Pilha

1. Introdução

Uma pilha é um tipo específico de lista encadeada, onde só é permitido inserir elementos no início da pilha, e retirar do início da pilha, também chamado topo da pilha. Assim, o último elemento inserido é o primeiro a ser retirado. Esse comportamento frequentemente é conhecido pela sigla LIFO (Last In – First Out).

O comportamento de uma estrutura de dados do tipo Pilha simula o comportamento de uma pilha de pratos, em que só se pode colocar mais um prato sobre o topo da pilha, ou retirar o prato que está no topo.

Da mesma forma que qualquer lista, a pilha pode conter qualquer tipo de dado, mas para nossos exemplos, vamos usar o tipo mais simples possível, que é um dado inteiro. A implementação de pilha pode ser feita com alocação estática (vetores) ou alocação dinâmica, utilizando ponteiros. Para nossos exemplos, usaremos alocação dinâmica. A implementação de pilha com vetores é deixada como exercício opcional. A declaração de tipo da pilha dinâmica pode ser:

```
struct elemento {
   int dado;                  // Conteúdo (inteiro)
   struct elemento *prox;     // Ponteiro para o próximo registro
};
typedef struct elemento *Pilha;
```

Ou, tudo em um único comando.

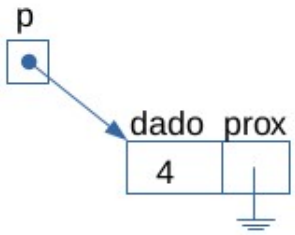
```
typedef struct elemento
{
     int dado;                // Conteúdo (inteiro)
     struct elemento *prox;   // Ponteiro para o próximo registro
} *Pilha;
```

Observe que qualquer variável declarada do tipo `Pilha` será um ponteiro para uma estrutura do tipo elemento, da mesma forma que já usamos para o tipo Lista, na implementação de listas encadeadas.

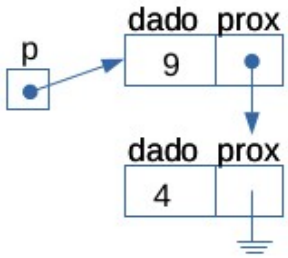
Para exemplificarmos o comportamento da pilha, suponha uma pilha vazia `p`, na qual vamos inserir os elementos 4, 9, 5 e 3. A sequência de resultados da inserção desses elementos será:

1. Pilha `p` vazia:

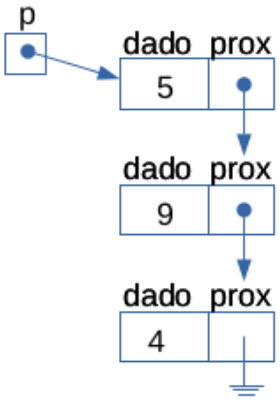
2. Após a inserção do elemento 4:



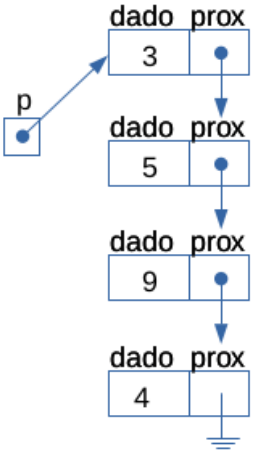
3. Após a inserção dos elementos 4 e 9:



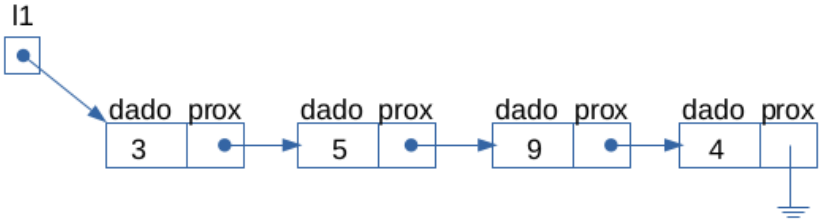
4. Após a inserção dos elementos 4, 9 e 5:



5. Após a inserção dos elementos 4, 9, 5 e 3:



Em geral o desenho da representação de uma pilha é feita na direção vertical somente por motivos didáticos. Observe que a sequência é a mesma da estrutura de uma lista com inserção no início, para a mesma sequência de elementos, já vista:



2. Operações com a pilha.

As operações definidas sobre uma pilha são:

- Criar uma pilha vazia;
- Empilhar um elemento (Inserir no topo da pilha);
- Desempilhar um elemento (Excluir do topo da pilha);
- Verificar se a pilha está vazia.

Embora não seja uma operação definida formalmente sobre uma pilha, vamos implementar também uma função para imprimir o conteúdo da pilha, apenas para fins didáticos, e de depuração do código.

