

Pilhas e Filas

Pilhas

Prof. Edson Alves - UnB/FGA

2020

1. Definição
2. Implementação
3. Aplicações de Pilhas

Definição

Definição de pilha

- Uma pilha é um tipo de dados abstrato cuja interface define que o último elemento inserido na pilha é o primeiro a ser removido
- Esta estratégia de inserção e remoção é denominada LIFO – *Last In, First Out*
- De acordo com sua interface, uma pilha não permite acesso aleatório aos seus elementos (apenas o elemento do topo da pilha pode ser acessado)
- As operações de inserção e remoção devem ter complexidade $O(1)$

Interface de uma pilha

Método	Complexidade	Descrição
clear(P)	$O(N)$	Esvazia a pilha P, removendo todos os seus elementos
empty(P)	$O(1)$	Verifica se a pilha P está vazia ou não
push(P, x)	$O(1)$	Insere o elemento x no topo da pilha P
pop(P)	$O(1)$	Remove o elemento que está no topo da pilha P
top(P)	$O(1)$	Retorna o elemento que está no topo da pilha P
size(P)	$O(1)$	Retorna o número de elementos armazenados na pilha P

Exemplo dos métodos da interface de uma pilha

Método

Retorno

empty(P)

Pilha

Exemplo dos métodos da interface de uma pilha

Método

Retorno

empty(P)

True

Pilha

Exemplo dos métodos da interface de uma pilha

Método

Retorno

push(P, 12)

Pilha

Exemplo dos métodos da interface de uma pilha

Método

Retorno

push(P, 12)

Pilha



Exemplo dos métodos da interface de uma pilha

Método

Retorno

push(P, -3)

Pilha



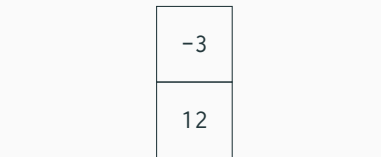
Exemplo dos métodos da interface de uma pilha

Método

Retorno

push(P, -3)

Pilha



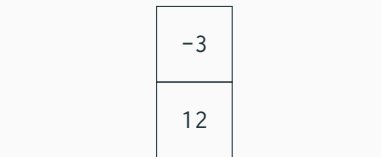
Exemplo dos métodos da interface de uma pilha

Método

Retorno

pop(P)

Pilha



Exemplo dos métodos da interface de uma pilha

Método

Retorno

pop(P)

Pilha



Exemplo dos métodos da interface de uma pilha

Método

Retorno

push(P, 5)

Pilha



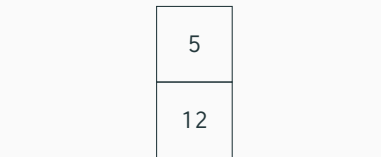
Exemplo dos métodos da interface de uma pilha

Método

Retorno

push(P, 5)

Pilha



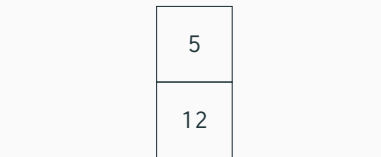
Exemplo dos métodos da interface de uma pilha

Método

Retorno

push(P, 47)

Pilha



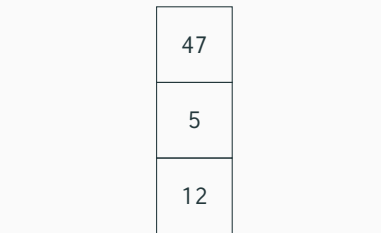
Exemplo dos métodos da interface de uma pilha

Método

Retorno

push(P, 47)

Pilha



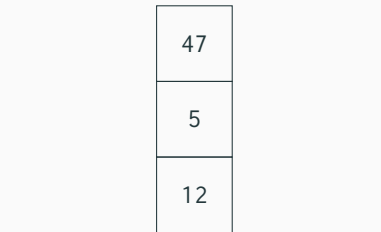
Exemplo dos métodos da interface de uma pilha

Método

Retorno

top(P)

Pilha



Exemplo dos métodos da interface de uma pilha

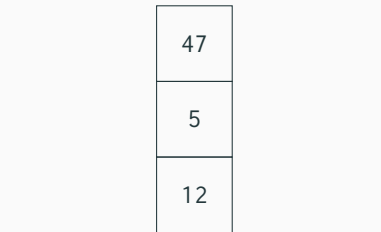
Método

Retorno

top(P)

