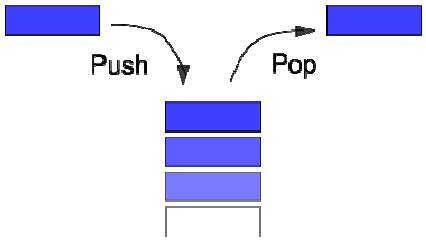
PRAKTIKUM 22
STACK (TUMPUKAN)

A. TUJUAN PEMBELAJARAN

1. Memahami konsep penyimpanan data dengan stack (tumpukan)
2. Memahami operasi pada stack
3. Mampu mengimplementasikan struktur data stack pada pemrograman berbasis obyek

B. DASAR TEORI

Salah satu konsep yang efektif untuk menyimpan dan mengambil data adalah "terakhir masuk sebagai pertama yang keluar" (Last In First Out/FIFO). Dengan konsep ini, pengambilan data akan berkebalikan urutannya dengan penyimpanan data. Stack(tumpukan) adalah sebuah kumpulan data dimana data yang diletakkan di atas data yang lain. Dengan demikian stack adalah struktur data yang menggunakan konsep LIFO. Elemen terakhir yang disimpan dalam stack menjadi elemen pertama yang diambil. Dalam proses komputasi, untuk meletakkan sebuah elemen pada bagian atas stack disebut dengan push. Dan untuk memindahkan dari tempat teratas tersebut, kita melakukan pop.



Gambar 1. Ilustrasi Stack

Ada 2 operasi paling dasar dari stack yang dapat dilakukan, yaitu :

1. Operasi push yaitu operasi menambahkan elemen pada urutan terakhir (paling atas).
2. Operasi pop yaitu operasi mengambil sebuah elemen data pada urutan terakhir dan menghapus elemen tersebut dari stack.

Mengubah Notasi Postfix menjadi Infix dengan Stack

Salah satu penggunaan stack adalah mengubah notasi infix menjadi postfix. Berikut ini adalah algoritma untuk mengubah notasi infix menjadi notasi postfix:

1. Baca ungkapan dalam notasi infix, misalnya S, tentukan panjang ungkapan tersebut, misalnya N karakter, siapkan sebuah stack kosong dan siapkan derajad masing-masing operator, misalnya: ^ berderajad 3, * dan / berderajad 2, + dan – berderajad 1 dan (berderajad 0.
2. Dimulai dari i = 1 sampai N kerjakan langkah-langkah sebagai berikut:
   a. R = S[I]
   b. Test nilai R. Jika R adalah:

operand	:	langsung ditulis
kurung buka	:	push ke dalam tumpukan
kurung tutup	:	pop dan tulis semua isi tumpukan sampai ujung tumpukan = ‘(‘. Pop juga tanda ‘(‘ ini, tetapi tidak usah ditulis
operator	:	jika tumpukan kosong atau derajad R lebih tinggi dibanding derajad ujung tumpukan, push operator ke dalam tumpukan. Jika tidak, pop ujung tumpukan dan tulis; kemudian ulangi pembandingan R dengan ujung tumpukan. Kenudian R di-push

   c. Jika akhir notasi infix telah tercapai, dan tumpukan masih belum kosong, pop semua isi tumpukan dan tulis hasilnya

Untuk memahami algoritma di atas, kita coba mengubah ungkapan berikut, yang ditulis menggunakan notasi infix, menjadi notasi postfix

(A + B) / ((C – D) * E ^ F)

Ilustrasi pengubahan notasi infix di atas menjadi notasi postfix secara lengkap tersaji dalam tabel sebagai berikut:

Tabel 1. Proses Mengubah Notasi Infix menjadi Postfix

Karakter dibaca	Isi Tumpukan	Karakter tercetak	Hasil Notasi Postfix Yang Terbentuk
(	(		
A	(+	A	A
+	(+		
B		B	A B
)		+	A B +
/	/		
(	/ (		
(	/ ((		
C	/ ((	C	A B + C
-	/ ((-		
D	/ ((-	D	A B + C D
)	/ (	-	A B + C D -
*	/ (*		
E	/ (*	E	A B + C D - E
^	/ (* ^		
F	/ (* ^	F	A B + C D - F
)	/ (*	^	A B + C D – F ^
	/ (	*	A B + C D – F ^ *
	/		
		/	A B + C D – F ^ *

Dari ilustrasi di atas, bisa kita lihat bahwa notasi postfix dari ungkapan:

(A + B) / ((C – D) * E ^ F)

adalah

A B + C D – F ^ *

Tabel 2. Contoh Infix ke Postfix

Infix	**Postfix**
(A + B) / ((C – D) * E ^ F)	A B + C D – F ^ *
a + b * c	abc*+
a * b / c + d	ab*c/d+

a * (b + c)	abc+*
a^b^c	abc^^

C. TUGAS PENDAHULUAN

1. Ubahlah ekspresi infix dibawah ini menjadi ekspresi postfix.
   a. (A * B) + (C -D)
   b. (A - B) - (C * D) +E
   c. A* (B - C) - (D * E)

D. PERCOBAAN

Percobaan 1 : *Mengubah notasi infix menjadi postfix*

```
import java.util.Stack;

public class InfixToPostfix {

    String infixExp = "";
    String postfixExp = "";
    Stack<Character> s = new Stack<Character>();

    public void setInfixExp(String infixExp) {
        this.infixExp = infixExp;
    }

    public boolean isOperator(char ch) {
        if (ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch
== '^') {
            return true;
        }
        return false;
    }

    public int degreeOp(char op) {
        if (op == '+' || op == '-') {
            return 1;
        } else if (op == '*' || op == '/') {
            return 2;
        } else if (op == '^') {
            return 3;
        } else {
            return 0;
        }
    }

    public String toPostfix() {
        char ch;
```

```
        for (int i = 0; i < infixExp.length(); i++) {
            ch = infixExp.charAt(i);
            if (isOperator(ch)) {
                if (s.isEmpty() || degreeOp(ch) >
degreeOp(s.peek())) { //perbandingan derajat relasi
                    s.push(ch);
                } else {
                    postfixExp += s.pop();
                    do {
                        if (s.isEmpty() || degreeOp(ch) >
degreeOp(s.peek())) {
                            break;
                        } else {
                            //System.out.println(ch);
                            postfixExp += s.pop();
                        }
                    } while (degreeOp(ch) <= degreeOp(s.peek()));
//perbandingan derajat relasi
                    s.push(ch);
                }
            } else if (ch == ')') {
                do {
                    postfixExp += s.pop();
                } while (s.peek() != '(');
                s.pop();
            } else if (ch == '(') {
                s.push(ch);
            } else {
                postfixExp += ch;
            }
        }

        if (!s.isEmpty()) {
            do {
                postfixExp += s.pop();
            } while (!s.isEmpty());
        }
        return postfixExp;
    }
}
```

```
import java.util.Scanner;

public class TestInfixToPostfix {

    public static void main(String[] args) {
        InfixToPostfix itp = new InfixToPostfix();
        String infix;
        Scanner keyIn = new Scanner(System.in);
        //(a+b)/((c-d)*e^f)
        //(A+B)/((C-D)*E^F)
        System.out.print("Infix Expression : ");
```

```
        infix = keyIn.nextLine();
        itp.setInfixExp(infix);
        System.out.println("Postfix Expression : " +
itp.toPostfix());
    }
}
```

E. LATIHAN

Ujilah program yang sudah dibuat dengan notasi infix berikut:

a. (A * B) + (C -D)

b. (A - B) - (C * D) +E

c. A* (B - C) - (D * E)

F. LAPORAN RESMI

Kerjakan hasil percobaan(D) dan latihan(E) di atas dan tambahkan analisa.