Pilha (Dinâmica)

LIFO - Last in First out

Pilha
- push(e)
- pop

Operações:

- Empilhar ou push (e) : recebe o parâmetro e.
- Desempilhar ou pop : remove quem esta no topo.

```
#include<stdio.h>
#include<malloc.h>

typedef struct temp{
	int data;
	struct temp *nextEl;
} stack;
stack *top, *newEl;

void push(int e){
	newEl = (stack*)malloc(sizeof(stack));
	newEl->data = e;

	if(top == NULL) newEl->nextEl = NULL;
	else newEl->nextEl = top;

	top = newEl;
}

void pop(){
	if(top == NULL) return;
	else {
	newEl = top;
	top = top -> nextEl;
	free(newEl);
	}
}

void printElements(){
	if(top==NULL)return;

	newEl = top;
	printf("\n Topo \n |\n V");

	while(newEl!=NULL){
	printf("\n %d \n |\n V",newEl->data);
	newEl= newEl -> nextEl;
	}
	printf("\n NULL");
}
```

```
int main (){
	top = NULL;
	int op, n;

	do {
	printf("\n 1- Empilhar \t 2- Desempilhar \t 3- Listar \t 4- Sair");
	printf("\n");
	scanf("%d", &op);

	switch(op){
		case 1:
		printf("\n data:");
		scanf("%d", &n);
		push(n);
		break;
		case 2:
		pop();
		break;
		case 3:
		printElements();
		break;
		case 4:
		exit(0);
		default:
		printf("\n Operacao invalida!");
	}
	}while(op!=4);
}
```

Pilha vazia (inserção)

```
top = null // topo começa nulo

novo = (pilha*)malloc(sizeof(stack)) // aloca espaço na memória
novo -> dado = valor // armazena valor

novo -> prox = null // proximo elemento é nulo, pois é o primeiro elemento
topo = novo // novo elemento sempre esta no topo
```

Pilha com elemento (inserção)

```
novo = (pilha*)malloc(sizeof(stack))
novo -> dado = valor

novo -> prox = topo // aponta para o elemento que esta no topo atual para não perder sua referencia
topo = novo // coloca o novo elemento no topo
```

Remover pilha

```
novo = topo // aponta o novo para o topo
topo = topo -> prox  // aponta o topo para o elemento “abaixo”
free(novo) // libera a memoria
```

Fila(Dinâmica)

FIFO - First in First out

Inicio->[ ]->[ ]->[ ]->Null

```c
#include<stdio.h>
#include<malloc.h>
typedef struct temp{
    int data;
    struct temp *nextEl;
} queue;
queue *front, *rear, *helper;

void enqueue(int e){
    helper = (queue*)malloc(sizeof(queue));
    helper->data = e;

    if(front == NULL) front = helper;
    else rear->nextEl = helper;

    rear = helper;
    helper->nextEl = NULL;
}

void dequeue(){
    if(front == NULL) return;
    else {
        helper = front;
        front = front -> nextEl;
        free(helper);
    }
}

void printElements(){
    if(front==NULL)return;

    helper = front;
    printf("\n Inicio");

    while(helper!=NULL){
        printf("->[%d]",helper->data);
        helper= helper -> nextEl;
    }
    printf("->NULL");
}
```

```c
int main (){
    front = NULL;
    int op, n;

    do {
        printf("\n 1- Enfileirar \t 2- Desenfileirar \t 3- Listar \t 4- Sair");
        printf("\n");
        scanf("%d", &op);

        switch(op){
            case 1:
                printf("\n valor:");
                scanf("%d", &n);
                enqueue(n);
                break;
            case 2:
                dequeue();
                break;
            case 3:
                printElements();
                break;
            case 4:
                exit(0);
            default:
                printf("\n Operacao invalida!");
        }
    }while(op!=4);
}
```

Inserir em fila vazia:

```
inicio = null

aux = (pilha*)malloc(sizeof(stack))
aux -> dado = n

inicio = aux // inicio da fila é o primeiro elemento
fim = aux
aux -> prox = null
```

Inserir em fila com elemento:

```
aux = (pilha*)malloc(sizeof(stack))
aux -> dado = n

fim -> prox = aux   // conecta o novo elemento na fila
fim = aux
aux -> prox = null
```

Remover elemento:

```
aux = inicio // referencia o primeiro elemento que será removido
inicio = inicio->prox      // inicio se torna o elemento posterior
free(aux)
```