47

Pilha



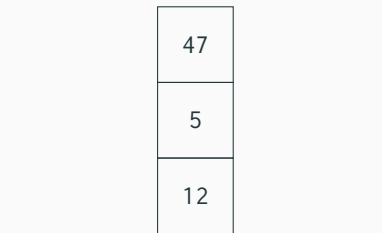
Exemplo dos métodos da interface de uma pilha

Método

Retorno

`size(P)`

Pilha



Exemplo dos métodos da interface de uma pilha

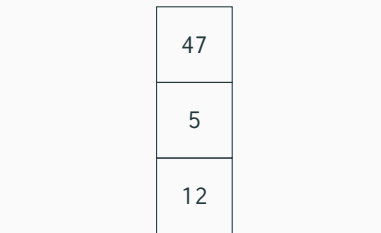
Método

Retorno

size(P)

3

Pilha



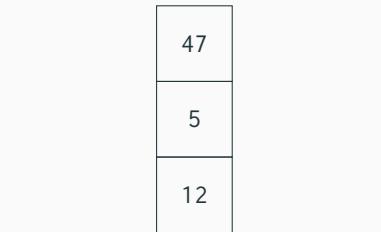
Exemplo dos métodos da interface de uma pilha

Método

Retorno

clear(P)

Pilha



Exemplo dos métodos da interface de uma pilha

Método

Retorno

clear(P)

Pilha

Implementação

Implementação de uma pilha

- Como uma pilha é um tipo de dados abstrato, ela não impõe nenhuma restrição quanto à sua implementação
- É possível implementar uma pilha por composição, usando listas encadeadas ou vetores
- A estratégia LIFO pode ser implementada fazendo-se a inserção e a remoção em uma mesma ponta da lista, de modo que uma lista simplesmente encadeada é suficiente
- Basta utilizar as operações `push_front()` e `pop_front()`, respectivamente, uma vez que ambas tem complexidade $O(1)$
- Utilizar vetores reduz a quantidade de memória necessária, porém as operações passam a ter complexidade $O(1)$ amortizada, devido às ocasionais realocações do vetor para ampliar sua capacidade máxima
- Se há uma estimativa do número máximo de elementos na pilha, estas realocações podem ser evitadas (ou minimizadas), resultando em uma implementação bastante eficiente

Implementação de uma pilha em C++

```
4 #include <array>
5
6 template<typename T, size_t N>
7 class Stack {
8 public:
9     Stack() : pos(0) {}
10
11     void clear() { pos = 0; }
12     bool empty() const { return pos == 0; }
13     size_t size() const { return pos; }
14
15     void push(const T& x) { elems[pos++] = x; }
16     void pop() { pos--; }
17     const T& top() const { return elems[pos - 1]; }
18
19 private:
20     std::array<T, N> elems;
21     size_t pos;
22 };
```

Implementação de uma pilha em C++

```
1 #include <iostream>
2 #include "stack.h"
3
4 using namespace std;
5
6 int main()
7 {
8     Stack<float, 10> s;
9
10    cout << "Empty? " << s.empty() << '\n';
11
12    s.push(1.8);
13    s.push(-0.7);
14    s.push(2.5);
15
16    cout << "Top = " << s.top() << '\n';
17
18    s.pop();
19
20    cout << "Size = " << s.size() << '\n';
21    cout << "Top = " << s.top() << '\n';
```

Implementação de uma pilha em C++

```
22
23     cout << "Empty? " << s.empty() << '\n';
24
25     s.clear();
26     cout << "Empty? " << s.empty() << '\n';
27
28     // Segmentation Fault: a pilha comporta, no máximo, 10 elementos
29     for (int i = 0; i < 20; ++i)
30         s.push(i*0.5);
31
32     return 0;
33 }
```

- A biblioteca padrão de templates (STL) do C++ provê o contêiner `stack`, que implementa uma pilha
- Tanto o tipo de dado a ser armazenado quando o contêiner que será usado na composição são parametrizáveis
- Por padrão, o contêiner utilizado é um deque (*double-ended queue*), mas os contêineres `vector` e `list` são igualmente válidos
- A interface é idêntica à apresentada anteriormente, exceto pela adição do método `swap()`, que troca os elementos de duas pilhas em $O(1)$, e pela exclusão do método `clear()`

Exemplo de uso da classe stack da STL

```
1 #include <iostream>
2 #include <stack>
3 #include <vector>
4
5 using namespace std;
6
7 int main()
8 {
9     stack<int, vector<int>>> s, t;
10
11     cout << "Empty? " << s.empty() << '\n';
12
13     for (int i = 1; i <= 10; ++i)
14         s.push(2*i);
15
16     s.pop();
17
```

