LAPORAN TUGAS BESAR II IF-2220
TEORI BAHASA FORMAL DAN AUTOMATA
APLIKASI CFG DAN PDA PADA PENGENALAN EKSPRESI MATEMATIKA

Disusun Oleh:

Kevin Nathaniel Wijaya - 13517072
Irfan Sofyana Putra - 13517078
Louis Cahyadi - 13517126

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2018

BAB I

DESKRIPSI PERSOALAN

Pada Tugas Besar Teori Bahasa Formal dan Automata yang kedua, kami ditugaskan untuk membuat sebuah program kalkulator sederhana. Kalkulator pada tahun 1960an adalah alat yang digunakan untuk melakukan perhitungan-perhitungan sederhana seperti penjumlahan, pengurangan, perkalian, dan pembagian. Sekarang, kalkulator sudah mengalami kemajuan sehingga kalkulator menjadi lebih terjangkau dan lebih banyak variasi, seperti kalkulator *scientific* yang memiliki fungsi-fungsi lebih dibanding kalkulator biasa. Kalkulator pun sekarang sudah mulai terimplementasi di dalam telepon genggam, komputer personal, dan lain-lain.

Pada tugas kali ini, kami diminta untuk mengimplementasi tata bahasa bebas konteks atau *context-free grammar* dan/atau *pushdown automata* untuk membuat sebuah kalkulator sederhana. Kalkulator ini harus cukup pintar sehingga kalkulator tersebut dapat mendeteksi kesalahan expresi dan menampilkan pesan *syntax error*, atau mendeteksi adanya perhitungan yang tidak dapat dilakukan (pembagian dengan 0) dan menampilkan pesan *math error*. Contoh dari *syntax error* adalah 3*+-12/(57), yaitu ketika ada tanda-tanda yang bersebelahan, dan contoh dari *math error* adalah 3/0.

Pada program kalkulator ini, masukan terdiri dari simbol aritmatika biasa (+, -, *, /), perpangkatan (^), tanda negatif (-), angka (0, 1, 2, 3, 4, 5, 6, 7, 8, 9), desimal (.), dan tanda kurung (()). Operator hanya terdiri dari simbol aritmatika biasa, tidak mengandung huruf-huruf (e, pi, dan lain-lain). Tidak ada spasi antar token. Untuk penulisan akar, digunakan fungsi pangkat dengan 1/faktor akar. Operasi yang dilakukan mengikuti urutan semestinya, yaitu yang ada di dalam tanda kurung, operasi pangkat, operasi kali dan bagi, dan operasi tambah dan kurang. Operasi juga dilakukan secara berurutan dari kiri.

Lebih lanjut kami juga mengerjakan bagian bonus dari tugas ini, yaitu implementasi bilangan komplex yang ditandai dengan adanya symbol “i” yang merepresentasikan √-1, sehingga program dapat menerima masukan expresi bilangan kompleks dan dapat menghitungnya serta mengembalikan nilai dalam bentuk ekspresi kompleks.

Asumsi kami dalam pembuatan program ini adalah:

1. Tanda “+” (plus) dianggap sebuah operator penjumlahan bukan sebagai *sign*, sehingga masukan “3*+5” tidak dianggap valid dan akan mengeluarkan pesan “Syntax error”.
2. Tanda “-” (minus) dianggap sebagai operator pengurangan dan juga sebagai *sign*, sehingga masukan “3*-5” dianggap valid.

