("|   |\n ---\n");
    printf ("\n");
}
```

mainVetor.c

```c
#include <stdio.h>
#include "pilha_vetor.h"

int main () {
    int op, valor;
    pilha p;

    cria_pilha (&p);

    do {
        printf ("MENU\n");
        printf ("1 - Empilha\n");
        printf ("2 - Desempilha\n");
        printf ("3 - Imprimir\n");
        printf ("4 - Sair\n");
        printf ("Escolha uma opcao: ");
        scanf ("%d", &op);

        switch (op) {
        case 1:
            printf ("Digite o valor a ser empilhado: ");
            scanf ("%d", &valor);
            if (empilha (&p, valor) == 0)
                printf ("Elemento empilhado com sucesso.\n\n");
            break;
        case 2:
            if (desempilha (&p, &valor) == 0)
                printf ("Valor desempilhado: %d\n\n", valor);
            else
                printf ("Pilha vazia.\n\n");

            break;
        case 3:
            imprime_pilha (p);
            break;
        }
    } while (op != 4);

    destroi_pilha (&p);

    return 0;
}
```

pilha_lista.h

```c
typedef struct no {
    int dado;
    struct no *prox;
} pilha;

pilha *cria_pilha ();
int empilha (pilha *topo, int x);
int desempilha (pilha *topo, int *y);
void destroi_pilha (pilha *topo);
void imprime_pilha (pilha *topo);
```

pilha_lista.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "pilha_lista.h"

pilha *cria_pilha () {
    pilha *topo = malloc (sizeof (pilha));
    topo->prox = NULL;
    return topo;
}

int empilha (pilha *topo, int x) {
    // Inserir no inicio da lista encadeada
    pilha *novo = malloc (sizeof (pilha));
    if (novo == NULL) return 1;
    novo->dado = x;
    novo->prox = topo->prox;
    topo->prox = novo;
    return 0;
}

int desempilha (pilha *topo, int *y) {
    // Remove do inicio da lista
    pilha *lixo = topo->prox;
    if (lixo == NULL) return 1;
    *y = lixo->dado;
    topo->prox = lixo->prox;
    free (lixo);
    return 0;
}

void destroi_pilha (pilha *topo) {
    int dummy;
    while (desempilha (topo, &dummy) == 0);
    free (topo);
}

void imprime_pilha (pilha *topo) {
    printf ("topo\n  |\n  v\n ---\n");
    for (pilha *p = topo->prox; p != NULL; p = p->prox)
        printf("|%3d|\n ---\n", p->dado);
    printf ("\n");
}
```

mainLista.c

```c
#include <stdio.h>
#include "pilha_lista.h"

int main () {
    int op, valor;
    pilha *p = cria_pilha ();

    do {
        printf ("MENU\n");
        printf ("1 - Empilha\n");
        printf ("2 - Desempilha\n");
        printf ("3 - Imprimir\n");
        printf ("4 - Sair\n");
        printf ("Escolha uma opcao: ");
        scanf ("%d", &op);

        switch (op) {
        case 1:
            printf ("Digite o valor a ser empilhado: ");
            scanf ("%d", &valor);
            if (empilha (p, valor) == 0)
                printf ("Elemento empilhado com sucesso.\n\n");
            break;
        case 2:
            if (desempilha (p, &valor) == 0)
                printf ("Valor desempilhado: %d\n\n", valor);
            else
                printf ("Pilha vazia.\n\n");

            break;
        case 3:
            imprime_pilha (p);
            break;
        }
    } while (op != 4);

    destroi_pilha (p);

    return 0;
}
```

Pilhas - remoção de elementos

Numa pilha, a remoção de elementos é chamada **desempilhamento**. Sua tarefa nesse exercício é implementar essa operação usando *vetores*. Para tanto, você deve submeter a função

```
int desempilha (pilha *p, int *y);
```

que deve

- remover um elemento da pilha `p` e salvá-lo em `y` e
- retornar 1 se a remoção foi bem sucedida, e 0 caso contrário.

