INF218
Struktur Data & Algoritma

Stack

Alim Misbullah, S.Si., M.S.

Definisi Stack

Definisi Stack

- Stack merupakan sebuah Abstract Data Type (ADT) yang digunakan pada bahasa pemrograman
- Dinamai dengan **Stack** karena sesuai dengan aplikasi di dunia nyata seperti tumpukan kartu, piring, dan lain-lain

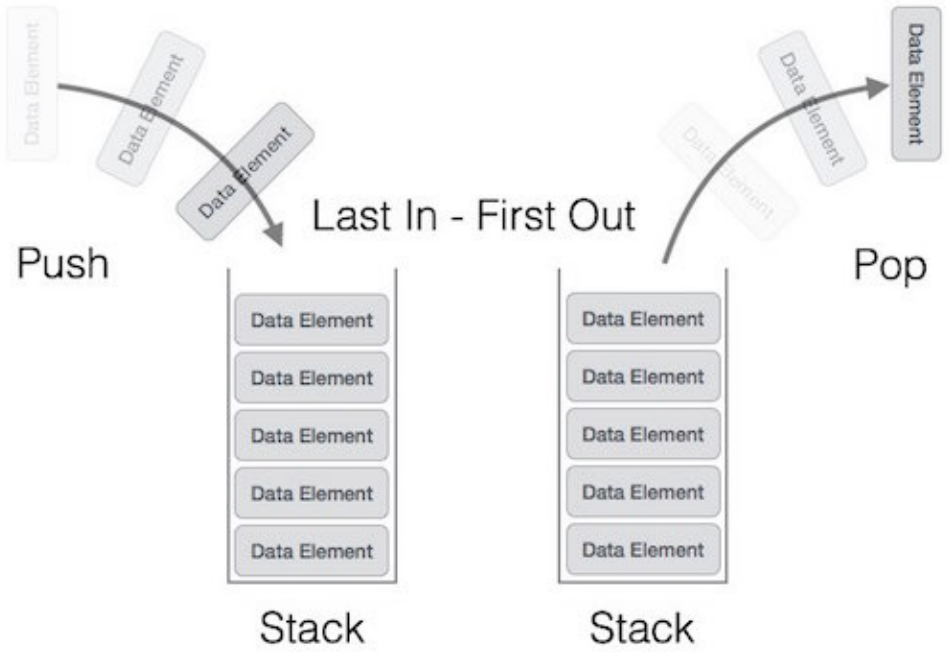
https://www.tutorialspoint.com/data_structures_algorithms/images/stack_example.jpg

- Pada kehidupan nyata, operasi stack hanya berlaku pada satu sisi saja yaitu bagian atas dari sebuah stack, seperti mengambil kartu atau piring dari tumpukan

Definisi Stack

- Sama halnya dengan Stack ADT, operasi yang berlaku juga hanya pada satu sisi saja yaitu bagian atas dari sebuah tumpukan sehingga elemen yang dapat diakses hanya bagian atasnya saja
- Stack disebut sebagai **LIFO (Last In First Out)** data structure yang berarti bahwa elemen yang ditambahkan terakhir akan diakses pertama
- Operasi penambahan elemen disebut dengan **PUSH** dan operasi penghapusan elemen disebut dengan **POP**

Definisi Stack



https://www.tutorialspoint.com/data_structures_algorithms/images/stack_representation.jpg