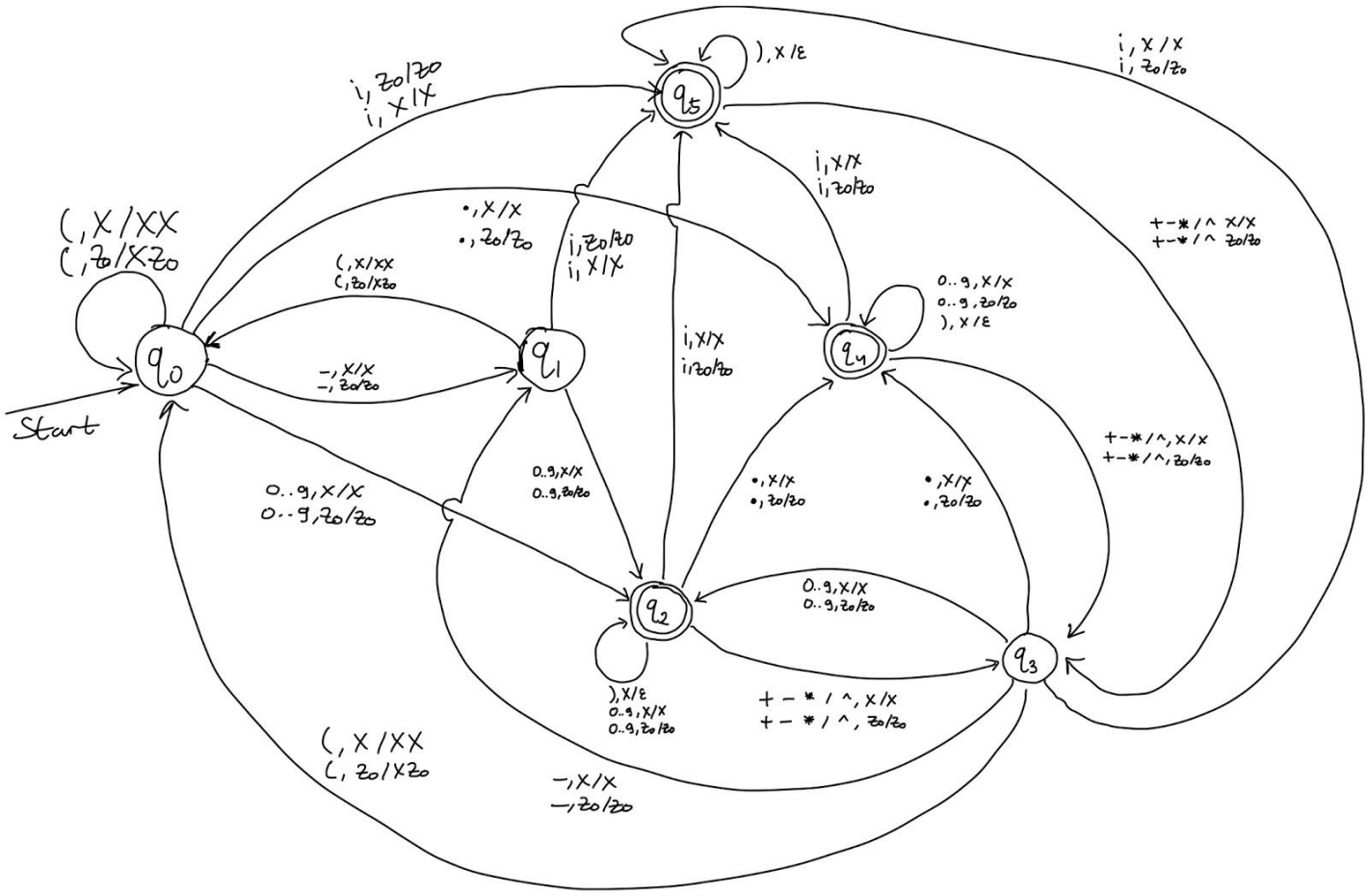
3. Tanda “.” (titik) boleh diletakan di depan dan di belakang bilangan, sebagai contoh “3.” akan dihitung sebagai “3.0”, dan “.3” akan dihitung sebagai “0.3”.

BAB II

DESKRIPSI SOLUSI

Pada pengerjaan tugas ini kelompok kami membagi menjadi dua program, yang pertama ialah pengecekan terhadap *string input* dari user apakah merupakan ekspresi matematika yang valid, dan yang berikutnya ialah pengenalan ekspresi matematika serta perhitungan matematikanya. Pada pengecekan input user digunakan sebuah *PushDown Automata* (PDA), setelah sebuah ekspresi *valid,* maka input user tersebut akan diubah menjadi beberapa *token* lalu proses perhitungan pun akan dijalankan oleh *Context free grammar* (CFG) yang diimplementasikan melalui algoritma *recursive descent parser*.

2.1 *PushDown Automata* (PDA) yang digunakan



String akan dianggap valid jika berakhir di final state, yaitu q_2, q_4, q_5, dan top of stacknya adalah Z_0.

2.2 *Context Free Grammar* (CFG)

Context Free Grammar(CFG) yang digunakan pada aplikasi kalkulator ini diperlukan untuk menghitung nilai dari ekspresi matematika yang sudah teruji valid oleh PDA.

CFG yang digunakan adalah sebagai berikut:

- *Terminal* = {n}
- *Variables* = {E, T, F, I}
- *Start symbol* = E
- *Productions* =
  E -> T | T+E | T- E
  T -> F | F*T | F/T | FiT
  F -> I | I^F | -I | -I^F
  I -> n | (I)

Terminal dari CFG ini adalah himpunan semua bilangan kompleks. Untuk mengimplementasikan CFG ini, kelompok kami memilih untuk menggunakan algoritma *recursive descent parser* dengan bantuan struktur data *stack*.