Espera-se que `pilha` seja uma `struct` da forma

```
typedef struct pilha {
  int *dados;
  int N, topo;
} pilha;
```

de tal forma que `topo` indica a **primeira posição livre** da pilha (ou seja, na inicialização vale `0` e quanto a pilha estiver cheia, vale `N`).

O arquivo a ser submetido deve incluir apenas

1. os `#include` necessários para a execução do seu código,
2. a declaração da estrutura necessária e
3. a função solicitada.

Auth

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct pilha {
  int *dados;
  int N, topo;
} pilha;

int desempilha (pilha *p, int *y){
if (p->topo==0){
return 0;
}
else{
    p->topo--;
    *y=p->dados[p->topo];
    return 1;
}
}
```

Pilhas - inserção de elementos

Numa pilha, a inserção de elementos é chamada **empilhamento**. Sua tarefa nesse exercício é implementar essa operação usando *vetores*. Para tanto, você deve submeter a função

```
int empilha (pilha *p, int x);
```

que deve

- inserir o elemento `x` na pilha `p` e
- retornar 1 se a inserção foi bem sucedida, e 0 caso contrário.

Espera-se que `pilha` seja uma `struct` da forma

```
typedef struct pilha {
  int *dados;
  int N, topo;
} pilha;
```

de tal forma que `topo` indica a **primeira posição livre** da pilha (ou seja, na inicialização vale `0` e quanto a pilha estiver cheia, vale `N`).

Observação: se na inserção, o elemento não couber na pilha, o vetor `dados` deve ser redimensionado.

O arquivo a ser submetido deve incluir apenas

1. os `#include` necessários para a execução do seu código,
2. a declaração da estrutura necessária e
3. a função solicitada.

Auth

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct pilha {
  int *dados;
  int N, topo;
} pilha;

int empilha (pilha *p, int x){
    if(p->topo==p->N){
        p->dados=realloc(p->dados,2*p->N*sizeof(int));
        if(p->dados==NULL) return 0;
        p->N*=2;
    }
    p->dados[p->topo]=x;
    p->topo++;
    return 1;
}
```

Pilhas - remoção de elementos

Numa pilha, a remoção de elementos é chamada **desempilhamento**. Sua tarefa nesse exercício é implementar essa operação usando *listas encadeadas*. Para tanto, você deve submeter a função

```
int desempilha (celula *p, int *y);
```

que deve

- remover um elemento da pilha `p` e salvá-lo em `y` e
- retornar 1 se a remoção foi bem sucedida, e 0 caso contrário.

Espera-se que `celula` seja uma `struct` da forma

```
typedef struct celula {
  int dado;
  struct celula *prox;
} celula;
```

e que a pilha seja representada por uma lista encadeada com nó cabeça.

O arquivo a ser submetido deve incluir apenas

1. os `#include` necessários para a execução do seu código,
2. a declaração da estrutura necessária e
3. a função solicitada.

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct celula {
  int dado;
  struct celula *prox;
} celula;

int desempilha (celula *p, int *y){
celula *lixo = p -> prox;
if (lixo == NULL) return 0;
*y = lixo -> dado;
p -> prox = lixo -> prox;
free (lixo);
return 1;
}
```

Pilhas - inserção de elementos

Numa pilha, a inserção de elementos é chamada **empilhamento**. Sua tarefa nesse exercício é implementar essa operação usando *listas encadeadas*. Para tanto, você deve submeter a função

```
int empilha (celula *p, int x);
```

que deve

- inserir o elemento `x` na pilha `p` e
- retornar 1 se a inserção foi bem sucedida, e 0 caso contrário.

Espera-se que `celula` seja uma `struct` da forma

```
typedef struct celula {
  int dado;
  struct celula *prox;
} celula;
```

e que a pilha seja representada por uma lista encadeada com nó cabeça.

O arquivo a ser submetido deve incluir apenas

1. os `#include` necessários para a execução do seu código,
2. a declaração da estrutura necessária e
3. a função solicitada.

