

Estruturas de Dados

Prof. Rodrigo Martins

rodrigo.martins@francomontoro.com.br

Cronograma da Aula

- Estruturas de Dados Avançadas
- Pilhas
 - Estática
 - Dinâmica
- Exemplos
- Exercícios

Estruturas de Dados Avançadas

- Entre as principais estruturas de dados avançadas temos:
 - pilhas, filas, listas e árvores.
- Estas estruturas armazenam dados e são manipuladas por funções básicas do tipo:
 - cria, insere, elimina, consulta e altera.

Pilhas

- Pilha é uma estrutura de dados onde os dados são armazenados um sobre o outro;
- As inserções ocorrem no topo da pilha.
- As exclusões também ocorrem no topo da pilha;
- Este tipo de estrutura é também conhecida como LIFO (“Last In First Out” = “Último a Entrar é o Primeiro a Sair”) ou FILO (“First In Last Out” = “Primeiro a Entrar é o Último a Sair”).

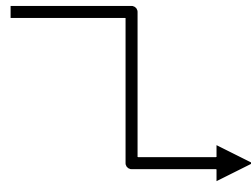
Pilhas

- Estas estruturas podem ser implementadas tanto da forma estática quanto da forma dinâmica.
- As estruturas de pilha são comumente usadas em algoritmos de gerenciamento de memória (escopo de variáveis e retorno de procedimentos), compiladores e em outras aplicações.

Pilhas

- Como resolver o problema em termos computacionais?
- Primeiro precisamos de um vetor de inteiros.
- Segundo uma variável que conterà a posição livre.

Posição

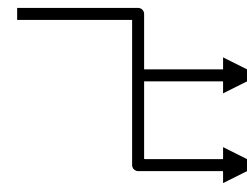


4	
3	
2	
1	
0	

Pilhas

- Vamos agora ao algoritmo de inserção, que é chamado de **push**.
- A pilha está cheia?
 - Sim, mensagem de pilha cheia.
 - Fim da inserção.
- Colocar o número a ser inserido no vetor no índice equivalente a posição.
- Adicionar 1 em posição.

Posição

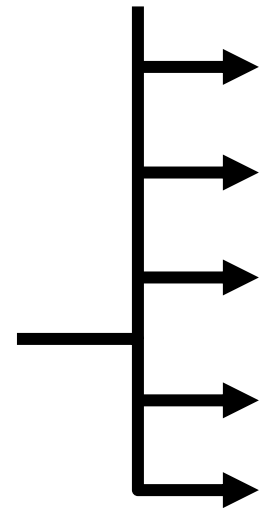


4	
3	
2	
1	
0	2

Pilhas

- Assim vamos preenchendo até o último.
- Agora a pilha está cheia!
- Veja onde está posição.
- Se houver mais um pedido de inserção, devemos emitir uma mensagem de erro.


Posição



4	8
3	7
2	5
1	1
0	2

Pilhas

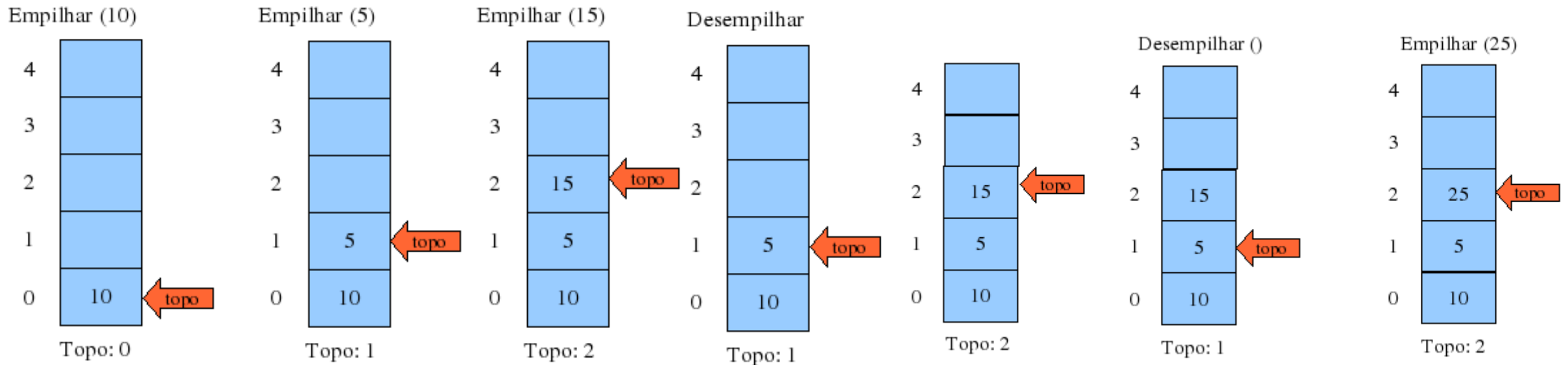
- Vamos ao algoritmo de retirada. Ele é chamado de **pop**.
- A pilha está vazia?
 - Se sim, mensagem de erro
 - Fim da retirada
- Subtrair 1 de posição.
- Retornar o número que está no índice equivalente a posição.



