Pilha:

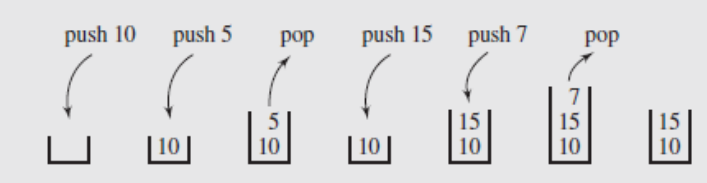
Estrutura linear de dados que pode ser acessada somente por uma de suas extremidades para armazenar e recuperar dados. É como uma pilha de bandejas em uma lanchonete, que são colocadas e retiradas do topo da pilha. A última bandeja colocada é a primeira removida da pilha. Por isso, uma pilha é chamada de estrutura LIFO (*Last in/ First out*).

Pode-se pegar uma bandeja somente se houver bandejas na pilha, e uma bandeja pode ser adicionada à pilha somente se houver espaço suficiente, isto é, se a pilha não estiver muito alta.

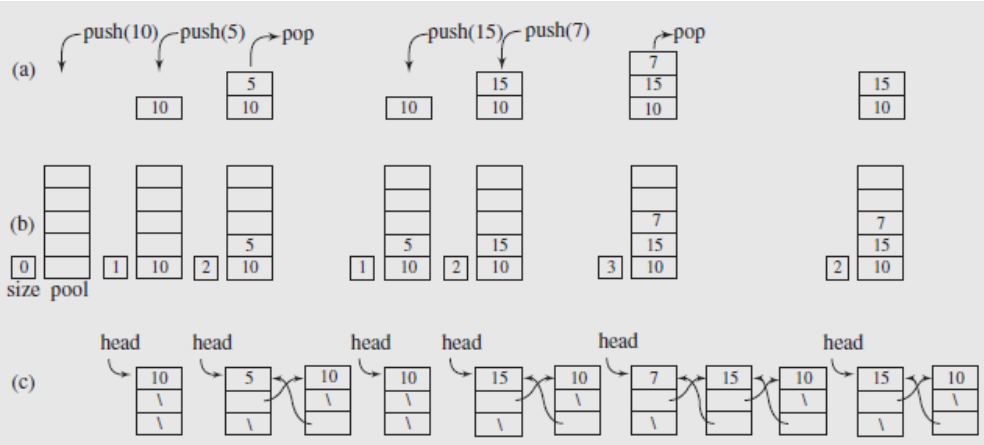
Uma pilha é definida em termos das operações que modificam e das que verificam seu status. As operações são:

- ***clear()*** – Limpa a pilha
- ***isEmpty()*** – Verifica se a pilha está vazia.
- *isFull()* – Verifica se a pilha está cheia (se for implementada por vetor)
- ***push(el)*** – Coloca o elemento *el* no topo da pilha.
- ***pop()*** – Toma o elemento mais alto da pilha.
- ***topEl()*** – Retorna o elemento mais alto da pilha sem removê-lo.

Série de operações executadas em uma pilha:



Série de operações executadas em (a) uma pilha abstrata e a pilha implementada (b) com um vetor e (c) com uma lista ligada:



A <u>implementação da lista ligada</u> casa com a <u>pilha abstrata</u> mais estritamente porque inclui somente os elementos que estão na pilha, já que o número de nós na lista é o mesmo que o de elementos na pilha. Na <u>implementação do vetor</u>, a capacidade da pilha com frequência pode superar seu tamanho.

<u>Implementação de pilha usando lista ligada:</u>

```
#include <iostream>
using namespace std;
class Node{
        public:
           int info;
```

```
        Node *next;

        Node()          {    next = 0;  }
        Node (int valor, Node *pr)  {
          info = valor;
          next = pr;
        }
};

class Stack{
        private:
          Node* head;

        public:
        Stack()          {   head = 0;   }
        void clear ()  {
            Node *tmp=head;
            while (tmp != NULL)  {
                  tmp = tmp -> next;
                  delete head;
                  head = tmp;
            }
        }
        bool isEmpty()    {
             return (head==NULL);
        }
        void push(int val)  {
             head = new Node (val, head);
             head->info=val;
        }
        void pop()  {
            cout << "\n Elemento removido" << popEl();
             if (head != NULL)  {
                  Node *tmp = head;
                  head = head -> next;
                  delete tmp;
            }
        }
        int popEl()    {         //Mostra o elemento no topo da pilha
                if (head == NULL)  {
                      cout << "\n Pilha vazia!";
                      return -1;
                }
                else
                      return head ->info;
        }
        void printStack()  {
               Node *tmp=head;
               while (tmp != NULL)   {
                     cout << "\n " << tmp->info;
                     tmp = tmp->next;
               }
        }
};
//Exemplo de implementação:
int main()   {
   Stack pilha;
```

```
    pilha.push(10);
    pilha.push(20);
    pilha.push(30);

    cout << "\n Pilha atual";
    pilha.printStack();
    cout << "\n Elemento no topo da pilha:"  << pilha.popEl();

    cout << "\n Pilha atual";
    pilha.pop();
    pilha.printStack();
    cout << "\n Elemento no topo da pilha:"  << pilha.popEl();

    if (pilha.isEmpty() != 0)
       cout << "\n Pilha vazia";
    else
       cout << "\n Pilha não vazia";

    pilha.clear();
    pilha.printStack();
    if (pilha.isEmpty() != 0)
       cout << "\n Pilha vazia!";
    else
       cout << "\n Pilha não vazia";
}
```

Exercícios:

1- Inverta a ordem dos elementos na pilha S (usando duas pilhas adicionais ou uma pilha adicional e algumas variáveis adicionais);

```
void Stack :: inverterElementos(Stack &pil)    {
      Stack pilha2;
      if (head != NULL) {
            Node *tmp = head;
            while (tmp != NULL) {
                pilha2.push(tmp->info);
                 tmp = tmp->next;
            }
      }
      pil = pilha2;
   }
   Na chamada à função:  pilha.inverterElementos(pilha);
```

2- Retornar a quantidade de elementos pares na lista;
3- Retornar a quantidade de elementos negativos na lista;
4- Retornar a média aritmética dos elementos da pilha;
5- Crie uma pilha de números reais, adicione cinco valores e crie apenas uma função que retorne a soma dos elementos ímpares da pilha e a soma dos elementos pares. Dica: Usar passagem por referência.
6- Ponha os elementos da pilha S na ordem ascendente usando uma pilha adicional e algumas variáveis adicionais;
7- Implemente a pilha como um vetor.