Universitas Esa Unggul
Powered by Arizona State University®

CSF 101

ALGORITMA & PEMROGRAMAN

Sesi 10

Stack Algorithm C++

Kelompok 7

1. Muhamad Akbar Fadilah - 20200801269
2. Arva Raihan Javier - 20240801344
3. Denis Prastya Putra - 20240801319
4. Christian Niko Saputra - 20240801295

Universitas Esa Unggul
Powered by Arizona State University®

Tujuan Presentasi:

Memahami apa itu Stack Algorithm dan bagaimana penggunaannya dalam pemrograman c++

Daftar Isi:

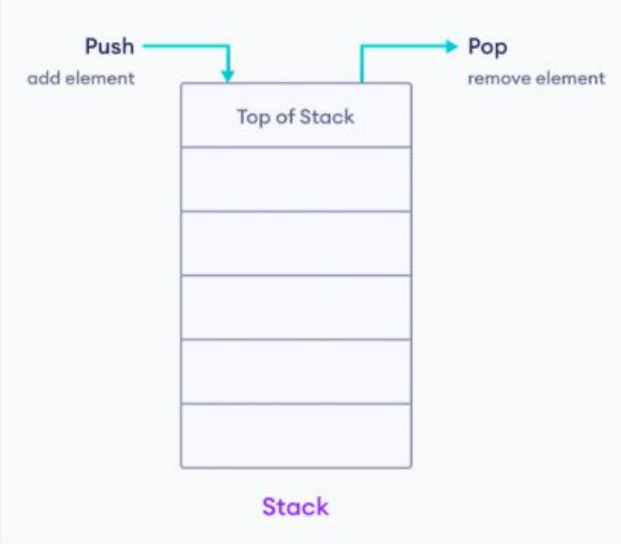
1. **Pengenalan Stack**
2. **Stack Methods**
3. **Implementasi**
4. **Kesimpulan**

Universitas Esa Unggul
Powered by Arizona State University®

PENGENALAN STACK

Pengenalan Stack

Stack adalah struktur data fundamental dalam ilmu komputer yang digunakan untuk menyimpan koleksi objek. Stack beroperasi berdasarkan prinsip ***Last In, First Out (LIFO)***, di mana objek yang terakhir ditambahkan akan menjadi objek pertama yang dihapus. Struktur ini sangat berguna dalam berbagai situasi, seperti membalikkan urutan operasi atau membatalkan tindakan secara berulang.



Pengenalan Stack

Contoh penggunaannya dapat ditemukan dalam fitur Undo/Redo pada aplikasi seperti Microsoft Word. Ketika tombol Undo (Ctrl+Z) ditekan dalam editor teks atau aplikasi desain, perubahan terakhir yang dilakukan akan dibatalkan terlebih dahulu. Semua perubahan ini disimpan dalam struktur ***LIFO***, sehingga perubahan terakhir keluar terlebih dahulu.

Universitas Esa Unggul
Powered by Arizona State University®

STACK METHODS

Stack Methods

Stack menyediakan berbagai methods untuk melakukan operasi berbeda pada tumpukan.

Operasi	Keterangan
push()	menambahkan elemen ke dalam stack.
pop()	menghapus elemen dari stack.
top()	mengembalikan elemen di bagian atas stack.
size()	mengembalikan jumlah element dalam stack.
empty()	mengembalikan ***true*** jika stack kosong.

Universitas Esa Unggul
Powered by Arizona State University®

IMPLEMENTASI

Create a Stack

Untuk membuat tumpukan dalam C++, pertama-tama kita perlu menyertakan file header stack.

```
#include <stack>
```

Setelah kita menyertakan file header stack, kita dapat membuat stack menggunakan sintaks berikut:

```
stack<type> st;
```

Di sini, type menunjukkan tipe data yang ingin kita simpan di stack. Misalnya

```
// create a stack of integers
stack<int> integer_stack;
// create a stack of strings
stack<string> string_stack;
```

Example STL Stack

```cpp
#include <iostream>
#include <stack>
using namespace std;

int main() {
    // create a stack of strings
    stack<string> languages;

    // add element to the Stack
    languages.push("C++");
    languages.push("Java");
    languages.push("Python");

    // print top element
    cout << languages.top();

    return 0;
}
```

Output: ***Python***

Example STL Stack

Pada contoh di slide sebelumnya, kita telah membuat stack string bernama ***languages***. Di sini, kita telah menggunakan medtods ***push()*** untuk menambahkan elemen ke stack. Kami kemudian menggunakan medtods ***top()*** untuk menampilkan elemen teratas.