```c
#include<stdio.h>

typedef struct celula {
  int dado;
  struct celula *prox;
} celula;

int empilha (celula *p, int x){
celula *extra=malloc(sizeof(celula));
if (extra==NULL){
  return 0;
}
extra->dado=x;
extra->prox=p->prox;
p->prox=extra;

return 1;
}
```

Filas - remoção de elementos

Numa fila, a remoção de elementos é chamada **desenfileiramento**. Sua tarefa nesse exercício é implementar essa operação usando *vetores*. Para tanto, você deve submeter a função

```
int desenfileira (fila *f, int *y);
```

que deve

- remover um elemento da fila `f` e salvá-lo em `y`,
- retornar 1 se a remoção foi bem sucedida, e 0 caso contrário.

Espera-se que `fila` seja uma `struct` da forma

```
typedef struct fila {
  int *dados;
  int N, p, u;
} fila;
```

Observação:

- Você deve considerar uma fila **circular**.

O arquivo a ser submetido deve incluir apenas

1. os `#include` necessários para a execução do seu código,
2. a declaração da estrutura necessária e
3. a função solicitada.

Autl

```c
#include <stdlib.h>
#include <stdio.h>

typedef struct fila {
  int *dados;
  int N, p, u;
} fila;

int desenfileira (fila *f, int *y) {
    if (f->p == f->u) return 1;
    *y = f->dados[f->p];
    f->p = (f->p + 1)%f->N;
    return 0;
}
```

Filas - inserção de elementos

Numa fila, a inserção de elementos é chamada **enfileiramento**. Sua tarefa nesse exercício é implementar essa operação usando *vetores*. Para tanto, você deve submeter a função

```
int enfileira (fila *f, int x);
```

que deve

- inserir o elemento `x` na fila `f` e
- retornar 1 se a inserção foi bem sucedida, e 0 caso contrário.

Espera-se que `fila` seja uma `struct` da forma

```
typedef struct fila {
  int *dados;
  int N, p, u;
} fila;
```

Observações:

- Você deve considerar uma fila **circular**.
- Se na inserção, o elemento não couber na fila, o vetor `dados` deve ser redimensionado.

O arquivo a ser submetido deve incluir apenas

1. os `#include` necessários para a execução do seu código,
2. a declaração da estrutura necessária e
3. a função solicitada.

Auti

```c

#include <stdlib.h>
#include <stdio.h>

typedef struct fila {
  int *dados;
  int N, p, u;
} fila;



int redimensiona (fila *f) {
    // retorna 0 se der certo, 1 se der errado
    f->dados = realloc (f->dados, 2*f->N*sizeof (int));
    if (f->dados == NULL) return 1;
    for (int i = f->N; i < 2*f->N; i++) f->dados[i] = 0;
    if (f->p != 0) {
        if (f->N - f->p < f->u) {
            for (int i = f->N-1; i >= f->p; i--)
                f->dados[i+f->N] = f->dados[i];
            f->p += f->N;
        }
        else {
            for (int i = 0; i < f->u; i++)
                f->dados[i+f->N] = f->dados[i];
            f->u += f->N;
        }
    }
    f->N *= 2;
    return 0;
}

int enfileira (fila *f, int x) {
    if ((f->u+1)%f->N == f->p) // fila cheia
        if (redimensiona (f) != 0) return 1;
    f->dados[f->u] = x;
    f->u = (f->u + 1)%f->N;
    return 0;
}
```

Filas - remoção de elementos

Numa fila, a remoção de elementos é chamada **desenfileiramento**. Sua tarefa nesse exercício é implementar essa operação usando *listas encadeadas*. Para tanto, você deve submeter a função

```
int desenfileira (celula *f, int *y);
```

que deve

- remover um elemento da fila `f` e salvá-lo em `y`,
- retornar 1 se a remoção foi bem sucedida, e 0 caso contrário.

Espera-se que `celula` seja uma `struct` da forma