BAB III

SOURCE CODE

Source code dari program kalkulator kami terdiri dari beberapa file:

- boolean.h : Kode untuk mendefinisi tipe data boolean pada C.
- main.c: Kode utama dari program kalkulator, menerima masukan dan mengeluarkan hasil kalkulasi.
- mesintoken.c dan mesintoken.h : Kode header dan implementasi untuk membaca string dan membagi string menjadi bagian-bagian tertentu.
- pda.c dan pda.h : Kode header dan implementasi untuk memvalidasi masukan dari pengguna menggunakan *Pushdown Automata*.
- stack.c dan stack.h : Kode header dan implementasi untuk menggunakan stack pada perhitungan.
- stackchar.c dan stackchar.h : Kode header dan implementasi untuk menggunakan stack yang telah dimodifikasi pada perhitungan.

Program kami dapat dikompilasi dengan menjalankan perintah “gcc -o main main.c pda.c stack.c stackchar.c mesintoken.c”.

Berikut adalah source code dari file boolean.h:

```
#ifndef boolean_H
#define boolean_H
#define true 1
#define false 0
#define boolean unsigned char
#endif
```

Berikut adalah source code dari file program utama main.c:

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include "mesintoken.h"
#include "stack.h"
#include "pda.h"
#include "stackchar.h"
```

```
#include "boolean.h"
#include "complex.h"

/* Kumpulan variabel yang digunakan untuk menghitung ekspresi */
Token CToken[10005];
int NToken;
Stackt tok;
boolean IsValid;

/* Kumpulan fungsi yang digunakan untuk menghitung ekspresi */
/* fungsi-fungsi di bawah ini juga merupakan implementasi dari recursive descent
parser algorithm yang menggunakan CFG */
double complex parse_item();
double complex parse_factor();
double complex parse_term();
double complex parse_expression();

/*fungsi dibawah ini berfungsi untuk mengecek apakah suatu operasi matematika
tersebut valid atau tidak */
/* contoh : 2 + 3 * 5 -> valid
            9 / (9 - 3*3) -> tidak valid
            0 ^ 0 -> tidak valid
*/
boolean cekvalid(double complex bil1, char opr, double complex bil2)
{
    if ((opr == '/') && (creal(bil2) == 0) && (cimag(bil2) == 0))
    {
        return false;
    }
    else if ((creal(bil1) == 0) && (cimag(bil1) == 0) && (opr == '^') &&
(creal(bil2) == 0) && (cimag(bil2) == 0))
    {
        return false;
    }
    else
    {
        return true;
    }
```

```
}

/*implementasi dari parse_item */
double complex parse_item()
{
    Infotype t, s;
    Popst(&tok, &t);
    if (t.tkn == 'b')
        return t.val;
    else if (t.tkn == 'i')
        return I;
    double complex expr;
    expr = parse_expression();
    Popst(&tok, &s);
    return expr;
}

/*implementasi dari parse_item */
double complex parse_factor()
{
    Infotype t;
    Popst(&tok, &t);
    Pushst(&tok, t);
    double complex sign, test;
    if (t.tkn == '-')
        sign = -1;
    else
        sign = 1;
    if (t.tkn == '+' || creal(sign) < 0)
        Popst(&tok, &t);
    double complex result;
    result = parse_item();
    Popst(&tok, &t);
    Pushst(&tok, t);
    while (t.tkn == '^')
    {
        Popst(&tok, &t);
        if (t.tkn != '^')
```

```
        {
            Pushst(&tok, t);
            break;
        }
        double complex rhs;
        rhs = parse_factor();
        if (cekvalid(result, '^', rhs))
        {
            result = cpow(result, rhs);
        }
        else
        {
            IsValid = false;
            return 0;
        }
        Popst(&tok, &t);
        Pushst(&tok, t);
    }
    return result * sign;
}

/*implementasi dari parse_term */
double complex parse_term()
{
    double complex result;
    result = parse_factor();
    Infotype t;
    Popst(&tok, &t);
    if (t.tkn != '*' && t.tkn != '/')
        Pushst(&tok, t);
    while (t.tkn == '*' || t.tkn == '/' || t.tkn == 'i')
    {
        double complex rhs;
        rhs = parse_factor();
        if (cekvalid(result, t.tkn, rhs))
        {
            if (t.tkn == '/')
                result /= rhs;
```

```
            else if (t.tkn == '*')
                result *= rhs;
            else
                result *= rhs;
        }
        else
        {
            IsValid = false;
            return 0;
        }
        Popst(&tok, &t);
        if (t.tkn != '*' && t.tkn != '/' && t.tkn != 'i')
        {
            Pushst(&tok, t);
        }
    }
    return result;
}

/*implementasi dari parse_expression */
double complex parse_expression()
{
    double complex result;
    result = parse_term();
    Infotype t;
    Popst(&tok, &t);
    if (t.tkn != '+' && t.tkn != '-')
        Pushst(&tok, t);
    while (t.tkn == '+' || t.tkn == '-')
    {
        double complex rhs;
        rhs = parse_term();
        if (t.tkn == '+')
            result += rhs;
        else
            result -= rhs;
        Popst(&tok, &t);
        if (t.tkn != '+' && t.tkn != '-')
```