4	
3	7
2	5
1	1
0	2

Exemplo 1

- Algoritmo em C++ da implementação da pilha da forma estática.



```
1  #include <iostream>
2  #define tamanho 5
3
4  using namespace std;
5
6  typedef struct{
7      int topo ;
8      int item[tamanho] ;
9  }PILHA;
10
11
12  void iniciaPilha (PILHA &p) {
13      p.topo = -1 ;
14  }
15
16  bool pilhaVazia(PILHA p){
17      if(p.topo == -1){
18          return true;
19      }
20      else{
21          return false;
22      }
23  }
24
25  bool pilhaCheia(PILHA p){
26      if (p.topo == tamanho-1){
27          return true;
28      }
29      else{
30          return false;
31      }
32  }
```

```
33
34 void empilha(PILHA &p, int x){
35     p.topo++;
36     p.item[p.topo]=x;
37 }
38
39 int desempilha(PILHA &p){
40     return (p.item[p.topo--]);
41 }
42
43
44 int main(){
45
46     int cont=0;
47     PILHA s;
48
49     //cria a pilha
50     iniciaPilha(s);
51
52     //Verifica se a pilha está vazia
53     if (pilhaVazia(s)){
54         cout << "A pilha esta vazia." << endl;
55     }
56     else{
57         cout << "A pilha nao esta vazia." << endl;
58     }
59
```

```
60 //empilha 5 elementos
61 empilha(s,12);
62 empilha(s,33);
63 empilha(s,7);
64 empilha(s,11);
65 empilha(s,22);
66
67 //exibe os itens da pilha
68 do{
69     cout << "Item empilhado: " << s.item[cont] << endl;
70     cont++;
71 }
72 while (cont!=tamanho);
73
74 //Verificar que a pilha está cheia
75 if(pilhaCheia(s)){
76     cout << "A pilha esta cheia." << endl;
77 }
78 else{
79     cout << "A pilha nao esta cheia." << endl;
80 }
81
82 //desempilha exibindo na tela os itens
83 do{
84     cout << "Item desempilhado: " << desempilha(s) << endl;
85 }
86 while (s.topo!=-1);
87
```

```
88 //Verificar que a pilha está cheia
89 if(pilhaCheia(s)){
90     cout << "A pilha esta cheia." << endl;
91 }
92 else{
93     cout << "A pilha nao esta cheia." << endl;
94 }
95
96 return 0;
97 }
98
```

A biblioteca <stack>

- A biblioteca <**stack**> é parte da biblioteca padrão do C++ e fornece a classe **stack** para trabalhar com pilhas.
- **stack** é uma estrutura de dados que segue o princípio LIFO (Last In, First Out), o que significa que o último elemento inserido é o primeiro a ser removido.

