MATERI MENGENAI STACK DAN QUEUE

NAMA : PUTRI AULIA NUR SALSABILA

NIM : H071211046

PRODI : SISTEM INFORMASI A

---

1. STACK

Definisi Stack

Stack atau tumpukan adalah suatu struktur data yang terbentuk dari barisan hingga yang terurut dari satuan data. Pada Stack, penambahan dan penghapusan elemennya hanya dapat dilakukan pada satu posisi, yaitu posisi akhir stack. Posisi ini disebut Puncak atau Top dari stack. Berkebalikan dengan Queue atau antrian yang mana data yang masuk terlebih dahulu akan diproses terlebih dahulu.

Terdapat dua operasi dasar dalam stack, yaitu push dan pop. Operasi push digunakan untuk memasukkan data ke dalam stack sedangkan operasi pop digunakan untuk mengeluarkan data dari stack. Stack dapat penuh apabila metode penyimpanan datanya menggunakan array klasik atau array yang didefinisikan panjangnya. Sedangkan apabila menggunakan ArrayList, maka tumpukan bisa lebih besar kapasitas (maksimal hingga nilai terbesar integer). Perlu diketahui, dalam struktur data stack ada dua kondisi yang perlu dihindari, yaitu underflow dan overflow.

a. Stack underflow, yaitu keadaan dimana kita mencoba mengakses atau menghapus elemen data pada stack yang kosong.
b. Stack overflow, yaitu keadaan di mana ruang memori yang dialokasikan untuk struktur data stack sudah penuh namun masih dilakukan operasi penyisipan elemen.

Jenis-Jenis Stack

Berdasarkan kemampuan menyimpan data, struktur data stack dapat dibagi menjadi 2 jenis, yaitu:

a) Register stack

Register stack merupakan stack yang hanya mampu menampung data dalam jumlah yang kecil. Kedalaman maksimum pada register stack cenderung dibatasi karena ukuran unit memorinya sangat kecil dibandingkan dengan memory stack.

b) Memory stack

Pada stack jenis ini, kedalaman dari stack cukup fleksibel dan mampu menangani dalam dalam skala yang lebih besar dibandingkan jenis sebelumnya.

Karakteristik Stack

Struktur data stack memiliki ciri sebagai berikut:

a) Stack diimplementasikan dengan struktur data array atau linked list.
b) Mengikuti prinsip operasi (Last In, First Out), yaitu elemen yang dimasukkan pertama akan muncul terakhir dan sebaliknya.
c) Penyisipan dan penghapusan terjadi di satu ujung yaitu dari atas tumpukan.
d) Apabila ruang memori yang dialokasikan untuk struktur data stack sudah penuh namun masih dilakukan operasi penyisipan elemen maka akan terjadi stack overflow.
e) Apabila struktur data tidak memiliki elemen data atau kosong, namun tetap dilakukan operasi penghapusan maka akan terjadi stack underflow.

Operasi-Operasi Dasar Pada Stack

Ada beberapa operasi dasar yang bisa dilakukan terhadap struktur data stack. Operasi-operasi tersebut meliputi:

a) Push untuk menyisipkan elemen ke bagian atas stack.
b) Pop untuk menghapus elemen atas dari stack.
c) IsEmpty untuk memeriksa apakah stack kosong.
d) IsFull untuk memerika apakah stack sudah penuh.
e) Peek untuk mendapatkan nilai elemen teratas tanpa menghapusnya.

Source Code Stack:

```
// Stack implementation in C++

#include <stdlib.h>
#include <iostream>

using namespace std;

#define MAX 10
int size = 0;

// Creating a stack
struct stack {
  int items[MAX];
  int top;
};
typedef struct stack st;

void createEmptyStack(st *s) {
  s->top = -1;
}
// Check if the stack is full
int isfull(st *s) {
  if (s->top == MAX - 1)
    return 1;
  else
    return 0;
}

// Check if the stack is empty
int isempty(st *s) {
  if (s->top == -1)
    return 1;
  else
    return 0;
}

// Add elements into stack
void push(st *s, int newitem) {
  if (isfull(s)) {
    cout << "STACK FULL";
  } else {
    s->top++;
    s->items[s->top] = newitem;
  }
  size++;
}
```

```
// Remove element from stack
void pop(st *s) {
  if (isempty(s)) {
    cout << "\n STACK EMPTY \n";
  } else {
    cout << "Item popped= " << s->items[s->top];
    s->top--;
  }
  size--;
  cout << endl;
}

// Print elements of stack
void printStack(st *s) {
  printf("Stack: ");
  for (int i = 0; i < size; i++) {
    cout << s->items[i] << " ";
  }
  cout << endl;
}
// Driver code
int main() {
  int ch;
  st *s = (st *)malloc(sizeof(st));

  createEmptyStack(s);

  push(s, 1);
  push(s, 2);
  push(s, 3);
  push(s, 4);

  printStack(s);

  pop(s);

  cout << "\nAfter popping out\n";
  printStack(s);
}
```

2. QUEUE

Queue atau antrian adalah struktur data dimana data yang pertama kali dimasukkan adalah data yang pertama kali bisa dihapus. Atau bisa juga disebut dengan struktur data yang menggunakan mekanisme FIFO (First In, First Out). Berbeda dengan struktur data stack yang menyimpan data secara bertumpuk dimana hanya terdapat satu ujung yang terbuka untuk melakukan operasi data, struktur data queue justru disusun secara horizontal dan terbuka di kedua ujungnya. Ujung pertama (head) digunakan untuk menghapus data sedangkan ujung lainnya (tail) digunakan untuk menyisipkan data.