```
typedef struct celula {
  int dado;
  struct celula *prox;
} celula;
```

e que a fila seja representada por uma lista encadeada com nó cabeça.

O arquivo a ser submetido deve incluir apenas

1. os `#include` necessários para a execução do seu código,
2. a declaração da estrutura necessária e
3. a função solicitada.

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct celula {
  int dado;
  struct celula *prox;
} celula;

int desenfileira (celula *f, int *y) {
celula *lixo = f->prox;
if (lixo == NULL) {
  return 0; //código de erro pois não podemos desinfileirar NULL
}
//como eu entrou nesse if vamos poder remover o elemento da lista
*y = lixo->dado; //vamos armazenar na variavel y o conteudo a ser removido
f->prox = lixo->prox; //reajustando a ordem da fila para termos
//um proximo elemento como elemento a ser desinfileirado
free(lixo); //removendo da memória o conteudo
return 1; //tivemos um desinfileiramento bem sucedido
}
```

Filas - inserção de elementos

Numa fila, a inserção de elementos é chamada **enfileiramento**. Sua tarefa nesse exercício é implementar essa operação usando *listas encadeadas*. Para tanto, você deve submeter a função

```
int enfileira (celula *f, int x);
```

que deve

- inserir o elemento `x` na fila encabeçada por `f` e
- retornar o ponteiro para o novo nó cabeça, se a inserção foi bem sucedida, ou `NULL` caso contrário.

Espera-se que `celula` seja uma `struct` da forma

```
typedef struct celula {
  int dado;
  struct celula *prox;
} celula;
```

O arquivo a ser submetido deve incluir apenas

1. os `#include` necessários para a execução do seu código,
2. a declaração da estrutura necessária e
3. a função solicitada.

```c
#include <stdio.h>
#include <stdlib.h>

//estrutura base da celula apresentada
typedef struct celula {
  int dado;
  struct celula *prox;
} celula;

//com essa estrutura base da celula poderemos fazer a função de enfileiramento
celula *enfileira (celula *f, int x) { //estrutura base da função dada no exercício
    //ao se criar uma fila com listas precisamos sempre fazer o ajuste no tamanho da memória
    celula*novo = malloc (sizeof(celula)); //abrindo espaço para implementação da lista
    //agora poderemos fazer o enfileiramento utilizando como referência o nó cabeça como ponto d
    novo->prox=f->prox; //inserindo o proximo elemento após o nó cabeça "f"
    f->prox=novo; //pegando a proxima celula como novo nó cabeça
    f->dado=x; //armazenando o conteúdo no novo nó cabeça
    return novo;//tivemos uma inserção bem sucedida e por isso retornamos o novo nó cabeça
}
//caso não seja bem sucedida essa inserção será retornado um NULL por conta da biblioteca "stdli
```

Expressões matemáticas 1

Usualmente, estamos acostumados a ver expressões matemáticas. Uma expressão matemática é um arranjo de termos que seguem o seguinte padrão

`operando operador operando`

sendo o operador algum dos operadores matemáticos, por exemplo, `+`, `-`, `*` e `/`, e o operando um número ou outro termo do mesmo formato. Por exemplo,

`A+B`

e

`(A+B)*C`

são exemplos de expressões matemáticas.

Sabemos que nas expressões matemáticas, os operadores possuem uma ordem de prioridade. Não obstante, há formas de se alterar essa ordem de prioridade, usando-se chaves, colchetes e parênteses. Deste modo, as expressões que estiverem dentro desses modificadores serão avaliadas primeiro.

A regra de ouro é a seguinte: sempre que abrimos um parêntese, devemos fechá-lo depois. O mesmo vale para chaves e colchetes.

Tarefa

Sua tarefa é, dada uma expressão matemática, verificar se a expressão está corretamente *parentizada*, ou seja, se toda *chave*, *colchete* e *parêntese* abertos são fechados posteriormente.

Entrada

A entrada é composta por uma única linha, contendo uma expressão matemática com, no máximo, 500 caracteres.

Saída

A saída é composta por uma única linha contendo "sim" se a expressão estiver corretamente parentizada, ou "nao" caso contrário.

