PILHAS

Estrutura de Dados

Pilhas / Stack

- Uma das estruturas de dados mais simples é a pilha. Possivelmente por essa razão, é a estrutura de dados mais utilizada em programação, sendo inclusive implementada diretamente pelo hardware da maioria das máquinas modernas.
- A idéia fundamental da pilha é que todo o acesso a seus elementos é feito através do seu topo.
- Assim, quando um elemento novo é introduzido na pilha, passa a ser o elemento do topo, e o único elemento que pode ser removido da pilha é o do topo. Isto faz com que os elementos da pilha sejam retirados na ordem inversa à ordem em que foram introduzidos: o primeiro que sai é o último que entrou
  - **LIFO – last in, first out – é usada para descrever esta estratégia**.

Pilhas / Stack

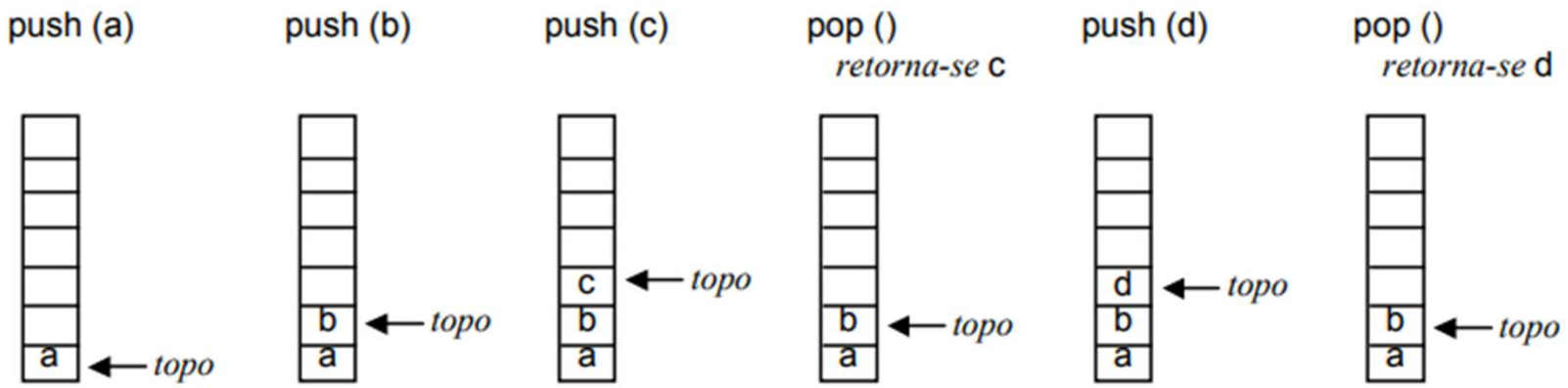
- Para entendermos o funcionamento de uma estrutura de pilha, podemos fazer uma analogia com uma pilha de pratos.
- Se quisermos adicionar um prato na pilha, o colocamos no topo.
- Para pegar um prato da pilha, retiramos o do topo. Assim, temos que retirar o prato do topo para ter acesso ao próximo prato.
- A estrutura de pilha funciona de maneira análoga. Cada novo elemento é inserido no topo e só temos acesso ao elemento do topo da pilha.

Pilhas / Stack

- São exemplos de uso de pilha em um sistema:
  - Funções recursivas em compiladores;
  - Mecanismo de desfazer/refazer dos editores de texto;
  - Navegação entre páginas Web;

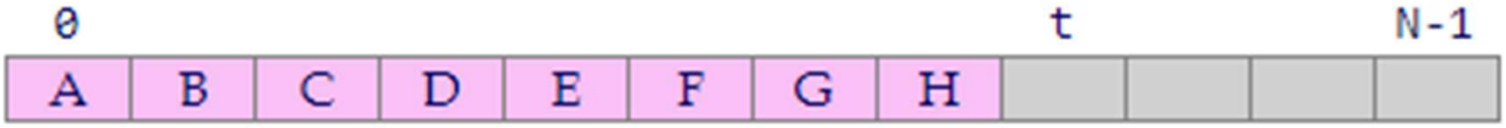
Pilhas / Stack

- Existem duas operações básicas que devem ser implementadas numa estrutura de pilha:
  - Push (empilhar)
  - Pop (desempilhar)



Implementação em um vetor

- Suponha que nossa pilha está armazenada em um vetor pilha[0..N-1]. (A natureza dos elementos do vetor é irrelevante: eles podem ser inteiros, bytes, ponteiros, etc.) Digamos que a parte do vetor ocupada pela pilha é
  - pilha[0..t-1] .
- O índice t indica a primeira posição vaga da pilha e t-1 é o índice do topo da pilha. A pilha está vazia se t vale 0 e cheia se t vale N. No exemplo da figura, os caracteres A, B, … , H foram inseridos na pilha nessa ordem:



Implementação em um vetor

- Para remover, ou tirar, um elemento da pilha, essa operação é conhecida como POP:
  - x = pilha[--t];
  - Isso equivale ao par de instruções t -= 1; x = pilha[t];
- Para inserir, ou colocar, um objeto y na pilha — a operação é conhecida como PUSH:
  - pilha[t++] = y;

Implementação em um vetor

- Todas as operações em uma pilha podem ser imaginadas as cartas de um baralho:
  - criação da pilha (informar a capacidade no caso de implementação sequencial - vetor);
  - empilhar (push) - o elemento é o parâmetro nesta operação;
  - desempilhar (pop);
  - mostrar o topo;
  - verificar se a pilha está vazia (isEmpty);
  - verificar se a pilha está cheia (isFull - implementação sequencial - vetor).

