LAPORAN PRAKTIKUM

STRUKTUR DATA

MODUL 7
DOUBLY LINKED LIST
(BAGIAN PERTAMA)

Disusun Oleh :

NAMA : LAHRA BUDI SAPUTRA

NIM : 103112430054

Dosen :

FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2025

A. Dasar Teori

Stack merupakan struktur data yang beroperasi dengan prinsip tumpukan, dimana elemen yang terakhir masuk akan menjadi elemen yang pertama kali diambil (LIFO - Last In First Out). Dalam stack, komponen utama yang disebut "Top" berperan sebagai akses data, dan hanya elemen paling atas saja yang dapat diakses. Operasi dasar pada stack meliputi "Push" untuk menyisipkan elemen baru ke bagian atas tumpukan, dan "Pop" untuk mengambil elemen dari bagian atas tumpukan, mirip dengan operasi insert first dan delete first pada list linear.

Implementasi stack dapat dilakukan melalui representasi pointer atau tabel. Representasi tabel menggunakan array berindeks dengan kapasitas tumpukan yang terbatas. Meskipun terdapat perbedaan dalam pendeklarasian struktur data dan manajemen memori, operasi Push dan Pop pada das stack representasi tabel memiliki prinsip yang sama dengan representasi pointer. Dalam konteks representasi tabel, Push akan menggeser indeks TOP ke indeks berikutnya untuk menyisipkan data, sementara Pop akan mengambil data di posisi indeks TOP dan kemudian menggeser indeks TOP ke indeks sebelumnya, tanpa menghilangkan informasi dari indeks TOP sebelumnya.

B. Guided (berisi screenshot source code & output program disertai penjelasannya)

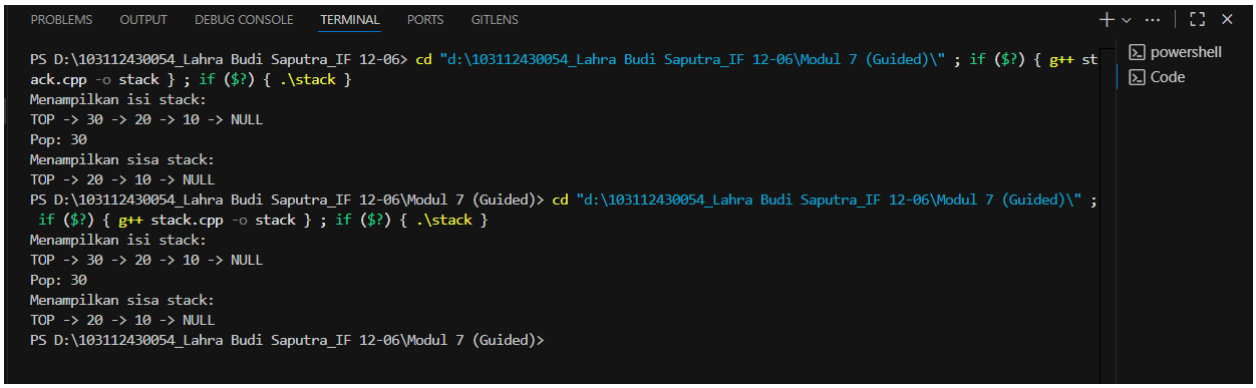
“stack.cpp”

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

struct Node
{
   int data;
   Node *next;
};

bool isEmpty(Node *top)
{
   return top == nullptr;
```

```
}

void push(Node *&top, int data)
{
   Node *newNode = new Node();
   newNode->data = data;
   newNode->next = top;
   top = newNode;
}

int pop(Node *&top)
{
   if (isEmpty(top))
   {
      cout << "Stack kosong, tidak dapat pop!" << endl;
      return 0;
   }

   int poppedData = top->data;
   Node *temp = top;
   top = top->next;

   delete temp;
   return poppedData;
}

void show(Node *top)
{
   if (isEmpty(top))
   {
      cout << "Stack Kosong." << endl;
      return;
   }
```

```
    cout << "TOP -> ";
    Node *temp = top;

    while (temp != nullptr)
    {
        cout << temp->data << " -> ";
        temp = temp->next;
    }

    cout << "NULL" << endl;
}
int main()
{
    Node *stack = nullptr;

    push(stack, 10);
    push(stack, 20);
    push(stack, 30);

    cout << "Menampilkan isi stack:" << endl;
    show(stack);

    cout << "Pop: " << pop(stack) << endl;

    cout << "Menampilkan sisa stack:" << endl;
    show(stack);