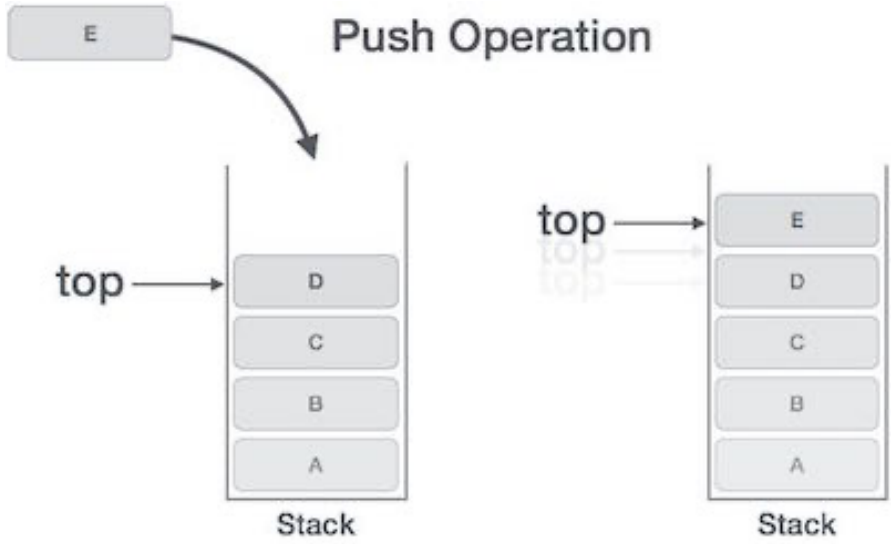
- Stack dapat diimplementasikan menggunakan Array, Struct, Pointer dan Linked List.
- Stack dapat memiliki ukuran yang tetap atau ukuran yang dinamis

Operasi Pada Stack

Operasi PUSH

- Operasi **Push** → Proses untuk menambahkan data baru ke dalam stack. Langkah-langkahnya:
  - Langkah 1: Cek apakah stack sudah penuh (isFull)
  - Langkah 2: Jika stack sudah penuh, tampilkan error/warning dan keluar dari program
  - Langkah 3: Jika stack tidak penuh, arahkan pointer **top** ke bagian kosong dari stack
  - Langkah 4: Tambahkan data elemen ke lokasi stack yang ditunjuk oleh pointer **top**
  - Langkah 5: Return sukses

Operasi PUSH



Algoritma

```
begin procedure push: stack, data

   if stack is full
      return null
   endif

   top ← top + 1
   stack[top] ← data

end procedure
```

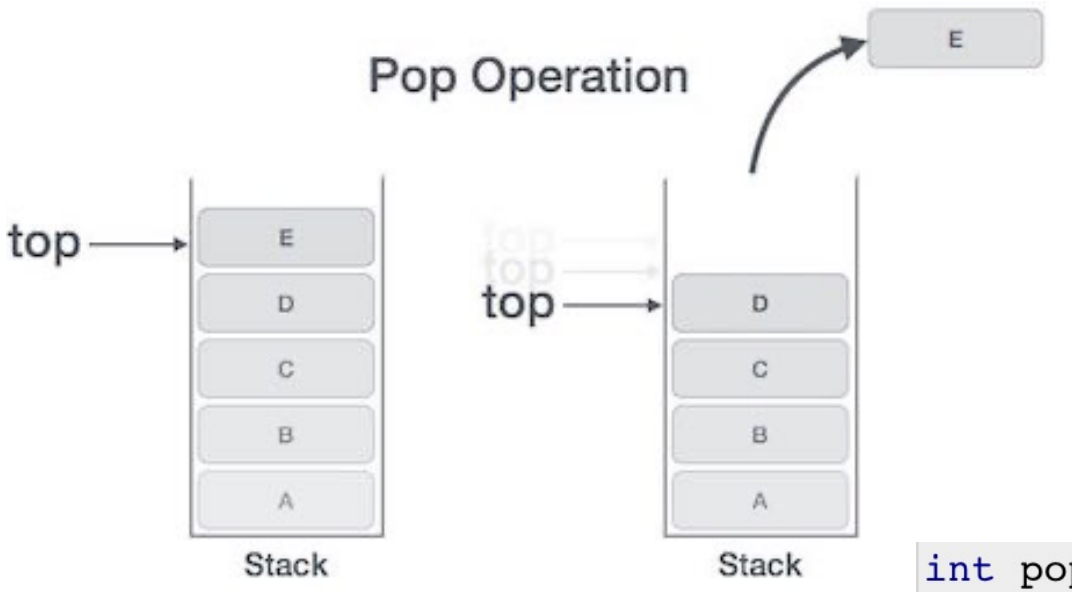
Implementasi

```
void push(int data) {
   if(!isFull()) {
      top = top + 1;
      stack[top] = data;
   } else {
      printf("Could not insert data, Stack is full.\n");
   }
}
```

