1.0- Listas lineares

1.1- Conceitos

A maneira mais básica de se agrupar dados é a lista. Matematicamente, uma lista é um conjunto L (e1, e2, e3, . . . , en) com as seguintes propriedades para n > 0:
- e1 é o primeiro elemento de L;
- en é o último elemento de L;
- um elemento ek é precedido pelo elemento ek-1 e seguido por ek+1.

Caso n = 0, dizemos que a lista é vazia. A ideia de lista é a unidimensionalidade dos elementos, ou seja, temos sempre um elemento a frente de outro. Com este conceito podemos afirmar com certeza absoluta que a lista tem um início e fim bem determinados.

1.2- Tipos de listas

Considerando as operações de inserção, remoção e pesquisa de elementos restritas às extremidades da lista, temos seguintes tipos especiais de listas:

- Pilha: inserções, remoções e pesquisa de elementos são realizadas a partir de uma única extremidade da lista, ou seja, o último elemento que entrou na lista, será o primeiro elemento a sair.
- Fila: as inserções são realizadas em um extremo e as remoções e pesquisa são realizadas no outro extremo, ou seja, o primeiro elemento que entrou na lista, será o primeiro elemento a sair.

2.0 - PILHAS

2.1 - Introdução

São estruturas de dados do tipo LIFO (last-in first-out), onde o último elemento a ser inserido, será o primeiro a ser retirado. Assim, uma pilha permite acesso a apenas um item de dados - o último inserido. Para processar o penúltimo item inserido, deve-se remover o último.
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;
- etc.

2.2 -Definição:

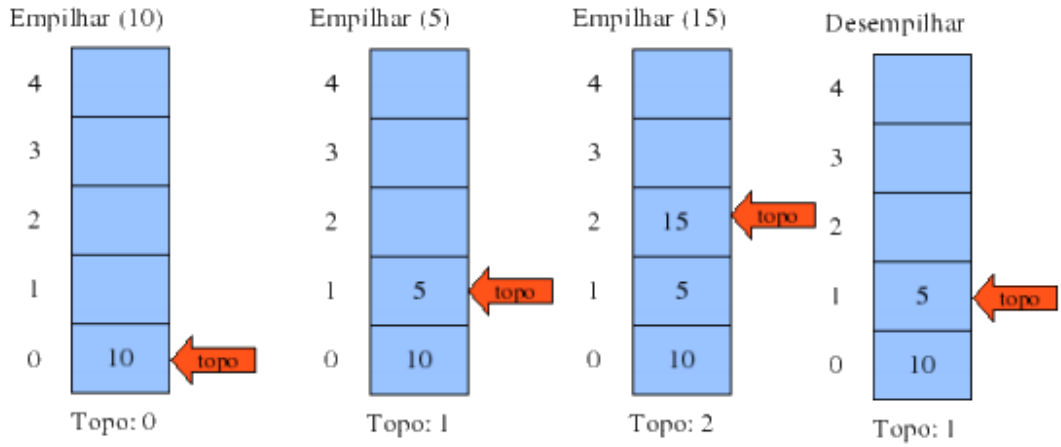
Dada uma pilha P=(a(1), a(2), ..., a(n)), dizemos que a(1) é o elemento da base da pilha; a(n) é o elemento topo da pilha; e a(i+1) está acima de a(i).

2.3- Exemplo:

Numa pilha, a manipulação dos elementos é realizada em apenas uma das extremidades, chamada de topo, em oposição a outra extremidade, chamada de base. Todas as operações em uma pilha podem ser

imaginadas como as que ocorre numa pilha de pratos em um restaurante ou como num jogo com as cartas de um baralho.

Supondo uma pilha com capacidade para 5 elementos (5 nós):



A implementação de pilhas pode ser realizada através de vetor (alocação do espaço de memória para os elementos é contígua) ou através de listas encadeadas.

2.4 - Pilhas sequenciais

A implementação de pilhas sequenciais, em geral as listas são implementadas através de vetores de tamanho fixo. Este tamanho determinará o número máximo de elementos que poderão estar na pilha ao mesmo tempo. É necessário um inteiro para armazenar o valor da posição do vetor aonde encontra-se o topo da pilha.