Exemplo de uso da classe stack da STL

```
18  cout << "Top = " << s.top() << '\n';
19  cout << "Size = " << s.size() << '\n';
20
21  s.swap(t);
22
23  cout << "Size = " << s.size() << '\n';
24  cout << "T size = " << t.size() << '\n';
25  cout << "T empty? " << t.empty() << '\n';
26
27  return 0;
28 }
```

Aplicações de Pilhas

Identificação de delimitadores

- Delimitadores são caracteres de marcação que delimitam um conjunto de informações, e devem ter um símbolo (ou conjunto de símbolos) que determine o início e o fim do conjunto
- Em C++, os caracteres `()`, `[]`, `{}` e os pares de caracteres `/*`, `*/` são delimitadores
- As pilhas podem ser utilizadas para verificar se, em uma determinada expressão, os delimitadores foram abertos e fechados corretamente
- Por exemplo, as expressões `(())`, `[()()]`, `()[]` são válidas
- Já as expressões `[]`, `)()`, `()`, `[][][]` são inválidas
- O algoritmo é simples: cada símbolo que abre um conjunto é colocado no topo da pilha
- A cada símbolo que fecha o topo da pilha é observado: se contiver o símbolo que abre correspondente, ele é removido e o algoritmo continua; caso contrário, a expressão é inválida
- Ao final do algoritmo a expressão será válida se a pilha estiver vazia

Exemplo de identificação de delimitadores

```
1 #include <iostream>
2 #include <stack>
3 #include <map>
4
5 using namespace std;
6
7 bool is_valid(const string& expression)
8 {
9     static map<char, char> open { {'}', '('}, {']', '['}, {'}', '{'}, };
10    stack<char> s;
11
12    for (const auto& c : expression)
13    {
14        switch (c) {
15            case '(':
16            case '[':
17            case '{':
18                s.push(c);
19                break;
```

Exemplo de identificação de delimitadores

```
21     case ')':
22     case ']':
23     case '}':
24         if (s.empty() || s.top() != open[c])
25             return false;
26
27         s.pop();
28     }
29 }
30
31 return s.empty();
32 }
33
34 int main()
35 {
36     string expression;
37     getline(cin, expression);
38     cout << (is_valid(expression) ? "Ok" : "Invalid") << '\n';
39
40     return 0;
41 }
```

Soma de grandes números

- As pilhas também podem ser utilizadas para somar números com um grande número de dígitos
- Os tipos primitivos integrais do C/C++ tem restrições de tamanho (em *bytes*) e podem levar a erros de *overflow* caso o resultado seja demasiadamente grande
- Com pilhas é possível somar números de qualquer magnitude
- A ideia é armazenar os números como pilhas de dígitos, de modo que as unidades fiquem nos topos das pilhas
- Daí é só colocar os resultados das somas dos topos em uma terceira pilha, tomando cuidado com o vai um (*carry*), quando for o caso

Exemplo de adição de grandes números

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 string add(const string& a, const string& b)
6 {
7     stack<int> x, y, z;
8
9     for (const auto& c : a)
10         x.push(c - '0');
11
12     for (const auto& c : b)
13         y.push(c - '0');
14
15     auto N = max(a.size(), b.size()), carry = 0ul;
16
17     while (N--)
18     {
19         auto c = x.empty() ? 0 : x.top();
20         auto d = y.empty() ? 0 : y.top();
21         auto res = c + d + carry;
```

Exemplo de adição de grandes números

```
23     z.push(res % 10);
24     carry = res / 10;
25
26     if (not x.empty()) x.pop();
27     if (not y.empty()) y.pop();
28 }
29
30 if (carry)
31     z.push(carry);
32
33 ostreamstream oss;
34
35 while (!z.empty())
36 {
37     oss << z.top();
38     z.pop();
39 }
40
41 return oss.str();
42 }
43
```

Exemplo de adição de grandes números

```
44 int main()
45 {
46     string a, b;
47
48     cin >> a >> b;
49     cout << a << " + " << b << " = " << add(a, b) << '\n';
50
51     return 0;
52 }
```

1. **DROZDEK**, Adam. *Algoritmos e Estruturas de Dados em C++*, 2002.
2. **KERNIGHAN**, Bryan; **RITCHIE**, Dennis. *The C Programming Language*, 1978.
3. **STROUSTROUP**, Bjarne. *The C++ Programming Language*, 2013.
4. C++ Reference¹.

¹<https://en.cppreference.com/w/>