Operasi POP

- Operasi **POP** → Proses untuk menghapus sebuah elemen di dalam stack.
- Pada implementasi **POP** dengan array, data elemen tidak dihapus secara permanen, namun **top** diturunkan ke posisi dibawahnya untuk menunjuk ke nilai selanjutnya.
- Sedangkan pada linkedlist, **POP** akan menghapus data elemen dan membatalkan alokasi memori
- Langkah-Langkah pada operasi **POP** adalah:
  - Langkah 1: Mengecek apakah stack kosong
  - Langkah 2: Jika stack kosong, tampilkan error/warning dan keluar dari program
  - Langkah 3: Jika stack tidak kosong, akses data elemen yang ditunjuk oleh **top**
  - Langkah 4: Kurangi nilai **top** sebanyak 1
  - Langkah 5: Return sukses

Operasi POP



Algoritma

```
begin procedure pop: stack

   if stack is empty
      return null
   endif

   data ← stack[top]
   top ← top - 1
   return data

end procedure
```

```
int pop(int data) {

   if(!isempty()) {
      data = stack[top];
      top = top - 1;
      return data;
   } else {
      printf("Could not retrieve data, Stack is empty.\n");
   }
}
```

Fungsi pada Stack

- Untuk menggunakan stack secara efisien, kita juga perlu mengecek status dari sebuah stack. Beberapa fungsi yang digunakan untuk mengecek status sebuah stack adalah sebagai berikut:
  - peek() → mendapatkan elemen data teratas dari stack, tanpa menghapusnya
  - isFull() → mengecek apakah stack penuh
  - isEmpty() → mengecek apakah stack kosong
- Pada stack, sebuah pointer selalu akan menunjuk ke elemen data terakhir yang dimasukkan ke dalam stack, sehingga pointer tersebut dinamai dengan **top**

Fungsi pada Stack

Algoritma dari peek()

```
begin procedure peek
   return stack[top]
end procedure
```

Implementasi dari peek()

```
int peek() {
   return stack[top];
}
```

Algoritma dari isFull()

```
begin procedure isfull

   if top equals to MAXSIZE
      return true
   else
      return false
   endif

end procedure
```

Implementasi dari isFull()

```
bool isfull() {
   if(top == MAXSIZE)
      return true;
   else
      return false;
}
```

Algoritma dari isEmpty()

```
begin procedure isempty

   if top less than 1
      return true
   else
      return false
   endif

end procedure
```