```
        {
            Pushst(&tok, t);
        }
    }
    return result;
}


/* main program */
int main(){
    printf("Selamat datang pada program kalkulator sederhana!\n");
    printf("Ekspresi yang ingin dihitung (masukan \"exit\" untuk keluar): \n");
    masukan inputuser;
    gets(inputuser);
    while (strcmp("exit", inputuser) != 0){
        while (!(validasi(inputuser)))
        {
            printf("Syntax error!\n");
            printf("Ekspresi yang ingin dihitung (masukan \"exit\" untuk keluar):
            \n");
            gets(inputuser);
        }
        if (validasi(inputuser))
        {
            NToken = 0;
            SplitToken(&inputuser, &CToken, &NToken);
            CreateEmptyst(&tok);
            IsValid = true;
            for (int i = 0; i < NToken; i++)
            {
                Pushst(&tok, CToken[i]);
            }
            tok = Reverse(tok);
            double complex ans;
            ans = parse_expression();
            if (IsValid)
            {
                if (floorf(fabs(cimag(ans)) * 1000000)/1000000 == 0)
```

```
                    printf("Hasil: %.6f\n", creal(ans));
                else if (cimag(ans) > 0)
                    printf("Hasil: %.6f+%.6fi\n", creal(ans), cimag(ans));
                else if (cimag(ans) < 0){
                    printf("Hasil: %.6f%.6fi\n", creal(ans), cimag(ans));
                }
            }
            else
                printf("Math error!\n");
        }
        printf("Ekspresi yang ingin dihitung (masukan \"exit\" untuk keluar): \n");
        gets(inputuser);
    }
    return 0;
}
```

Berikut adalah source code dari file header mesintoken.h:

```
#ifndef __MESIN_TOKEN_H_
#define __MESIN_TOKEN_H_

#include <complex.h>

typedef struct{
       double complex val;
       char tkn;
}Token;

void SplitToken(char (*s)[], Token (*Arr)[], int *Ntoken);

#endif
```

Berikut adalah source code dari file mesintoken.c, implementasi dari file header mesintoken.h:

```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
#include <complex.h>
#include "mesintoken.h"
```

```
int NToken;

void SplitToken(char (*s)[], Token (*Arr)[], int *NToken){
    int i = 0;
    while (i < strlen(*s)){
        if ((*s)[i] == ' ') {
            while (i < strlen(*s) && (*s)[i] == ' ') i++;
        }
        else if ((*s)[i] == '+' || (*s)[i] == '-' || (*s)[i] == '*' || (*s)[i] ==
        '/' || (*s)[i] == '^' || (*s)[i] == '(' || (*s)[i] == ')') {
            (*Arr)[*NToken].tkn = (*s)[i];
            (*Arr)[*NToken].val = -1.00;
            (*NToken)++;
            i++;
         }
        else if ((*s)[i] == 'i'){
            (*Arr)[*NToken].tkn = 'i';
            (*Arr)[*NToken].val = I;
            (*NToken)++;
            i++;
        }
        else {
            double complex tmp = 0.00;
            int dotpos = 0;
            int cnt = 0;
            while (i < strlen(*s) && (((*s)[i] >= '0' && (*s)[i] <= '9') ||
            (*s)[i] == '.')){
                if ((*s)[i] == '.') {
                    dotpos = i;
                    cnt++;
                }
                else tmp = 10 * tmp + ((*s)[i] - '0');
                i++;
            }
            if (cnt != 0) {
                dotpos = i - dotpos - 1;
                while (dotpos > 0){
```

```
                    tmp /= 10.0;
                    dotpos--;
                }
            }
            (*Arr)[*NToken].tkn = 'b';
            (*Arr)[*NToken].val = tmp;
            (*NToken)++;
        }
    }
}
```

Berikut adalah source code dari file header pda.h:

```
#ifndef pda_H
#define pda_H

#include <stdio.h>
#include <string.h>
#include "stackchar.h"
#include "boolean.h"

typedef char masukan[1005];
//tipe data inputan (string)

typedef struct {
    int Posisi;
    Stack S;
} State;
//Tipe data bentukan state pada PushDown Automata
//Terdiri dari nomor state dan stack
//Manipulasi stack dapat dilihat di stackt.h dan stackt.c

//Selektor
#define Posisi(Q) (Q).Posisi
#define Stack(Q) (Q).S

void transisi (State *Q, char input);
```

```
//Merubah kondisi state Q (nomor dan stack) jika menerima masukan input

boolean validasi (masukan s);
//Mengembalikan true jika input dari user valid dan false jika input dari user tidak
valid
#endif
```

Berikut adalah source code dari file pda.c, implementasi dari file header pda.h:

```
#include <stdio.h>
#include <string.h>
#include "stackchar.h"
#include "boolean.h"
#include "pda.h"

void transisi(State *Q, char input) {
    char C;
    if (Posisi(*Q) == 0) {
        if (input == '(') {
            if (InfoTop(Stack(*Q)) == 'Z') {
                Push(&Stack(*Q), 'X');
            }
            else if (InfoTop(Stack(*Q)) == 'X') {
                Push(&Stack(*Q), 'X');
            }
            else {
                Posisi(*Q) = 6;
            }
        }
        else if (input == '-') {
            if (InfoTop(Stack(*Q)) == 'Z') {
                Posisi(*Q) = 1; //Pindah ke state 1
            }
            else if (InfoTop(Stack(*Q)) == 'X') {
                Posisi(*Q) = 1; //Pindah ke state 1
            }
            else {
                Posisi(*Q) = 6;
```

```
            }
        }
        else if (input == '0' || input == '1' || input == '2' || input == '3' ||
        input == '4' || input == '5' || input == '6' || input == '7' || input == '8'
        || input == '9') {
            if (InfoTop(Stack(*Q)) == 'X') {
                Posisi(*Q) = 2;
            }
            else if (InfoTop(Stack(*Q)) == 'Z') {
                Posisi(*Q) = 2;
            }
            else {
                Posisi(*Q) = 6;
            }
        }
        else if (input == '.') {
            if (InfoTop(Stack(*Q)) == 'X') {
                Posisi(*Q) = 4;
            }
            else if (InfoTop(Stack(*Q)) == 'Z') {
                Posisi(*Q) = 4;
            }
            else {
                Posisi(*Q) = 6;
            }
        }
        else if (input == 'i') {
            if (InfoTop(Stack(*Q)) == 'Z') {
                Posisi(*Q) = 5;
            }
            else if (InfoTop(Stack(*Q)) == 'X') {
                Posisi(*Q) = 5;
            }
            else {
                Posisi(*Q) = 6;
            }
        }
        else {
```

```
            Posisi(*Q) = 6;
        }
    }
    else if (Posisi(*Q) == 1) {
        if (input == '0' || input == '1' || input == '2' || input == '3' || input ==
        '4' || input == '5' || input == '6' || input == '7' || input == '8' ||
        input == '9') {
            if (InfoTop(Stack(*Q)) == 'X') {
                Posisi(*Q) = 2;
            }
            else if (InfoTop(Stack(*Q)) == 'Z') {
                Posisi(*Q) = 2;
            }
            else {
                Posisi(*Q) = 6;
            }
        }
        else if (input == '(') {
            if (InfoTop(Stack(*Q)) == 'Z') {
                Posisi(*Q) = 0;
                Push(&Stack(*Q), 'X');
            }
            else if (InfoTop(Stack(*Q)) == 'X') {
                Posisi(*Q) = 0;
                Push(&Stack(*Q), 'X');
            }
            else {
                Posisi(*Q) = 6;
            }
        } else if (input == 'i') {
            if (InfoTop(Stack(*Q)) == 'X' || InfoTop(Stack(*Q)) == 'Z') {
                Posisi(*Q) = 5;
            } else {
                Posisi(*Q) = 6;
            }
        }
        else {
            Posisi(*Q) = 6;
```

```
        }
    }
    else if (Posisi(*Q) == 2) {
        if (input == '0' || input == '1' || input == '2' || input == '3' || input ==
        '4' || input == '5' || input == '6' || input == '7' || input == '8' ||
        input == '9') {
            if (InfoTop(Stack(*Q)) == 'X') {
                //do nothing
            }
            else if (InfoTop(Stack(*Q)) == 'Z') {
                //do nothing
            }
            else {
                Posisi(*Q) = 6;
            }
        }
        else if (input == '.') {
            if (InfoTop(Stack(*Q)) == 'X') {
                Posisi(*Q) = 4;
            }
            else if (InfoTop(Stack(*Q)) == 'Z') {
                Posisi(*Q) = 4;
            }
            else {
                Posisi(*Q) = 6;
            }
        }
        else if (input == ')') {
            if (InfoTop(Stack(*Q)) == 'X') {
                Pop(&Stack(*Q), &C);
            }
            else {
                Posisi(*Q) = 6;
            }
        }
        else if (input == '+' || input == '-' || input == '*' || input == '/' ||
        input == '^') {
            if (InfoTop(Stack(*Q)) == 'X') {
```