Exemplo de Entrada 1

`A+B`

Exemplo de Saída 1

`sim`

Exemplo de Entrada 2

`{X-Y+[Z*(A^2-C)/(A+B)}`

Exemplo de Saída 2

`nao`

Exemplo de Entrada 3

`(A+B)/C`

Exemplo de Saída 3

`sim`

Exemplo de Entrada 4

`{(Z-A]/(K+10)}+C`

Exemplo de Saída 4

`nao`

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct cel
{
    int dado;
    struct cel *prox;
} cel;

cel *cria_pilha()
{
    cel *topo = malloc(sizeof(cel));
    topo->prox = NULL;
    return topo;
}

void empilha(cel *p, int x)
{
    cel *ponteiro = malloc(sizeof(cel));
    ponteiro->dado = x;
    ponteiro->prox = p->prox;
    p->prox = ponteiro;
}

int desempilha(cel *p)
{
    char desempilhado;
    cel *lixo = p->prox;

    if (lixo == NULL)
    {
        return 0;
    }

    desempilhado = lixo->dado;
    p->prox = lixo->prox;
    return desempilhado;
    free(lixo);
}

int main()
{
    int i = 0, balanceado = 0, abre = 0, objeto = 0, fecha = 0;
    char exp[500];
    cel *p = cria_pilha();

    scanf("%[^\n]", exp);

    for (i = 0; i < strlen(exp); i++)
    {
        if (exp[i] == '(')
        {
            empilha(p, exp[i]);
            abre++;
        }

        if (exp[i] == '[')
        {
            empilha(p, exp[i]);
            abre++;
        }

        if (exp[i] == '{')
        {
            empilha(p, exp[i]);
            abre++;
        }

        if (exp[i] == ')')
        {
            fecha++;
            objeto = desempilha(p);
            if (objeto == 0)
            {
                balanceado++;
            }
            if (objeto == 40)
            {
                balanceado++;
            }
        }

        if (exp[i] == ']')
        {
            fecha++;
            objeto = desempilha(p);
            if (objeto == 0)
            {
                balanceado++;
            }
            if (objeto == 91)
            {
                balanceado++;
            }
        }

        if (exp[i] == '}')
        {
            fecha++;
            objeto = desempilha(p);
            if (objeto == 0)
            {
                balanceado++;
            }
            if (objeto == 123)
            {
                balanceado++;
            }
        }
    }

    if (balanceado == abre && fecha == abre)
    {
        printf("sim\n");
    }
    else
    {
        printf("nao\n");
    }

    return 0;
}
```

Cartas

Considere uma pilha de n cartas enumeradas de 1 até n, sendo que a carta 1 está no topo e a carta n está na base. A seguinte operação é realizada enquanto tiver duas ou mais cartas na pilha:

> Jogue fora a carta do topo e mova a próxima carta (que ficou no topo) para a base da pilha.

Sua tarefa é escrever um programa que leia o valor de n e imprima na tela a sequência de cartas descartadas e a última carta da pilha.

Entrada

A entrada é composta por um único valor inteiro n ($n \geq 2$).

Saída

Observe os exemplos abaixo.

Exemplo de Entrada 1

```
6
```

Exemplo de Saída 1

```
Cartas descartadas: 1, 3, 5, 2, 6
Carta restante: 4
```

Exemplo de Entrada 2

```
7
```

Exemplo de Saída 2

```
Cartas descartadas: 1, 3, 5, 7, 4, 2
Carta restante: 6
```

Exemplo de Entrada 3

```
12
```

Exemplo de Saída 3

```
Cartas descartadas: 1, 3, 5, 7, 9, 11, 2, 6, 10, 4, 12
Carta restante: 8
```

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct cel
{
    int dado;
    struct cel *prox;
} cel;

cel *cria_pilha()
{
    cel *novo = malloc(sizeof(cel));
    novo->prox = novo;
    return novo;
}

cel *empilhar(cel *pilha, int x)
{
    cel *ponteiro = malloc(sizeof(cel));
    ponteiro->prox = pilha->prox;
    pilha->prox = ponteiro;
    pilha->dado = x;
    return ponteiro;
}

int desempilhar(cel *f)
{
    int numero = 0;
    cel *retirar = f->prox;
    if (f->prox == f)
    {
        return 0;
    }
    else
    {
        numero = retirar->dado;
        f->prox = retirar->prox;
        free(retirar);
        return numero;
    }
}

int main()
{

    int cartas, num = 0;
    int i = 0, contador = 0;
    cel *pilha = cria_pilha();

    scanf("%d", &cartas);

    for (i = 1; i <= cartas; i++)
    {
        pilha = empilhar(pilha, i);
    }

    printf("Cartas descartadas:");
    contador = 1;

    for (contador = cartas; contador >= 1; contador--)
    {
        if (contador > 2)
        {
            printf(" %d", desempilhar(pilha));
            printf(",");
            num = desempilhar(pilha);
            pilha = empilhar(pilha, num);
        }

        if (contador == 2)
        {
            printf(" %d\n", desempilhar(pilha));
        }

        if (contador == 1)
        {
            printf("Carta restante:");
            printf(" %d\n", desempilhar(pilha));
        }
    }

    return 0;
}
```