Operações básicas da forma dinâmica

- **push:**
 - A operação **push** é usada para adicionar um elemento ao topo da pilha.
 - Quando um elemento é inserido na pilha, ele é colocado na posição mais alta disponível, ou seja, no topo.
 - Se a pilha tiver um tamanho fixo e já estiver cheia, a operação **push** pode falhar (ou lançar um erro) porque não haverá espaço disponível para adicionar o novo elemento.

```
#include <iostream>
#include <stack>

using namespace std;

int main() {
    stack<int> pilha;

    pilha.push(100); // Adiciona o número 100 à pilha
    pilha.push(200); // Adiciona o número 200 à pilha

    cout << "Elemento no topo após inserção: " << pilha.top() << endl;
}
```


Operações básicas da forma dinâmica

- **pop:**
 - A operação **pop** é usada para remover o elemento que está no topo da pilha.
 - Quando um elemento é removido, ele deixa o topo da pilha e o próximo elemento na ordem se torna o novo topo.
 - Se a pilha estiver vazia, a operação **pop** pode falhar (ou lançar um erro) porque não há elementos para remover.

```
#include <iostream>
#include <stack>

using namespace std;

int main() {
    stack<int> pilha;

    pilha.push(50);
    pilha.push(60);
    pilha.pop(); // Remove o número 60 (o último elemento inserido) do topo

    cout << "Elemento no topo após remoção: " << pilha.top() << endl;
}
```

Operações básicas da forma dinâmica

- **top:**

- A operação **top** é usada para examinar o elemento que está no topo da pilha, sem removê-lo.
- Ela retorna o valor do elemento no topo, permitindo que o programa saiba qual é o elemento atual sem alterar a pilha.
- Se a pilha estiver vazia, a operação **top** pode retornar um valor especial ou lançar um erro, dependendo da implementação.

```
#include <iostream>
#include <stack>

using namespace std;

int main() {
    stack<string> pilha;

    pilha.push("pacote 1");
    pilha.push("pacote 2");

    // Retorna o elemento no topo sem removê-lo
    cout << "Elemento no topo: " << pilha.top() << endl;
}
```

Operações básicas da forma dinâmica

- **empty:**
 - A operação **empty** verifica se a pilha está vazia. Ela retorna **true** se a pilha estiver vazia e **false** caso contrário.

```
#include <iostream>
#include <stack>

using namespace std;

int main() {
    stack<int> pilha;

    cout << "A pilha está vazia? " << (pilha.empty() ? "Sim" : "Não") << endl;

    pilha.push(10);

    cout << "A pilha está vazia? " << (pilha.empty() ? "Sim" : "Não") << endl;
}
```

Operações básicas da forma dinâmica

- **size:**
 - A operação **size** retorna o número de elementos atualmente na pilha.

```
#include <iostream>
#include <stack>

using namespace std;

int main() {
    stack<string> pilha;

    pilha.push("pacote 1");
    pilha.push("pacote 2");
    pilha.push("pacote 3");
    pilha.push("pacote 4");

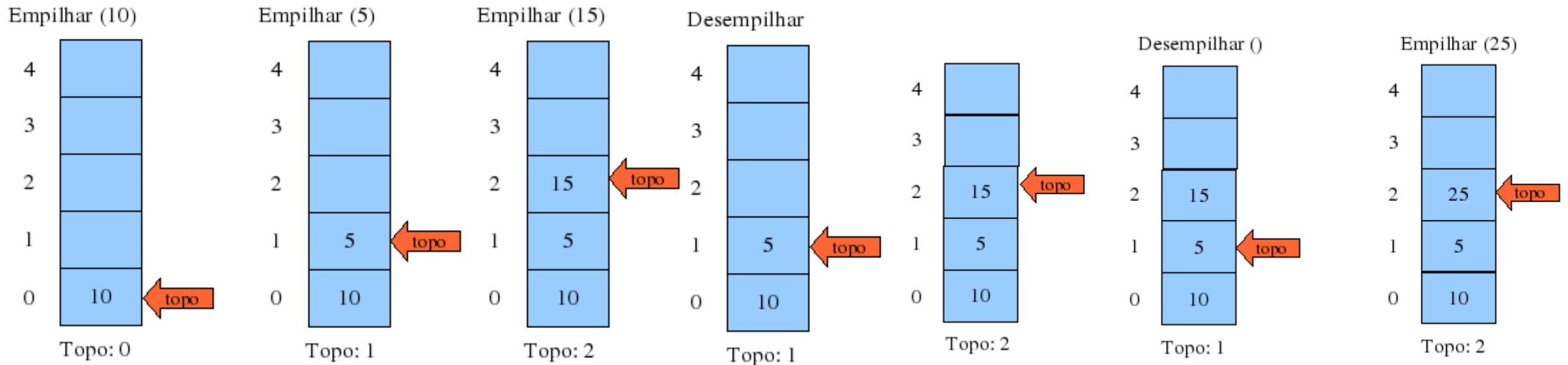
    cout << "Tamanho da pilha: " << pilha.size() << endl;

    pilha.pop();

    cout << "Tamanho da pilha após remoção: " << pilha.size() << endl;
}
```