    return 0;
}
```

Screenshots Output



Deskripsi:

Program di atas mengimplementasikan struktur data Stack (Tumpukan) menggunakan Singly Linked List sebagai dasarnya. Berbeda dengan Doubly Linked List yang memiliki pointer prev dan next, struct Node di sini hanya berisi int data dan satu pointer Node *next. Prinsip kerja stack adalah LIFO (Last In, First Out), di mana data terakhir yang masuk adalah data pertama yang keluar. Hal ini dicapai melalui dua operasi utama: push, yang berfungsi seperti "Insert First" dengan menambahkan elemen baru di top (kepala) list, dan pop, yang berfungsi seperti "Delete First" dengan mengambil data dari top, menggeser top ke elemen berikutnya, lalu membebaskan memori node yang lama.
Kode ini juga dilengkapi fungsi utilitas isEmpty untuk memeriksa kekosongan stack dan show untuk mencetak semua elemen dari TOP hingga NULL. Fungsi main mendemonstrasikan alur ini dengan memasukkan (push) tiga angka, menampilkannya, kemudian mengeluarkan (pop) satu angka, dan menampilkan sisa isi stack.

C. Unguided/Tugas (berisi screenshot source code & output program disertai penjelasannya)

Unguided (SOAL 1)

(“stack.h”)

```
#ifndef STACK_H_INCLUDED

#define STACK_H_INCLUDED
```

```
typedef int infotype;

struct Stack {

infotype info[20];

int top;

};

void createStack(Stack &S);

void push(Stack &S, infotype x);

infotype pop(Stack &S);

void printInfo(Stack S);

void balikStack(Stack &S);

#endif
```

(“stack.cpp”)

```
  #include "stack.h"

#include <iostream>

void createStack(Stack &S) {

S.top = 0;

}

void push(Stack &S, infotype x) {

if (S.top < 20) {

S.top = S.top + 1;

S.info[S.top] = x;
```

```
} else {
std::cout << "Stack penuh!" << std::endl;
}
}
infotype pop(Stack &S) {
infotype data;
if (S.top > 0) {
data = S.info[S.top];
S.top = S.top - 1;
return data;
} else {
std::cout << "Stack kosong!" << std::endl;
return -1;
}
}
void printInfo(Stack S) {
std::cout << "[TOP] ";
for (int i = S.top; i >= 1; i--) {
std::cout << S.info[i] << " ";
}
std::cout << std::endl;
```

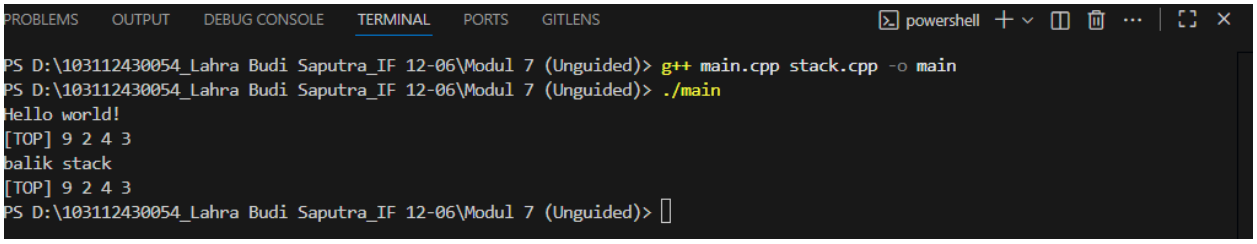
```
}

void balikStack(Stack &S) {

Stack S_temp;

createStack(S_temp);

while (S.top != 0) {

push(S_temp, pop(S));

}

while (S_temp.top != 0) {

push(S, pop(S_temp));

}

}
```

(main.cpp)

```
#ifndef STACK_H_INCLUDED

#define STACK_H_INCLUDED

typedef int infotype;

struct Stack {

infotype info[20];

int top;

};

void createStack(Stack &S);
```

```
void push(Stack &S, infotype x);

infotype pop(Stack &S);

void printInfo(Stack S);

void balikStack(Stack &S);

#endif
```

Screenshots Output:



Deskripsi:

Ketiga file ini bekerja sama untuk membuat dan menguji Abstract Data Type (ADT) Stack menggunakan array.

File stack.h bertindak sebagai header yang mendefinisikan struktur Stack (terdiri dari array info berukuran 20 dan sebuah integer top) serta mendeklarasikan semua prototipe fungsi seperti createStack, push, pop, printInfo, dan balikStack .

File stack.cpp adalah file implementasi yang berisi kode C++ aktual dan logika untuk setiap fungsi yang dideklarasikan di stack.h, seperti bagaimana push menambahkan elemen dengan menaikkan top atau pop mengambil elemen dengan menurunkan top.

Terakhir, file main.cpp adalah program utama yang menjalankan tes; ia membuat sebuah Stack bernama S dan memanggil fungsi-fungsi tadi secara berurutan (misalnya push(S, 3), pop(S), printInfo(S), dll.)

Unguided (SOAL 2)

(“stack.h”)