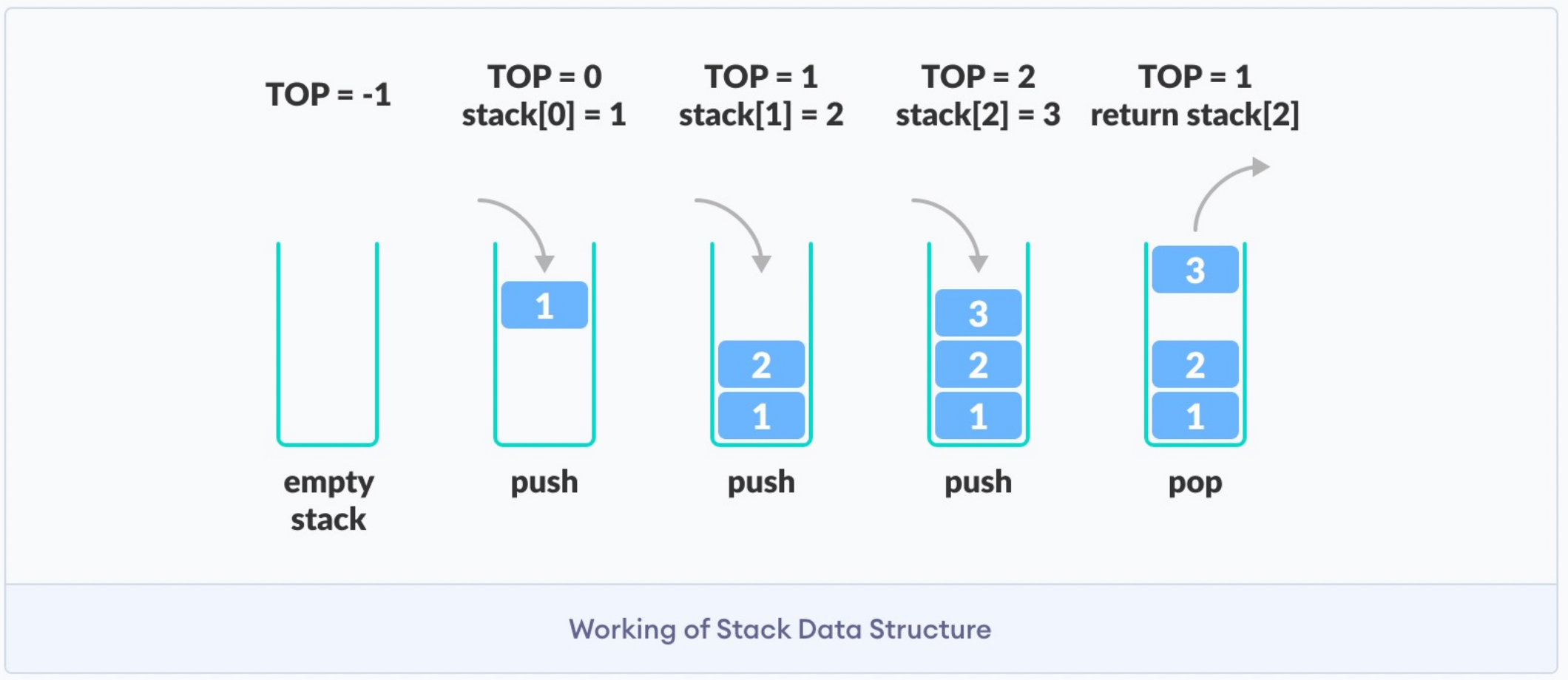
Implementasi dari isEmpty()

```
bool isempty() {
   if(top == -1)
      return true;
   else
      return false;
}
```

Proses Kerja Stack

1. Sebuah pointer disebut sebagai **TOP** digunakan untuk selalu menunjuk data elemen pada bagian atas stack
2. Ketika menginisialkan sebuah stack, kita memberikan nilai -1 untuk **TOP** sehingga kita dapat mengecek jika stack tersebut kosong dengan membandingkan nilai **TOP == -1**
3. Pada saat memasukkan sebuah data elemen ke stack, kita menaikkan nilai **TOP** dan tempatkan data elemen tersebut pada posisi yang ditunjuk oleh **TOP**
4. Pada saat mengeluarkan sebuah data elemen dari stack, kita mengembalikan data elemen yang ditunjuk oleh **TOP** dan menurunkan nilai dari **TOP** tersebut
5. Sebelum memasukkan data elemen ke stack, kita mengecek jika stack sudah penuh atau tidak
6. Sebelum mengeluarkan data elemen dari stack, kita mengecek jika stack sudah kosong

