LAPORAN PRAKTIKUM
Modul 7
STACK

Disusun Oleh:
Jauhar Fajar Zuhair
2311104072
S1SE-07-2

Dosen :
Wahyu Andri Saputra, S.Pd., M.Eng.

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO

2024

Tujuan Praktikum

1. Memberikan pemahaman mendalam tentang konsep stack dalam konteks struktur data dan algoritma
2. Mengembangkan kemampuan dalam menerapkan berbagai operasi yang terkait dengan stack
3. Meningkatkan kemampuan dalam menyelesaikan berbagai permasalahan menggunakan solusi berbasis stack

Landasan Teori

Stack atau tumpukan adalah sebuah struktur data yang mengorganisir dan mengelola data menggunakan prinsip Last-In, First-Out (LIFO). Prinsip ini dapat diilustrasikan dengan mudah melalui analogi tumpukan piring di kafetaria, dimana piring yang terakhir diletakkan akan menjadi piring pertama yang diambil.

Stack memiliki dua operasi fundamental yaitu push (untuk menambah elemen) dan pop (untuk menghapus elemen teratas). Struktur data ini memainkan peran penting dalam berbagai aplikasi komputasi, seperti:

- Pengelolaan memori komputer
- Evaluasi ekspresi matematika
- Manajemen pemanggilan fungsi dalam pemrograman

Beberapa operasi dasar yang dapat dilakukan pada stack meliputi:

1. Push: Memasukkan elemen baru ke posisi teratas
2. Pop: Mengeluarkan elemen dari posisi teratas
3. Top: Melihat elemen teratas tanpa menghapusnya
4. IsEmpty: Mengecek apakah stack kosong
5. IsFull: Memeriksa apakah stack sudah penuh
6. Size: Menghitung jumlah elemen dalam stack
7. Peek: Melihat nilai pada posisi tertentu tanpa mengubahnya
8. Clear: Menghapus seluruh elemen dalam stack
9. Search: Mencari keberadaan elemen tertentu

Efisiensi stack terletak pada kemampuannya untuk mengakses data secara terstruktur dengan prinsip LIFO, menjadikannya komponen vital dalam pengembangan software dan pemrograman komputer.

Guided
guided1.cpp

```cpp
#define MAX 100
#include <iostream>
using namespace std;

class Stack
{
private:
    int top;
    int arr[MAX];

public:
    Stack() { top = -1; }
    bool isFull() { return top == MAX - 1; }
    bool isEmpty() { return top == -1; }

    void push(int x)
    {
        if (isFull())
        {
            cout << "Stack is full/Overflow\n";
            return;
        }
        arr[++top] = x;
    }
    void pop()
    {
        if (isEmpty())
        {
            cout << "Stack is empty/Underflow\n";
            return;
        }
        top--;
    }
    int peek()
    {
        if (!isEmpty())
        {
            return arr[top];
        }
        cout << "Stack is empty\n";
        return -1;
    }
    void display()
    {
        if (isEmpty())
        {
            cout << "Stack is empty\n";
        }
        else
        {
            for (int i = top; i >= 0; i--)
            {
                cout << arr[i] << " ";
            }
```

```
            cout << "\n";
        }
    }
};
int main()
{
    Stack s;
    s.push(10);
    s.push(20);
    s.push(30);
    cout << "Stack Elements: ";
    s.display();

    cout << "Top Element: " << s.peek() << "\n";
    s.pop();
    s.pop();
    cout << "After Popping, Stack Elements: ";
    s.display();
    return 0;
};
```

Guided2.cpp

```
#include <iostream>

using namespace std;

class Node
{
public:
  int data;
  Node *next;
  Node(int value)
  {
    data = value;
    next = nullptr;
  }
};

class Stack
{
private:
  Node *top;

public:
  Stack() { top = nullptr; };
  bool isEmpty() { return top == nullptr; };

  void push(int x)
  {
    Node *newNode = new Node(x);
    newNode->next = top;
    top = newNode;
  }
  void pop()
```

```
  {
    if (isEmpty())
    {
      cout << "Stack is Underflow/empty\n";
      return;
    }
    Node *temp = top;
    top = top->next;
    delete temp;
  }
  int peek()
  {
    if (!isEmpty())
    {
      return top->data;
    }
    cout << "Stack is empty\n";
    return -1;
  }
  void display()
  {
    if (isEmpty())
    {
      cout << "Stack is empty\n";
      return;
    }
    Node *current = top;
    while (current)
    {
      cout << current->data << " ";
      current = current->next;
    }
    cout << "\n";
  }
};
int main()
{
  Stack s;
  s.push(10);
  s.push(20);
  s.push(30);
  cout << "Stack Elements: ";
  s.display();

  cout << "Top Element: " << s.peek() << "\n";
  s.pop();
  s.pop();
  cout << "After Popping, Stack Elements: ";
  s.display();
  return 0;
};
```

Unguided

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

bool cekPalindrom(string str)
{
    stack<char> st;
    string cleanStr = "";