Exemplo 2

- Algoritmo em C++ da implementação da pilha da forma dinâmica.



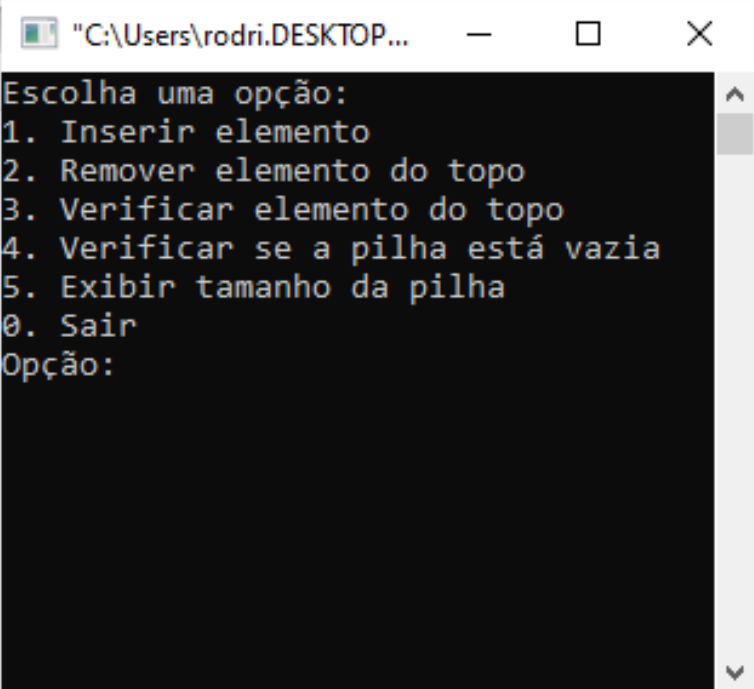
```
1  #include <iostream>
2  #include <stack>
3
4  using namespace std;
5
6  int main(){
7
8      stack <string> roupas;
9
10     //empty retorna se pilha cheia ou vazia
11     if (roupas.empty()){
12         cout << "Pilha vazia" << endl;
13     }
14
15     roupas.push("calca azul"); //push adiciona elementos na pilha
16     roupas.push("calca preta");
17     roupas.push("camiseta branca");
18     roupas.push("camiseta amarela");
19     roupas.push("calca branca");
20     roupas.push("bermuda preta");
21
22     // size mostra o tamanho da pilha
23     cout << "Tamanho da Pilha: " << roupas.size() << endl;
24
25     while(!roupas.empty()){
26         cout << "Peca de Roupas no Topo: " << roupas.top() << endl;
27         roupas.pop(); // retira elementos da pilha
28     }
29
```

```
30     cout << endl;
31
32     cout << "Tamanho da Pilha: " << roupas.size() << endl;
33
34     return 0;
35 }
```