2.1. Criação de uma pilha.

A criação de uma pilha precisa produzir uma lista vazia, o que neste caso significa um ponteiro do tipo `Pilha`, apontando para `NULL`. Isso pode ser implementado por uma função que retorna um ponteiro nulo, do mesmo modo já realizado para a criação de uma lista vazia.

```
Pilha criaPilha()
{
     return NULL;
}
```

Para chamar essa função, o retorno da função `criaPilha()` deve ser armazenado em uma variável do tipo `Pilha`, a exemplo das linhas a seguir.

```
Pilha p;            // Declaração da variável p

p = criaPilha();    // Inicialização da p (pilha vazia)
```

Observe que após a execução dessas linhas a pilha `p` contém uma pilha vazia, ou seja, `p` contém o valor `NULL`. A representação gráfica é:

2.2. Testar se a pilha está vazia.

Uma função bastante simples, mas útil para a compreensão do código é a função pilhaVazia(), que retorna TRUE (1) se a pilha está vazia, e FALSE (0) em caso contrário.

```
#define TRUE 1
#define FALSE 0

int pilhaVazia(Pilha p)
{
     if (p == NULL)
          return (TRUE);
     else
          return (FALSE);
}
```

2.3. Exibição do conteúdo de uma pilha.

Para imprimir o conteúdo de uma pilha é necessário percorrer toda a pilha, sequencialmente, do primeiro elemento até o final da lista. Isso pode ser implementado pelo código abaixo.

```
// Imprime todos os nodos da pilha p
void imprimePilha(Pilha p)
{
     Pilha ap;

     printf("\nItens da pilha\n");
     ap = p;
     while (ap != NULL)
     {
          printf("%d - ",ap->dado);
          ap = ap->prox;
     }
     printf("\n");
}
```

2.4. Empilhar um elemento.

Empilhar um elemento em uma pilha segue o mesmo procedimento da inserção no início de uma lista. Isso implica em criar um novo nodo, colocar o conteúdo desejado nesse novo nodo e depois conectar esse novo nodo no topo da pilha (antes do topo anterior).

```
// Empilha o elemento e no topo da pilha p
Pilha empilha(Pilha p, int e)
{
     Pilha novo;

     novo = malloc(sizeof(struct elemento));  // Aloca o espaço
     novo -> dado = e;  // Atribui os dados ao novo elemento
     novo -> prox = p;  // O próximo do novo é o topo anterior
     return (novo);  // O novo elemento passa a ser o topo
}
```

Observe que a função funciona tanto para o caso onde a pilha está vazia (p == NULL), quanto para o caso onde p não está vazia (p != NULL), pois o próximo elemento do novo topo será NULL se a pilha estava vazia ou o antigo topo, se a pilha não estava vazia.

Essa função sempre inclui um novo elemento no início (topo) da pilha, e portanto, altera o início da pilha. O endereço do novo elemento é retornado, e o topo da pilha precisa ser atualizado com o novo endereço. Nesse caso, considere que no programa principal tenhamos uma lista `p1` e queremos empilhar um dado que está na variável inteira `valor`. A chamada da função pode ser:

```
p1 = empilha(p1, valor);
```

2.5. Desempilhar um elemento da pilha.

O único elemento que se pode retirar de uma pilha é o elemento que está no topo (no início) da pilha. Dessa forma não é necessário percorrer a lista até localizar o elemento. Nesse caso, como o topo da pilha será alterado, e também será retornado o elemento retirado, optamos nesse exemplo por passar o parâmetro `e` por referência (ponteiro), de modo que ele possa ser alterado dentro da função.

Antes de retirar um elemento, é necessário apenas verificar se a pilha não está vazia.

```
Pilha desempilha (Pilha p, int *e)
{
     Pilha ap;  // apontador auxiliar

     if (!pilhaVazia(p))
     {    // se a pilha não for vazia
          // Retorna o valor do elemento que está no topo da pilha
          *e = p->dado;
          // Salva o endereço do topo para liberação
          ap = p;
          // Faz o topo apontar para o próximo elemento
          p = p->prox;
          // Libera o espaço ocupado pelo elemento removido
          free(ap);
     }
     else
     {    // A pilha está vazia. Retorna valor -1.
          *e = -1;
          printf("\nPilha Vazia");
     }
     return (p);
}
```

Resumindo textualmente, se a pilha não estiver vazia, o valor de retorno é o valor do topo. Libera o elemento do topo, atualizando o topo da pilha para o próximo elemento. Caso a pilha esteja vazia, o valor de retorno será -1, e será impressa a mensagem “Pilha Vazia”. Ao final, retorna o endereço do elemento que está no topo da pilha.