Add Element Into the Stack

Kami menggunakan methods ***push()*** untuk menambahkan elemen ke dalam stack. Misalnya:

```
#include <iostream>
#include <stack>
using namespace std;

int main() {

  // create a stack of strings
  stack<string> colors;

  // push elements into the stack
  colors.push("Red");
  colors.push("Orange");

  cout << "Stack: ";

  // print elements of stack
   while(!colors.empty()) {
    cout << colors.top() << ", ";
    colors.pop();
  }

  return 0;
}
```

Output: ***Stack: Orange, Red,***

Add Element Into the Stack

Pada contoh di slide sebelumnya, kita telah membuat tumpukan string bernama ***colors***. Kemudian, kita menggunakan methods ***push()*** untuk menambahkan elemen ke tumpukan.

```
colors.push("Red");
colors.push("Orange");
```

kami menggunakan perulangan ***while***.

```
while(!colors.empty()) {
  cout << colors.top() << ", ";
  colors.pop();
}
```

Untuk mencetak semua elemen stack, kita mencetak elemen teratas dan kemudian ***pop*** (hapus) di dalam ***loop***. Proses ini berlanjut berulang kali hingga stack kosong.

Remove Elements From the Stack

Kita dapat menghapus elemen dari stack menggunakan methods ***pop()***. Misalnya:

```
int main() {
  stack<string> colors;

  // push elements into the stack
  colors.push("Red");
  colors.push("Orange");
  colors.push("Blue");

  cout << "Initial Stack: ";
  // print elements of stack
  display_stack(colors);

  // removes "Blue" as it was inserted last
  colors.pop();

  cout << "Final Stack: ";

  // print elements of stack
  display_stack(colors);

  return 0;
}

// utility function to display stack elements
void display_stack(stack<string> st) {

  while(!st.empty()) {
    cout << st.top() << ", ";
    st.pop();
  }

  cout << endl;
}
```

Output:

Initial Stack: Blue, Orange, Red,

Final Stack: Orange, Red,

Remove Elements From the Stack

Pada contoh di slide sebelumnya, kita menggunakan metode ***pop()*** untuk menghapus elemen dari stack. Awalnya, konten stack adalah ***{"Blue", "Orange", "Red"}***. Kemudian kita menggunakan methods ***pop()*** untuk menghapus elemen tersebut.

```
// removes top element
colors.pop()
```

Tindakan ini akan menghapus elemen di bagian atas stack yaitu elemen yang dimasukkan terakhir, yaitu ***"Blue"***.

Oleh karena itu, stack terakhir menjadi ***{"Orange", "Red"}***.

Access Elements From the Stack

Kita mengakses elemen di bagian atas tumpukan menggunakan methods ***top()***. Misalnya:

```
#include <iostream>
#include <stack>
using namespace std;

int main() {

  // create a stack of strings
  stack<string> colors;

  // push element into the stack
  colors.push("Red");
  colors.push("Orange");
  colors.push("Blue");

  // get top element
  string top = colors.top();

  cout << "Top Element: " << top;

  return 0;
}
```

Output: ***Top Element: Blue***

Access Elements From the Stack

Pada contoh di slide sebelumnya, kita telah membuat stack string bernama ***colors*** dan menambahkan stack berikut: ***"Red", "Orange" dan "Blue"***. Kami kemudian menggunakan methods ***top()*** untuk mengakses elemen teratas:

```
string top = colors.top();
```

Di sini, ***"Blue"*** disisipkan terakhir, sehingga merupakan elemen teratas.