Exercício

1. Escreva um programa em C++ que implemente uma pilha e permita ao usuário realizar as seguintes operações:

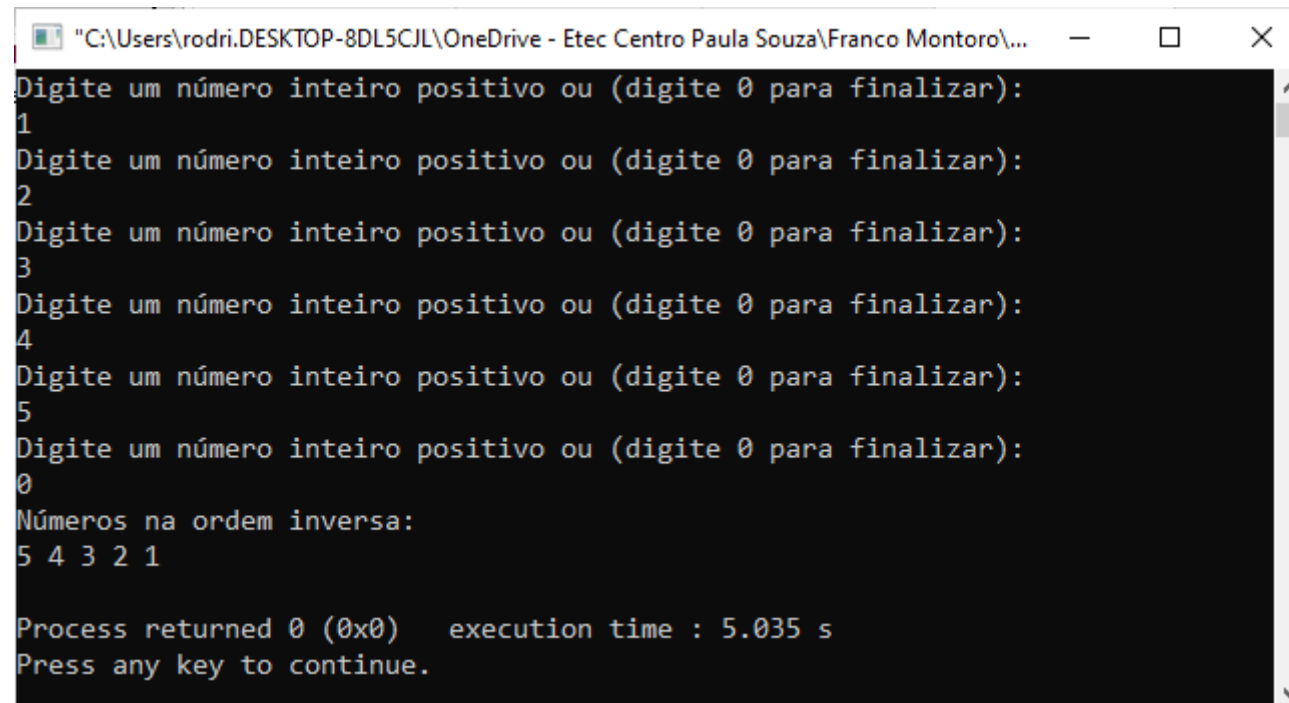
- Inserir um elemento na pilha.
- Remover o elemento do topo da pilha.
- Verificar o elemento do topo da pilha sem removê-lo.
- Verificar se a pilha está vazia.
- Exibir o tamanho da pilha.



```
"C:\Users\rodri.DESKTOP..."
Escolha uma opção:
1. Inserir elemento
2. Remover elemento do topo
3. Verificar elemento do topo
4. Verificar se a pilha está vazia
5. Exibir tamanho da pilha
0. Sair
Opção:
```


Exercício

2. Escreva um programa em C++ que leia uma sequência de números inteiros positivos do usuário e imprima-os na ordem inversa utilizando uma pilha.



```
"C:\Users\rodri.DESKTOP-8DL5CJL\OneDrive - Etec Centro Paula Souza\Franco Montoro\..."
Digite um número inteiro positivo ou (digite 0 para finalizar):
1
Digite um número inteiro positivo ou (digite 0 para finalizar):
2
Digite um número inteiro positivo ou (digite 0 para finalizar):
3
Digite um número inteiro positivo ou (digite 0 para finalizar):
4
Digite um número inteiro positivo ou (digite 0 para finalizar):
5
Digite um número inteiro positivo ou (digite 0 para finalizar):
0
Números na ordem inversa:
5 4 3 2 1

Process returned 0 (0x0)   execution time : 5.035 s
Press any key to continue.
```

Referência desta aula

- Notas de Aula do Prof. Prof. Armando Luiz N. Delgado baseado em revisão sobre material de Prof.a Carmem Hara e Prof. Wagner Zola
- <http://www.cplusplus.com/reference/>

Obrigado

Rodrigo