```
//*Soal 1*//

#ifndef STACK_H_INCLUDED

#define STACK_H_INCLUDED

typedef int infotype;

struct Stack {

infotype info[20];

int top;

};

void createStack(Stack &S);

void push(Stack &S, infotype x);

infotype pop(Stack &S);

void printInfo(Stack S);

void balikStack(Stack &S);

//*Soal 2* Tambahan Code//

void pushAscending(Stack &S, infotype x);

#endif
```

(“stack.cpp”)

```
//*Soal 1*//
```

```
#include "stack.h"

#include <iostream>

void createStack(Stack &S) {

S.top = 0;

}

void push(Stack &S, infotype x) {

if (S.top < 20) {

S.top = S.top + 1;

S.info[S.top] = x;

} else {

std::cout << "Stack penuh!" << std::endl;

}

}

infotype pop(Stack &S) {

infotype data;

if (S.top > 0) {

data = S.info[S.top];

S.top = S.top - 1;

return data;

} else {

std::cout << "Stack kosong!" << std::endl;
```

```
return -1;

}

}

void printInfo(Stack S) {

std::cout << "[TOP] ";

for (int i = S.top; i >= 1; i--) {

std::cout << S.info[i] << " ";

}

std::cout << std::endl;

}

void balikStack(Stack &S) {

Stack S_temp;

createStack(S_temp);

while (S.top != 0) {

push(S_temp, pop(S));

}

S = S_temp;

}

//*Soal 2* Tambahan Code//

void pushAscending(Stack &S, infotype x) {

Stack S_temp;
```

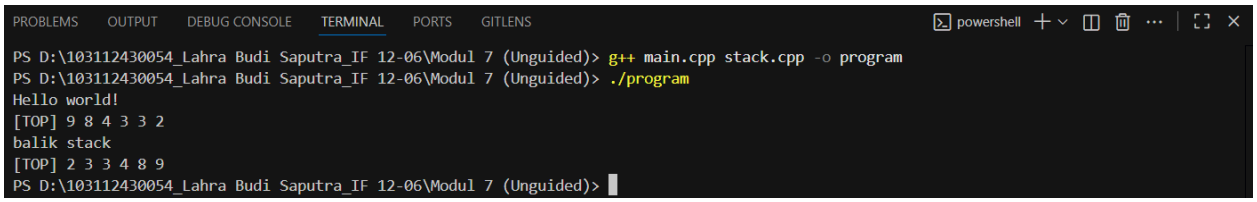
```
createStack(S_temp);

while (S.top > 0 && x < S.info[S.top]) {

push(S_temp, pop(S));

}

push(S, x);

while (S_temp.top > 0) {

push(S, pop(S_temp));

}

}
```

(main.cpp)

```
//*Soal 2* Tambahan Code//

#include <iostream>

#include "stack.h"

using namespace std;

int main()

{

cout << "Hello world!" << endl;

Stack S;

createStack(S);

pushAscending(S,3);
```

```
pushAscending(S,4);

pushAscending(S,8);

pushAscending(S,2);

pushAscending(S,3);

pushAscending(S,9);

printInfo(S);

cout<<"balik stack"<<endl;

balikStack(S);

printInfo(S);

return 0;

}
```

Screenshots Output:



Deskripsi:

Ketiga file ini bekerja sama untuk membuat dan menguji ADT Stack yang terurut.

File stack.h bertindak sebagai header yang mendefinisikan struktur Stack (terdiri dari array info[20] dan integer top) serta mendeklarasikan semua prototipe fungsi, termasuk pushAscending dan balikStack

File stack.cpp menyediakan implementasi atau logika C++ untuk semua fungsi tersebut, ***menambahkan code baru untuk pushAscending*** yang menggunakan stack sementara untuk memasukkan data secara terurut, dan balikStack (yang sudah diperbaiki) yang menggunakan stack sementara dan assignment untuk membalik urutan elemen.

Terakhir, file main.cpp adalah program utama yang menguji kode ini dengan membuat sebuah Stack (S) dan memanggil pushAscending berulang kali, lalu mencetak hasilnya sebelum dan sesudah memanggil balikStack

Unguided (SOAL 3)

(“stack.h”)

```
//*Soal 1*//

#ifndef STACK_H_INCLUDED

#define STACK_H_INCLUDED

typedef int infotype;

struct Stack {

infotype info[20];

int top;

};

void createStack(Stack &S);

void push(Stack &S, infotype x);

infotype pop(Stack &S);

void printInfo(Stack S);

void balikStack(Stack &S);

//*Soal 2* Tambahan Code//

void pushAscending(Stack &S, infotype x);

//*Soal 3* Tambahan Code//
```

```
void getInputStream(Stack &S);