```
            Posisi(*Q) = 3;
        }
        else if (InfoTop(Stack(*Q)) == 'Z') {
            Posisi(*Q) = 3;
        }
        else {
            Posisi(*Q) = 6;
        }
    }
    else if (input == 'i') {
        if (InfoTop(Stack(*Q)) == 'X' || InfoTop(Stack(*Q)) == 'Z') {
            Posisi(*Q) = 5;
        }
        else {
            Posisi(*Q) = 6;
        }
    }
    else {
        Posisi(*Q) = 6;
    }
}
else if (Posisi(*Q) == 3) {
    if (input == '0' || input == '1' || input == '2' || input == '3' || input ==
    '4' || input == '5' || input == '6' || input == '7' || input == '8' || input
    == '9') {
        if (InfoTop(Stack(*Q)) == 'X') {
            Posisi(*Q) = 2;
        }
        else if (InfoTop(Stack(*Q)) == 'Z') {
            Posisi(*Q) = 2;
        }
        else {
            Posisi(*Q) = 6;
        }
    }
    else if (input == '(') {
        if (InfoTop(Stack(*Q)) == 'X') {
            Push(&Stack(*Q), 'X');
```

```
            Posisi(*Q) = 0;
        }
        else if (InfoTop(Stack(*Q)) == 'Z') {
            Push(&Stack(*Q), 'X');
            Posisi(*Q) = 0;
        }
        else {
            Posisi(*Q) = 6;
        }
    }
    else if (input == '-') {
        if (InfoTop(Stack(*Q)) == 'X') {
            Posisi(*Q) = 1;
        }
        else if (InfoTop(Stack(*Q)) == 'Z') {
            Posisi(*Q) = 1;
        }
        else {
            Posisi(*Q) = 6;
        }
    }
    else if (input == '.') {
        if (InfoTop(Stack(*Q)) == 'X') {
            Posisi(*Q) = 4;
        }
        else if (InfoTop(Stack(*Q)) == 'Z') {
            Posisi(*Q) = 4;
        }
        else {
            Posisi(*Q) = 6;
        }
    }
    else if (input == 'i') {
        if (InfoTop(Stack(*Q)) == 'X' || InfoTop(Stack(*Q)) == 'Z') {
            Posisi(*Q) = 5;
        }
        else {
            Posisi(*Q) = 6;
```

```
            }
        }
        else {
            Posisi(*Q) = 6;
        }
    }
    else if (Posisi(*Q) == 4) {
        if (input == '0' || input == '1' || input == '2' || input == '3' || input ==
        '4' || input == '5' || input == '6' || input == '7' || input == '8' || input
        == '9') {
            if (InfoTop(Stack(*Q)) == 'X') {
                //do nothing
            }
            else if (InfoTop(Stack(*Q)) == 'Z') {
                //do nothing
            }
            else {
                Posisi(*Q) = 6;
            }
        }
        else if (input == '+' || input == '-' || input == '*' || input == '/' ||
        input == '^') {
            if (InfoTop(Stack(*Q)) == 'X') {
                Posisi(*Q) = 3;
            }
            else if (InfoTop(Stack(*Q)) == 'Z') {
                Posisi(*Q) = 3;
            }
            else {
                Posisi(*Q) = 6;
            }
        }
        else if (input == ')') {
            if (InfoTop(Stack(*Q)) == 'X') {
                Pop(&Stack(*Q), &C);
            }
            else {
                Posisi(*Q) = 6;
```

```
            }
        }
        else if (input == 'i') {
            if (InfoTop(Stack(*Q)) == 'X' || InfoTop(Stack(*Q)) == 'Z') {
                Posisi(*Q) = 5;
            }
            else {
                Posisi(*Q) = 6;
            }
        }
        else {
            Posisi(*Q) = 6;
        }
    }
    else if (Posisi(*Q) == 5) {
        if (input == ')') {
            if (InfoTop(Stack(*Q)) == 'X') {
                Pop(&Stack(*Q), &C);
            }
            else {
                Posisi(*Q) = 6;
            }
        }
        else if (input == '+' || input == '-' || input == '*' || input == '/' ||
        input == '^') {
            if (InfoTop(Stack(*Q)) == 'X') {
                Posisi(*Q) = 3;
            }
            else if (InfoTop(Stack(*Q)) == 'Z') {
                Posisi(*Q) = 3;
            }
            else {
                Posisi(*Q) = 6;
            }
        }
        else {
            Posisi(*Q) = 6;
        }
```

```
    }
}
boolean validasi(masukan s)
{
    int i;
    State Q;
    char karakter;
    //Inisialisasi
    Posisi(Q) = 0;
    CreateEmpty(&Stack(Q));
    Push(&Stack(Q), 'Z');

    for (i = 0; i < strlen(s); i++) {
        karakter = s[i];
        transisi(&Q, karakter);
        //Uncomment next line to check state movement
        // printf("%d, %c\n", Posisi(Q), InfoTop(Stack(Q)));
    }

    if ((Posisi(Q) == 2 && InfoTop(Stack(Q)) == 'Z') || (Posisi(Q) == 4 &&
    InfoTop(Stack(Q)) == 'Z') || (Posisi(Q) == 5 && InfoTop(Stack(Q)) == 'Z')) {
        return true;
    }
    else {
        return false;
    }
}
```

Berikut adalah source code dari file header stack.h:

```
#ifndef Stacktt_H
#define Stacktt_H
#include <stdbool.h>
#include "mesintoken.h"

#define Nill 0
#define MaxEll 20000
/* Nill adalah Stackt dengan elemen kosong . */
```

```
/* Karena indeks dalam bhs C dimulai 0 maka tabel dg indeks 0 tidak dipakai */

typedef Token Infotype;
typedef int address;   /* indeks tabel */

/* Contoh deklarasi variabel bertype Stackt dengan ciri TOP : */
/* Versi I : dengan menyimpan tabel dan alamat top secara eksplisit*/
typedef struct {
       Infotype T[MaxEll+1]; /* tabel penyimpan elemen */
       address TOP;  /* alamat TOP: elemen puncak */
} Stackt;
/* Definisi Stackt S kosong : S.TOP = Nill */
/* Elemen yang dipakai menyimpan Nillai Stackt T[1]..T[MaxEll] */
/* Jika S adalah Stackt maka akses elemen : */
   /* S.T[(S.TOP)] untuk mengakses elemen TOP */
   /* S.TOP adalah alamat elemen TOP */

/* Definisi akses dengan Selektor : Set dan Get */
#define Top(S) (S).TOP
#define InfoTop(S) (S).T[(S).TOP]

/* ************ Prototype ************ */
/* *** Konstruktor/Kreator *** */
void CreateEmptyst (Stackt *S);
/* I.S. sembarang; */
/* F.S. Membuat sebuah Stackt S yang kosong berkapasitas MaxEll */
/* jadi indeksnya antara 1.. MaxEll+1 karena 0 tidak dipakai */
/* Ciri Stackt kosong : TOP berNillai Nill */

/* ************ Predikat Untuk test keadaan KOLEKSI ************ */
bool IsEmptyst (Stackt S);
/* Mengirim true jika Stackt kosong: lihat definisi di atas */
bool isFull (Stackt S);
/* Mengirim true jika tabel penampung Nillai elemen Stackt penuh */

/* ************ Menambahkan sebuah elemen ke Stackt ************ */
void Pushst (Stackt * S, Infotype X);
/* Menambahkan X sebagai elemen Stackt S. */
```

```
/* I.S. S mungkin kosong, tabel penampung elemen Stackt TIDAK penuh */
/* F.S. X menjadi TOP yang baru,TOP bertambah 1 */

/* ************ Menghapus sebuah elemen Stackt ************ */
void Popst (Stackt * S, Infotype* X);
/* Menghapus X dari Stackt S. */
/* I.S. S  tidak mungkin kosong */
/* F.S. X adalah Nillai elemen TOP yang lama, TOP berkurang 1 */

Stackt Reverse(Stackt S);

#endif
```

Berikut adalah source code dari file stack.c, implementasi dari file header stack.h:

```
#include <stdio.h>
#include "stack.h"

/* ************ Prototype ************ */
/* *** Konstruktor/Kreator *** */
void CreateEmptyst (Stackt *S)
/* I.S. sembarang; */
/* F.S. Membuat sebuah Stackt S yang kosong berkapasitas MaxEll */
/* jadi indeksnya antara 1.. MaxEll+1 karena 0 tidak dipakai */
/* Ciri Stackt kosong : TOP berNillai Nill */
{
	Top(*S) = Nill;
}

/* ************ Predikat Untuk test keadaan KOLEKSI ************ */
bool IsEmptyst (Stackt S)
/* Mengirim true jika Stackt kosong: lihat definisi di atas */
{
	return (Top(S) == Nill);
}

bool isFull (Stackt S)
/* Mengirim true jika tabel penampung Nillai elemen Stackt penuh */
```

```
{
	return (Top(S) == MaxEll);
}

/* ************ Menambahkan sebuah elemen ke Stackt ************ */
void Pushst (Stackt * S, Infotype X)
/* Menambahkan X sebagai elemen Stackt S. */
/* I.S. S mungkin kosong, tabel penampung elemen Stackt TIDAK penuh */
/* F.S. X menjadi TOP yang baru,TOP bertambah 1 */
{
	Top(*S)++;
	InfoTop(*S) = X;
}

/* ************ Menghapus sebuah elemen Stackt ************ */
void Popst (Stackt * S, Infotype* X)
/* Menghapus X dari Stackt S. */
/* I.S. S  tidak mungkin kosong */
/* F.S. X adalah Nillai elemen TOP yang lama, TOP berkurang 1 */
{
	*X = InfoTop(*S);
	Top(*S)--;
}