    // Membersihkan string dan mengubah ke lowercase
    for (char c : str)
    {
        if (isalpha(c))
        {
            cleanStr += tolower(c);
        }
    }

    // Push semua karakter ke stack
    for (char c : cleanStr)
    {
        st.push(c);
    }

    // Bandingkan karakter asli dengan yang di stack
    for (char c : cleanStr)
    {
        if (c != st.top())
        {
            return false;
        }
        st.pop();
    }

    return true;
}

int main()
{
    string kalimat;

    cout << "Masukan Kalimat : ";
```

```
    getline(cin, kalimat);

    cout << "Kalimat tersebut adalah : ";
    if (cekPalindrom(kalimat))
    {
        cout << "Palindrom" << endl;
    }
    else
    {
        cout << "bukan Palindrom" << endl;
    }

    return 0;
}
```

Cara kerja program:

Input:

- Program meminta input kalimat dari user menggunakan getline()
- Contoh input: "ini" atau "telkom"

Pembersihan String:

- Program membersihkan string dari karakter non-huruf
- Mengubah semua huruf menjadi lowercase untuk memastikan pengecekan yang konsisten
- Contoh: "Ini" menjadi "ini"

Proses Pengecekan Palindrom:

- Semua karakter dari string yang sudah dibersihkan di-push ke dalam stack
- Kemudian program membandingkan setiap karakter dari string asli dengan karakter teratas stack
- Jika ada yang tidak cocok, berarti bukan palindrom
- Jika semua cocok, berarti palindrom

Output:

```
Masukan Kalimat : ini
Kalimat tersebut adalah : Palindrom
PS C:\Users\Administrator\Documents\dok ku
kuliah\semester 3\Stuktur Data\07_Pengenal
Masukan Kalimat : telkom
Kalimat tersebut adalah : bukan Palindrom
```

Keunggulan menggunakan stack dalam pengecekan palindrom:

- Memanfaatkan sifat LIFO (Last In First Out) dari stack

- Memudahkan proses pembandingan karakter dari depan dan belakang

- Efisien dalam penggunaan memori karena hanya menyimpan karakter yang diperlukan

Unguided 2

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

string balikKata(string kata)
{
    stack<char> st;
    string hasil = "";

    // Push semua karakter ke stack
    for (char c : kata)
    {
        st.push(c);
    }

    // Pop karakter dari stack untuk mendapatkan kata terbalik
    while (!st.empty())
    {
        hasil += st.top();
        st.pop();
    }

    return hasil;
}
```

```cpp
int main()
{
    string kalimat;
    stack<string> st_kata;

    cout << "Masukkan Kata : ";
    getline(cin, kalimat);

    // Hitung jumlah kata
    stringstream check(kalimat);
    string kata;
    int jumlah_kata = 0;

    while (check >> kata)
    {
        jumlah_kata++;
        // Balik setiap kata dan masukkan ke stack
        st_kata.push(balikKata(kata));
    }

    // Cek minimal 3 kata
    if (jumlah_kata < 3)
    {
        cout << "Error: Masukkan minimal 3 kata!" << endl;
        return 1;
    }

    cout << "Datastack Array : " << endl;
    cout << "Data : ";

    // Ambil kata dari stack dan tampilkan
    while (!st_kata.empty())
    {
        cout << st_kata.top() << " ";
        st_kata.pop();
    }
    cout << endl;

    return 0;
}
```

Penjelasan program:

Fungsi dan Operasi yang Digunakan:

- `balikKata(string kata)`: Fungsi untuk membalik setiap kata individual
- Stack digunakan dua kali:
  - Untuk membalik setiap kata individual (dalam fungsi balikKata)
  - Untuk menyimpan kata-kata yang sudah dibalik (st_kata)

Cara Kerja Program:

a. Input:
   - Program meminta input kalimat dari user
   - Contoh: "Telkom Purwokerto"

b. Proses:
   - Memisahkan kalimat menjadi kata-kata individual
   - Setiap kata dibalik menggunakan fungsi balikKata()
   - Kata yang sudah dibalik disimpan dalam stack

c. Validasi:
   - Memeriksa apakah input memiliki minimal 3 kata
   - Jika tidak, program menampilkan pesan error

d. Output:
   - Menampilkan hasil pembalikan dalam format yang diminta

Contoh Output:

```
Masukkan Kata : Telkom University Pekalongan
Datastack Array :
Data : nagnolakeP ytisrevinU mokleT
```

Penjelasan Operasi Stack:

- Push: Digunakan untuk menyimpan karakter/kata ke dalam stack
- Pop: Digunakan untuk mengambil karakter/kata dari stack
- Top: Digunakan untuk melihat elemen teratas stack
- Empty: Digunakan untuk mengecek apakah stack kosong

Keunggulan Menggunakan Stack:

- Memudahkan proses pembalikan karena sifat LIFO
- Efisien dalam penggunaan memori
- Mudah dalam implementasi