Definição da Estrutura de Dados:

```
#define MAX 50
typedef struct {
        int n;
        float vet[MAX];
} Pilha;
```

a) A função para criar a pilha aloca dinamicamente essa estrutura e inicializa a pilha como sendo vazia, isto é, com o número de elementos igual a zero

```
Pilha* cria(void)
{
        Pilha* p;
        p = (Pilha*) malloc(sizeof(Pilha));
        p->n = 0;/*Inicializa com zero elementos*/
        return p;
}
```

b) Para inserir um elemento na pilha, usamos a próxima posição livre do vetor. Devemos ainda assegurar que exista espaço para a inserção do novo elemento, tendo em vista que trata-se de um vetor com dimensão fixa.

```
void push(Pilha* p, float v)
{
	if(p->n==MAX){
		printf("Capacidade da pilha estourou.\n");
		exit(1); /*aborta programa*/
	}
	/* insere elemento na próxima posição livre */
	p->vet[p->n] = v;
	p->n++;
}
```

c) A função pop retira o elemento do topo da pilha, fornecendo seu valor como retorno. Podemos também verificar se a pilha está ou não vazia.

```
float pop(Pilha* p)
{
	float v;
	if (vazia(p)) {
		printf("Pilha vazia.\n");
		exit(1); /*aborta programa*/
	}
	/*retira elemento do topo*/
	v = p->vet[p->n-1];
	p->n--;
	return v;
}
```

d) A função que verifica se a pilha está vazia pode ser dada por:

```
int vazia(Pilha* p)
{
	return (p->n == 0);
}
```

e) Finalmente, a função para liberar a memória alocada pela pilha pode ser:

```
void libera(Pilha* p)
{
	free(p);
}
```

2.5 - Pilhas encadeadas

A implementação estática de pilhas impõe várias restrições no que se refere a capacidade de crescimento da estrutura de dados e com relação ao tipo de dados a ser alocado. Adicionalmente, a

implementação de pilhas utilizando arrays impõe uma restrição adicional referente ao fato de que toda a estrutura deve ser alocada sequencialmente na memória. Uma forma de lidar com tais limitações, é utilizar a implementação dinâmica.

A implementação dinâmica de pilhas pode ser feita de diversas maneiras. Neste contexto, será apresentada a implementação dinâmica usando listas encadeadas. Nela, os elementos são armazenados na lista e a pilha pode ser representada simplesmente por um ponteiro para o primeiro nó da lista.

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

```
typedef struct {
float info;
struct No* anterior;
} No;
```

A estrutura da pilha é então simplesmente:

```
typedef struct {
No* topo;
} Pilha;
```

Assim como uma pilha de alocação sequencial, temos as operações de inicialização da pilha, inserção de um elemento e remoção de um elemento:

a) A função cria aloca a estrutura da pilha e inicializa a lista como sendo vazia:

```
Pilha* cria(void)
{
Pilha *p;
p = (Pilha*) malloc(sizeof(Pilha));
p->topo = NULL;
return p;
}
```

b) A função push empilha um elemento, ou seja coloca um elemento no tipo da pilha

```
Pilha* push(Pilha *p, float v)
{
        No* aux;
        aux = (No*) malloc(sizeof(No));
        aux->info = v;
        aux->anterior = p->topo;
        p->topo = aux;
        return aux;
}
```

c) A função pop desempilha um elemento, ou seja retira um elemento do topo da pilha

```
float pop(Pilha *p)
{
        float v;
        No* aux;
        if (vazia(p))
```

```
	{
		printf("Pilha vazia.");
		exit(1); /*aborta o programa*/
	}
	v = p->topo->info;
	aux = p->topo;
	p->topo = aux->anterior;
	free(aux);
	return v;
}
```

d) A função vazia verifica se a pilha tem elementos

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

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

```
void libera(Pilha *p)
{
	No* q = p->topo;
	while (q != NULL)
	{
		No *t = q->anterior;
		free(q);
		q = t;
	}
	free(p);
}
```

2.6 - Exemplo de uso: calculadora pós-fixada

Um bom exemplo de aplicação de pilha é o funcionamento das calculadoras da HP (Hewlett-Packard). Elas trabalham com expressões pós-fixadas, então para avaliarmos uma expressão como (1-2)*(4+5) podemos digitar 1 2 – 4 5 + *. O funcionamento dessas calculadoras é muito simples. Cada operando é empilhado numa pilha de valores. Quando se encontra um operador, desempilha-se o número apropriado de operandos (dois para operadores binários e um para operadores unários), realiza-se a operação devida e empilha-se o resultado. Deste modo, na expressão acima, são empilhados os valores 1 e 2. Quando aparece o operador -, 1 e 2 são desempilhados e o resultado da operação, no caso -1 (= 1 - 2), é colocado no topo da pilha. A seguir, 4 e 5 são empilhados. O operador seguinte, +, desempilha o 4 e o 5 e empilha o resultado da soma, 9. Nesta hora, estão na pilha os dois resultados parciais, -1 na base e 9 no topo. O operador *, então, desempilha os dois e coloca -9 (= -1 * 9) no topo da pilha.

3 – Filas

3.1- Introdução