Stackt Reverse(Stackt S){
	Stackt res;
	CreateEmptyst(&res);
	while (!IsEmptyst(S)){
		Infotype top;
		Popst(&S, &top);
		Pushst(&res, top);
	}
	return res;
}
```

Berikut adalah source code dari file header stackchar.h:

```
#ifndef stackchar_H
```

```
#define stackchar_H

#include "boolean.h"

#define Nil 0
#define MaxEl 100
/* Nil adalah stack dengan elemen kosong . */
/* Karena indeks dalam bhs C dimulai 0 maka tabel dg indeks 0 tidak dipakai */

typedef char infotype;
typedef int address;   /* indeks tabel */

/* Contoh deklarasi variabel bertype stack dengan ciri TOP : */
/* Versi I : dengan menyimpan tabel dan alamat top secara eksplisit*/
typedef struct {
       infotype T[MaxEl+1]; /* tabel penyimpan elemen */
       address TOP;  /* alamat TOP: elemen puncak */
} Stack;
/* Definisi stack S kosong : S.TOP = Nil */
/* Elemen yang dipakai menyimpan nilai Stack T[1]..T[MaxEl] */
/* Jika S adalah Stack maka akses elemen : */
   /* S.T[(S.TOP)] untuk mengakses elemen TOP */
   /* S.TOP adalah alamat elemen TOP */

/* Definisi akses dengan Selektor : Set dan Get */
#define Top(S) (S).TOP
#define InfoTop(S) (S).T[(S).TOP]

/* ************ Prototype ************ */
/* *** Konstruktor/Kreator *** */
void CreateEmpty (Stack *S);
/* I.S. sembarang; */
/* F.S. Membuat sebuah stack S yang kosong berkapasitas MaxEl */
/* jadi indeksnya antara 1.. MaxEl+1 karena 0 tidak dipakai */
/* Ciri stack kosong : TOP bernilai Nil */

/* ************ Predikat Untuk test keadaan KOLEKSI ************ */
boolean IsEmpty (Stack S);
```

```
/* Mengirim true jika Stack kosong: lihat definisi di atas */
boolean IsFull (Stack S);
/* Mengirim true jika tabel penampung nilai elemen stack penuh */

/* ************ Menambahkan sebuah elemen ke Stack ************ */
void Push (Stack * S, infotype X);
/* Menambahkan X sebagai elemen Stack S. */
/* I.S. S mungkin kosong, tabel penampung elemen stack TIDAK penuh */
/* F.S. X menjadi TOP yang baru,TOP bertambah 1 */

/* ************ Menghapus sebuah elemen Stack ************ */
void Pop (Stack * S, infotype* X);
/* Menghapus X dari Stack S. */
/* I.S. S  tidak mungkin kosong */
/* F.S. X adalah nilai elemen TOP yang lama, TOP berkurang 1 */

#endif
```

Berikut adalah source code dari file stackchar.c, implementasi dari file header stackchar.h:

```
#include "stackchar.h"
#include "boolean.h"

/* ************ Prototype ************ */
/* *** Konstruktor/Kreator *** */
void CreateEmpty (Stack *S){
	Top(*S) = Nil;
}
/* I.S. sembarang; */
/* F.S. Membuat sebuah stack S yang kosong berkapasitas MaxEl */
/* jadi indeksnya antara 1.. MaxEl+1 karena 0 tidak dipakai */
/* Ciri stack kosong : TOP bernilai Nil */

/* ************ Predikat Untuk test keadaan KOLEKSI ************ */
boolean IsEmpty (Stack S){
	return(Top(S) == Nil);
}
/* Mengirim true jika Stack kosong: lihat definisi di atas */
```

```
boolean IsFull (Stack S){
	return(Top(S) == MaxEl);
}
/* Mengirim true jika tabel penampung nilai elemen stack penuh */

/* ************ Menambahkan sebuah elemen ke Stack ************ */
void Push (Stack * S, infotype X){
	Top(*S) += 1;
	InfoTop(*S) = X;
}
/* Menambahkan X sebagai elemen Stack S. */
/* I.S. S mungkin kosong, tabel penampung elemen stack TIDAK penuh */
/* F.S. X menjadi TOP yang baru,TOP bertambah 1 */

/* ************ Menghapus sebuah elemen Stack ************ */
void Pop (Stack * S, infotype* X){
	*X = InfoTop(*S);
	Top(*S) -= 1;
}
/* Menghapus X dari Stack S. */
/* I.S. S  tidak mungkin kosong */
/* F.S. X adalah nilai elemen TOP yang lama, TOP berkurang 1 */
```

Bab IV

CONTOH MASUKAN DAN KELUARAN

Berikut adalah cuplikan layar terminal yang menunjukan masukan dan keluaran.