Proses Kerja Stack



Working of Stack Data Structure

Demo Implementasi Stack

- Bahasa C
  - https://www.tutorialspoint.com/data_structures_algorithms/stack_program_in_c.htm
  - https://www.programiz.com/dsa/stack#c-code
- Java
  - https://www.programiz.com/dsa/stack#java-code
- Python
  - https://www.programiz.com/dsa/stack#python-code

Implementasi Stack

- Menggunakan Array
  - https://www.javatpoint.com/ds-array-implementation-of-stack
- Menggunakan Linked List
  - https://www.javatpoint.com/ds-linked-list-implementation-of-stack

Expression Parsing

Definisi Parsing Ekspresi

- Cara untuk menulis sebuah ekspresi aritmatika dikenal dengan istilah **notasi**. Sebuah expresi aritmatika dapat ditulis dalam 3 cara untuk ekspresi aritmatika yang sama tanpa mengubah hasil operasi dari aritmatika tersebut, yaitu:
  - Infix Notation
  - Prefix (Polish) Notation
  - Postfix (Reverse-Polish) Notation

Definisi Parsing Ekspresi

- **Infix Notation** → Operator diletakkan diantara operand. Ini akan memudahkan manusia dalam membaca, menulis dan menghitung ekspresi aritmatika tersebut. Namun, notasi ini akan sulit ketika diimplementasikan pada komputer karena akan menghabiskan banyak waktu dan memory.
  - Contoh infix notation: **a + b + c**
- **Prefix Notation** → Operator diletakkan tepat sebelum operand. Prefix notation juga dikenal dengan nama **Polish Notation**
  - Contoh prefix (polish) notation: **+ab** → infix notationnya **a + b**

Definisi Parsing Ekspresi

- **Postfix Notation** → Operator diletakkan tepat setelah operand. Notasi ini juga dikenal dengan istilah **Reversed Polish Notation**
  - Contoh postfix (reverse-polish) notation: **ab+** → infix notation **a + b**

Sr.No.	Infix Notation	Prefix Notation	Postfix Notation
1	a + b	+ a b	a b +
2	(a + b) * c	* + a b c	a b + c *
3	a * (b + c)	* a + b c	a b c + *
4	a / b + c / d	+ / a b / c d	a b / c d / +
5	(a + b) * (c + d)	* + a b + c d	a b + c d + *
6	((a + b) * c) - d	- * + a b c d	a b + c * d -

Operasi Parsing Expression

- **Precedence (Hak Lebih Tinggi)** → Ketika sebuah operand berada diantara dua operator berbeda, maka operator yang melakukan operasi lebih dulu adalah yang memiliki hak lebih tinggi
  - Contoh: **a + b * c → a + (b * c) → Prefix Notation: *c+ab**
- **Associativity (Asosiatif)** → Jika dua operator memiliki hak yang sama, maka operasinya ditentukan oleh sifat asosiatif
  - Contoh: **a + b -c → (a + b) – c**

Operasi Parsing Expression

Sr.No.	Operator	Precedence	Associativity
1	Exponentiation ^	Highest	Right Associative
2	Multiplication (∗) & Division (/)	Second Highest	Left Associative
3	Addition (+) & Subtraction (−)	Lowest	Left Associative

Implementasi Parsing Expression

- Postfix Evaluation Algorithm

```
Step 1 - scan the expression from left to right
Step 2 - if it is an operand push it to stack
Step 3 - if it is an operator pull operand from stack and perform operation
Step 4 - store the output of step 3, back to stack
Step 5 - scan the expression until all operands are consumed
Step 6 - pop the stack and perform operation
```

- Contoh C code untuk Postfix Evaluation
  - https://www.tutorialspoint.com/data_structures_algorithms/expression_parsing_using_stack.htm

Q & A

Referensi

- https://www.tutorialspoint.com/data_structures_algorithms/stack_algorithm.htm
- https://www.javatpoint.com/data-structure-stack
- https://www.programiz.com/dsa/stack
- https://www.tutorialspoint.com/data_structures_algorithms/expression_parsing.htm