Desfazer no Editor

Você foi contratado para trabalhar na Fábrica de Geração de Aplicativos (FGA). Todos na FGA estão focados no desenvolvimento de um novo editor de texto, versátil, que funciona como IDE, editor de linguagens de marcação e também um editor de texto.

O gerente de projetos te deu a tarefa de desenvolver a Estrutura de Desfazer do Aplicativo (EDA). O objetivo da EDA é prover, para o editor, o famoso `ctrl+z`, ou seja, a funcionalidade de desfazer certas ações.

A demanda é a seguinte: o editor de texto, de tempos em tempos, enviará ao seu módulo EDA o conteúdo que havia em determinado trecho do texto antes das alterações mais recentes do usuário. Caso o usuário deseje desfazer alguma ação, o editor enviará ao seu aplicativo o comando `desfazer`, e você deve devolver a alteração mais recente do usuário que você tem armazenada.

Nessa primeira versão do módulo EDA, não há limite para a quantidade de `desfazer` que o usuário pode solicitar.

Entrada

A entrada é composta por várias linhas, sendo que cada linha pode conter:

- `inserir STR`, sendo `STR` a alteração mais recente que o usuário efetuou no editor, de tamanho máximo 100.
- `desfazer`, caso em que você deve devolver a última alteração do usuário no editor. Caso não haja nenhuma alteração a ser devolvida, você deve devolver a string `NULL`.

A entrada termina com `EOF`.

Saída

Observe o exemplo abaixo.

Exemplo de Entrada 1

```
desfazer
inserir Era uma vez
inserir uma terra muito
inserir distante
desfazer
inserir em que tudo era muito legal.
inserir Mas ai, de
inserir repente
desfazer
desfazer
desfazer
```

Exemplo de Saída 1

```
NULL
distante
repente
Mas ai, de
em que tudo era muito legal.
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct no
{
    char *palavra;
    struct no *prox;
} no;

no *cria_pilha()
{
    no *topo;
    topo = malloc(sizeof(no));
    topo->prox = NULL;
    return topo;
}

int empilha(no *topo, char *str)
{
    no *novo = malloc(sizeof(no));

    if (novo == NULL)
        return 1;

    novo->palavra = str;
    novo->prox = topo->prox;
    topo->prox = novo;

    return 0;
}

int desempilha(no *topo)
{
    no *lixo = topo->prox;
    if (lixo == NULL)
    {
        printf("NULL\n");
        return 1;
    }
    printf("%s\n", lixo->palavra);
    topo->prox = lixo->prox;
    free(lixo);
    return 0;
}

int op(char *s)
{
    char *insere = "inserir";
    char *desfaz = "desfazer";

    int inserir;
    int desfazer;

    inserir = strcmp(s, insere);
    desfazer = strcmp(s, desfaz);

    if (inserir == 0)
    {
        return 1;
    }
    else if (desfazer == 0)
    {
        return 2;
    }
    else
    {
        return 0;
    }
}

int main()
{
    no *topo = cria_pilha();
    char a[10] = "";
    int opcao;

    while (scanf("%s", a) != EOF)
    {
        opcao = op(a);
        if (opcao == 1)
        {
            char *str = calloc(101, sizeof(char));
            scanf("%[^\n]s", str);
            empilha(topo, str);
        }
        else if (opcao == 2)
        {
            desempilha(topo);
        }
        else
        {
            printf("Operacao invalida\n");
        }
    }

    return 0;
}
```