Get the Size of the Stack

Kami menggunakan methods ***size()*** untuk mendapatkan jumlah elemen di stack. Misalnya:

```
#include <iostream>
#include <stack>
using namespace std;

int main() {

  // create a stack of int
  stack<int> prime_nums;

  // push elements into the stack
  prime_nums.push(2);
  prime_nums.push(3);
  prime_nums.push(5);

  // get the size of the stack
  int size = prime_nums.size();
  cout << "Size of the stack: " << size;

  return 0;
}
```

Output: ***Size of the stack: 3***

Get the Size of the Stack

Pada contoh di slide sebelumnya, kita telah membuat stack bilangan bulat bernama ***prime_nums*** dan menambahkan tiga elemen ke dalamnya. Kemudian kita menggunakan methods ***size()*** untuk menemukan jumlah elemen dalam stack:

```
prime_nums.size();
```

Karena kita telah menambahkan 3 elemen ke stack, ***prime_nums.size()*** mengembalikan 3.

Check if the Stack Is Empty

Kami menggunakan methods ***empty()*** untuk memeriksa apakah stack kosong. Methods ini mengembalikan:

- 1(***true***) - jika stack kosong
- 0(***false***) - jika stack tidak kosong

```cpp
#include <iostream>
#include <stack>
using namespace std;

Codeium: Refactor | Explain | Generate Function Comment | ×
int main() {

  // create a stack of double
  stack<double> nums;

  cout << "Is the stack empty? ";

  // check if the stack is empty
  if (nums.empty()) {
    cout << "Yes" << endl;
  }
  else {
    cout << "No" << endl;
  }

  cout << "Pushing elements..." << endl;

  // push element into the stack
  nums.push(2.3);
  nums.push(9.7);

  cout << "Is the stack empty? ";

  // check if the stack is empty
  if (nums.empty()) {
    cout << "Yes";
  }
  else {
    cout << "No";
  }

  return 0;
}
```

Output:

Is the stack empty? Yes

Pushing elements...

Is the stack empty? No

Check if the Stack Is Empty

Pada contoh di slide sebelumnya, kita telah menggunakan methods ***empty()*** untuk menentukan apakah stack kosong,

```
if(nums.empty()) { // returns false
  cout << "Yes" << end;;
}
else {
  cout << "No" << endl;
}
```

Awalnya, stack tidak memiliki elemen di dalamnya. Jadi ***nums.empty()*** mengembalikan ***true***. Kami kemudian menambahkan elemen ke stack. kita menggunakan ***nums.empty()*** untuk menentukan apakah stack kosong. Kali ini, ia mengembalikan false.

Universitas Esa Unggul
Powered by Arizona State University®

KESIMPULAN

Kesimpulan

Stack dalam C++ adalah implementasi struktur data stack yang mengikuti prinsip ***Last In, First Out (LIFO)***.

Struktur ini berguna dalam berbagai kasus seperti pembatalan operasi (Undo/Redo).

Stack Methods:

- push(): Menambahkan elemen ke puncak stack.
- pop(): Menghapus elemen dari puncak stack.
- top(): Mengakses elemen di puncak tanpa menghapusnya.
- empty(): Memeriksa apakah stack kosong.
- size(): Mendapatkan jumlah elemen dalam stack.

Kesimpulan

Kelebihan:

- Mudah digunakan dengan antarmuka sederhana.
- Mendukung pengelolaan data secara terstruktur untuk operasi ***LIFO***.
- Tersedia sebagai bagian dari ***Standard Template Library (STL)*** di C++, sehingga pengembang tidak perlu mengimplementasikannya dari awal.

Kekurangan:

- Tidak mendukung ***traversal*** langsung (karena berbasis ***LIFO***).
- Bukan pilihan terbaik untuk struktur data yang membutuhkan akses acak.

Universitas Esa Unggul
Powered by Arizona State University®

Ada Pertanyaan ?

Universitas Esa Unggul
Powered by Arizona State University®

Terima Kasih