Jenis-Jenis Queue

Secara umum ada 4 jenis struktur data queue, meliputi:

a) Simple Queue

Simple queue adalah struktur data queue paling dasar di mana penyisipan item dilakukan di simpul belakang (rear atau tail) dan penghapusan terjadi di simpul depan (front atau head).

b) Circular Queue

Pada circular queue, simpul terakhir terhubung ke simpul pertama. Queue jenis ini juga dikenal sebagai ring buffer karena semua ujungnya terhubung ke ujung yang lain. Penyisipan terjadi di akhir antrian dan penghapusan di depan antrian.

c) Priority Queue

Priority queue adalah strruktur data queue dimana simpul akan memiliki beberapa prioritas yang telah ditentukan. Simpul dengan prioritas terbesar akan menjadi yang pertama dihapus dari antrian. Sedangkan penyisipan item terjadi sesuai urutan kedatangannya.

d) Double-Ended Queue (Dequeue)

Dalam double-ended queue (dequeue), operasi penyisipan dan penghapusan dapat terjadi di ujung depan dan belakang dari queue.

Karakteristik Queue

Queue memiliki berbagai karakteristik sebagai berikut:

a) Queue adalah struktur FIFO (First In, First Out).
b) Untuk menghapus elemen terakhir dari Queue (semua elemen yang dimasukkan sebelum elemen tersebut harus dihilangkan atau dihapus).
c) Queue adalah daftar berurutan dari elemen-elemen dengan tipe data yang serupa.

Operasi-Operasi Dasar Pada Queue

Queue adalah struktur data abstrak (ADT) yang memungkinkan operasi berikut:

a) Enqueue untuk menambahkan elemen ke akhir antrian.
b) Dequeue untuk menghapus elemen dari depan antrian.
c) IsEmpty untuk memeriksa apakah antrian kosong.
d) IsFull untuk memeriksa apakah antrian sudah penuh.
e) Peek untuk mendapatkan nilai bagian depan antrian tanpa menghapusnya.
f) Initialize untuk membuat antrian baru tanpa elemen data (kosong).

Namun, secara umum antrian memiliki 2 operasi utama, yaitu enqueue dan dequeue.

Fungsi dan Kegunaan Queue

Berikut ini adalah beberapa fungsi queue yang paling umum dalam struktur data:

a) Queue banyak digunakan untuk menangani lalu lintas (traffic) situs web.
b) Membantu untuk mempertahankan playlist yang ada pada aplikasi media player.
c) Queue digunakan dalam sistem operasi untuk menangani interupsi.
d) Membantu dalam melayani permintaan pada satu sumber daya bersama, seperti printer, penjadwalan tugas CPU, dll.
e) Digunakan dalam transfer data asinkronus misal pipeline, IO file, dan socket.

Source Code Queue:

```
// Queue implementation in C++

#include <iostream>
#define SIZE 5

using namespace std;

class Queue {
   private:
  int items[SIZE], front, rear;

   public:
  Queue() {
    front = -1;
    rear = -1;
  }

  bool isFull() {
    if (front == 0 && rear == SIZE - 1) {
      return true;
    }
    return false;
  }

  bool isEmpty() {
    if (front == -1)
      return true;
    else
      return false;
  }

  void enQueue(int element) {
    if (isFull()) {
      cout << "Queue is full";
    } else {
      if (front == -1) front = 0;
      rear++;
      items[rear] = element;
      cout << endl
         << "Inserted " << element << endl;
    }
  }
```

```
  int deQueue() {
    int element;
    if (isEmpty()) {
      cout << "Queue is empty" << endl;
      return (-1);
    } else {
      element = items[front];
      if (front >= rear) {
        front = -1;
        rear = -1;
      } /* Q has only one element, so we reset the queue after deleting it. */
      else {
        front++;
      }
      cout << endl
         << "Deleted -> " << element << endl;
      return (element);
    }
  }

  void display() {
    /* Function to display elements of Queue */
    int i;
    if (isEmpty()) {
      cout << endl
         << "Empty Queue" << endl;
    } else {
      cout << endl
         << "Front index-> " << front;
      cout << endl
         << "Items -> ";
      for (i = front; i <= rear; i++)
        cout << items[i] << "  ";
      cout << endl
         << "Rear index-> " << rear << endl;
    }
  }
};
```

```
int main() {
  Queue q;

  //deQueue is not possible on empty queue
  q.deQueue();

  //enQueue 5 elements
  q.enQueue(1);
  q.enQueue(2);
  q.enQueue(3);
  q.enQueue(4);
  q.enQueue(5);

  // 6th element can't be added to because the queue is full
  q.enQueue(6);

  q.display();

  //deQueue removes element entered first i.e. 1
  q.deQueue();

  //Now we have just 4 elements
  q.display();

  return 0;
}
```