Exemplo 1 – Vetor Fixo

```c
#include <stdio.h>
void push(int *v, int *t);
void pop(int *v, int *t);
void top(int *v, int *t);
int isEmpty(int *t);
int isFull(int *v, int *t);
int max = 20;

int main()
{
    int pilha[max];
    int *p = pilha;
    int t = -1;
    push(p, &t);
    top(p, &t);
    push(p, &t);
    push(p, &t);
    top(p, &t);
    pop(p, &t);
    top(p, &t);
    return 0;
}
```

```c
void push(int *v, int *t){
    int aux;
    printf("Informe o valor:");
    scanf("%i", &aux);
    (*t)++; //isfull
    if(isFull(v, t)) {
        printf("Pilha cheia !!! \n");
    } else {
        v[(*t)] = aux;
    }
}

void pop(int *v, int *t){
    if(isEmpty(t)) {
        printf("Pilha vazia !!! \n");
    } else {
        (*t)--;
    }
}
```

Exemplo 1 – Vetor Fixo

```
void top(int *v, int *t){
    if(isEmpty(t)) {
        printf("Pilha vazia !!! \n");
    } else {
        printf("O topo = %i \n", v[(*t)]);
    }
}

int isEmpty(int *t){
    if((*t)<0) {
        return 1;
    }  else {
        return 0;
    }
}
```

```
int isFull(int *v, int *t){
    if(max<(*t)) {
        return 1;
    }  else {
        return 0;
    }
}
```

Implementação de pilha com lista

- Quando o número máximo de elementos que serão armazenados na pilha não é conhecido, devemos implementar a pilha usando uma estrutura de dados dinâmica, no caso, empregando uma lista encadeada.
- Os elementos são armazenados na lista e a pilha pode ser representada simplesmente por um ponteiro para o primeiro nó da lista.

Implementação de pilha com lista

- O nó da lista para armazenar valores reais pode ser dado por:

```
struct item {
 float info;
 struct no* prox;
};
typedef struct item Item;
```

- A estrutura da pilha é então simplesmente:

```
struct Pilha {
        Item* prim;
};
```

Implementação de pilha com lista

- A função cria aloca a estrutura da pilha e inicializa a lista como sendo vazia.

```
Pilha* criar()
{
    Pilha* p = (Pilha*)
    malloc(sizeof(Pilha));
    p->prim = NULL;
    return p;
}
```

Implementação de pilha com lista

- O primeiro elemento da lista representa o topo da pilha.
- Cada novo elemento é inserido no início da lista e, conseqüentemente, sempre que solicitado, retiramos o elemento também do início da lista. Desta forma, precisamos de duas funções auxiliares da lista: para inserir no início e para remover do início.
- Ambas as funções retornam o novo primeiro nó da lista.

Implementação de pilha com lista

```
/* função auxiliar: insere no início */
Item* ins_ini(Item *l, float v)
{
 Item* p = (Item*) malloc(sizeof(No));
 p->info = v;
 p->prox = l;
 return p;
}
/* função auxiliar: retira do início */
Item* ret_ini(Item* l)
{
 Item* p = l->prox;
 free(l);
 return p;
}
```

Implementação de pilha com lista

```
//As funções que manipulam a pilha fazem uso dessas funções de lista:
void push(Pilha* p, float v)
{
	p->prim = ins_ini(p->prim,v);
}
float pop(Pilha* p)
{
	float v;
	if(vazia(p)) {
		printf("Pilha vazia.\n");
		exit(1); /* aborta programa */
	}
	v = p->prim->info;
	p->prim = ret_ini(p->prim);
	return v;
}
```

Implementação de pilha com lista

A pilha estará vazia se a lista estiver vazia:

```
int vazia (Pilha* p)
{
      return (p->prim==NULL);
}
```

Implementação de pilha com lista

Por fim, a função que libera a pilha deve antes liberar todos os elementos da lista.

```
void libera (Pilha* p)
{
 Item* q = p->prim;
 while (q!=NULL) {
        Item* t = q->prox;
        free(q);
        q = t;
 }
 free(p);
}
```

Implementação de pilha com lista

- A rigor, pela definição da estrutura de pilha, só temos acesso ao elemento do topo.
- No entanto, para testar o código, pode ser útil implementarmos uma função que imprima os valores armazenados na pilha.

```
void imprime(Pilha* p)
{
    Item* q;
     for (q=p->prim; q!=NULL; q=q->prox)
         printf("%f\n",q->info);
}
```