#endif
```

(“stack.cpp”)

```
//*Soal 1*//

#include "stack.h"

#include <iostream>

void createStack(Stack &S) {

S.top = 0;

}

void push(Stack &S, infotype x) {

if (S.top < 20) {

S.top = S.top + 1;

S.info[S.top] = x;

} else {

std::cout << "Stack penuh!" << std::endl;

}

}

infotype pop(Stack &S) {

infotype data;

if (S.top > 0) {
```

```
data = S.info[S.top];

S.top = S.top - 1;

return data;

} else {

std::cout << "Stack kosong!" << std::endl;

return -1;

}

}

void printInfo(Stack S) {

std::cout << "[TOP] ";

for (int i = S.top; i >= 1; i--) {

std::cout << S.info[i] << " ";

}

std::cout << std::endl;

}

void balikStack(Stack &S) {

Stack S_temp;

createStack(S_temp);

while (S.top != 0) {

push(S_temp, pop(S));

}
```

```
S = S_temp;

}

//*Soal 2* Tambahan Code//

void pushAscending(Stack &S, infotype x) {

Stack S_temp;

createStack(S_temp);

while (S.top > 0 && x < S.info[S.top]) {

push(S_temp, pop(S));

}

push(S, x);

while (S_temp.top > 0) {

push(S, pop(S_temp));

}

}

// *Soal 3* Tambahan Code

void getInputStream(Stack &S) {

char ch;

ch = std::cin.get();

while (ch != '\n') {

infotype x = ch - '0';

push(S, x);
```

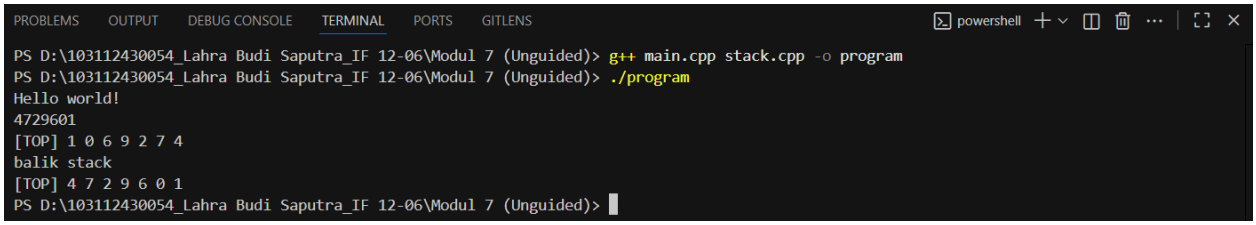
```
ch = std::cin.get();

}

}
```

(main.cpp)

```
//*Soal 3* Tambahan Code / Menggubah file main //

#include <iostream>

#include "stack.h"

using namespace std;

int main()

{

cout << "Hello world!" << endl;

Stack S;

createStack(S);

// Panggil fungsi baru

getInputStream(S);

printInfo(S);

cout << "balik stack" << endl;

balikStack(S);

printInfo(S);

return 0;
```

```
}
```

Screenshots Output:



Deskripsi:

Kode di atas menambahkan prosedur getInputStream ke stack.h dan stack.cpp. Prosedur ini memungkinkan program untuk terus membaca input karakter tunggal dari pengguna menggunakan cin.get(), mengonversinya dari char (misalnya '4') ke infotype (int 4), dan melakukan push ke stack S. Proses ini berlanjut hingga pengguna menekan tombol 'Enter' (karakter \n).

File main.cpp yang baru kemudian memanggil getInputStream(S) setelah createStack(S) untuk mengisi stack dengan data dari pengguna (seperti 4729601). Akhirnya, main.cpp memanggil printInfo dan balikStack untuk mencetak isi stack dan versi terbaliknya.

D. Kesimpulan

Modul ini mengenalkan Anda pada struktur data Stack, yang beroperasi dengan prinsip LIFO (Last In First Out), di mana semua akses hanya terjadi di "Top" . Modul ini menjelaskan dua metode implementasi, yaitu menggunakan pointer dan tabel (array) , dengan fokus latihan praktis pada penggunaan array. Anda telah membangun ADT Stack lengkap, dimulai dengan operasi dasar seperti push dan pop , kemudian menambahkan fungsionalitas yang lebih kompleks seperti pushAscending untuk memasukkan data secara terurut , dan diakhiri dengan getInputStream untuk membaca input pengguna secara dinamis.

E. Referensi

Amaylia, S., Setiabud, V. A., Alvianino, R., Saputra, R. N., Wardhani, H. K., & Suroni, A. (2025). Application of stack data structure in application development. Journal of Advanced Systems Intelligence and Cybersecurity, 1(1). https://journal.unesa.ac.id/index.php/jasic/article/view/44459