**LAPORAN**

**Aplikasi CFG dan PDA pada Pengenalan Ekspresi Matematika**

Tugas II IF2220 Teori Bahasa Formal dan Automata

**JESSLYN NATHANIA 13517053**

**ABEL STANLEY 13517068**

**KEVIN ANGELO 13517086**

****

**PROGRAM STUDI TEKNIK INFORMATIKA**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**2018**

# **Deskripsi Masalah**

Kalkulator adalah alat untuk menghitung dari perhitungan sederhana seperti penjumlahan, pengurangan, perkalian, dan pembagian, sampai kepada kalkulator sains yang dapat menghitung rumus matematika tertentu. Pada perkembangannya saat ini, kalkulator sering dimasukkan sebagai fungsi tambahan dari komputer dan telepon genggam. Oleh karena itu, program ini dibuat sedemikian hingga menghasilkan sebuah kalkulator sederhana, yang menggunakan implementasi tata bahasa bebas konteks (CFG). Jika kalkulator diberikan sebuah ekspresi matematika, program dapat mengenali apakah ekspresi tersebut valid atau tidak (syntax error). Bila ekspresi tersebut valid, program akan menghitung nilai dari ekspresi tersebut dengan mengubah terlebih dahulu setiap simbol terminal (angka) menjadi nilai numerik yang bersesuaian. Program juga dapat mengenali apakah ekspresi tersebut memungkinkan dihitung atau tidak (math error).

Terminal terdiri dari simbol aritmatika biasa (+, -, \*, /), perpangkatan (^), tanda negatif (-), angka (0, 1, 2, 3, 4, 5, 6, 7, 8, 9), desimal (.), dan tanda kurung (()). Operator terdiri dari symbol aritmatika biasa, tidak mengandung huruf-huruf (e, pi, dan lain-lain). Tidak ada spasi antar token. Untuk implementasi fungsi pangkat, terdapat kemungkinan implementasi fungsi pangkat negatif dan pangkat pecahan (akar). Notasi 9^(0.5) dipergunakan untuk menuliskan notasi akar 2 dan notasi 2^(-1) untuk menuliskan notasi ½ pada command prompt.

Dalam implementasi perhitungan pada kalkulator, aturan sebagai berikut berlaku.

1. Operasi yang berada dalam kurung dikerjakan lebih dahulu.
2. Urutan eksekusi operasi secara umum.
3. Implementasi dikerjakan berurutan dari kiri, kecuali untuk pangkat, dikerjakan dari kanan (Contoh: 4^3^2 = 4^9, bukan 64^2).

Operan terdiri dari bilangan riil (baik positif atau negatif), tidak hanya bilangan bulat, dari digit (0,1, 2,3, 4, 5, 6, 7, 8, 9). Program merupakan implementasi dari tata bahasa CFGyang dibuat terlebih dahulu menggunakan teori yang telah dipelajari. Program dapat menangani masukan dan hasil ekspresi bilangan imajiner dan bilangan kompleks. Bilangan imajiner ditandai dengan huruf i.

# **Deskripsi Umum Solusi**

Dalam menyelesaikan permasalahan tersebut, program dibuat dengan metode parsing algorithm, yang disebut recursive descent. Fungsi dibuat terurut dari prioritas terendah yaitu tambah/kurang, kali/bagi, pangkat, minus, lalu pengambilan bilangan (termasuk kompleks atau bilangan real biasa). Selain itu, program juga dilengkapi dengan mesin token. Token merupakan struktur data yang terdiri dari *tkn* dan *val*. *Tkn* membedakan antara digit dan kompleks *i* dan operator (+,-,/,\*,- atau minus). *Val* yang menampung nilai; jika *tkn* adalah digit, *val* menampung nilai yang ada, jika *tkn* kompleks *i*, *val* menampung nilai 1, jika *tkn* ialah operator, maka *val* menampung nilai -1.

Jika diberikan sebuah ekspresi, program akan memulai pengecekan dari fungsi terurut terendah(prioritas terpenting). Apakah itu minus atau angka, jika ya, nilai setelah symbol minus atau nilai secara langsung akan disimpan di suatu variable bertipe *double complex* tertentu, jika bukan, akan memberikan error message “syntax error”. Lalu dilanjuti dengan pengecekan operator yang sesuai fungsi urutan terendah ke tinggi (prioritas terpenting ke menurun).

1. E -> (E)
2. E -> E+E | E-E
3. E -> E\*E | E/E
4. E -> -E
5. E -> I
6. E -> Ei
7. I -> 0|1|2|3|4|5|6|7|8|9

# **CFG yang Digunakan**

CFG (Context Free Grammar) adalah suatu kelompok peraturan rekursif yang digunakan untuk menghasilkan pola-pola *string*. Komponen-komponen CFG terdiri dari:

1. Simbol terminal atau yang dikenal dengan alfabet
2. Simbol non-terminal, yaitu simbol yang dapat diubah ke simbol terminal, dapat berupa hasil dari simbol non-terminal lainnya
3. Koleksi produksi (*set of production*), yang berisi peraturan-peraturan yang untuk menggantikan simbol non-terminal (biasanya terletak di sebelah kiri produksi) pada *string* dengan simbol non-terminal lain maupun simbol terminal (pada sebelah kanan produksi).
4. Simbol awal (*start symbol*), yaitu simbol non-terminal yang muncul pada *string* yang dihasilkan oleh suatu grammar.

Pada kasus ini, program kalkulator menggunakan CFG untuk ekspresi aritmatika, yang terdiri dari ekspresi dan operator. Secara umum, bentuk dari *production* yang digunakan adalah

<expression> 🡪 <expression> <operator> <expression>

Dapat juga mengandung tanda kurung, decimal, dan tanda negatif (minus).

<expression> 🡪 (<expression>)

<expression> 🡪 -<expression>

<expression> 🡪 <operan>.<operan>

Simbol terminal pada grammar ini yaitu {+,-,\*,/,(,),0,1,2,3,4,5,6,7,8,9,^,.}, sedangkan simbol non-terminal nya adalah <expression>, yang juga merupakan *start symbol*.

Dalam eksekusinya, digunakan algoritma *recursive descent*, yaitu metode parsing dimana fungsi-fungsi dieksekusi berdasarkan prioritas tertentu. Penjumlahan dan pengurangan menjadi prioritas terendah, diikuti oleh perkalian dan pembagian, fungsi pangkat, bilangan negatif, lalu bilangan itu sendiri.

CFG yang digunakan pada program dalam bentuk quadruple G = (N,T,P,S) dimana

* N adalah simbol-simbol non-terminal
* T adalah simbol-simbol terminal yang tidak terdapat pada N
* P adalah peraturan (*production*) yang memetakan N pada gabungan N dan T, atau dapat dituliskan sebagai P: N 🡪 (N U T)\*, dengan U adalah Union dan \* adalah *closure*
* S adalah simbol awal

adalah

G = ({<expression>}, {+,-,\*,/,(,),0,1,2,3,4,5,6,7,8,9,^}, P, <calculator>)

dengan P (*set of production*) berupa:

1. <calculator> 🡪 <expression>
2. <expression> 🡪 I
3. <expression> 🡪 I.I
4. <expression> 🡪 <expression>i
5. <expression> 🡪 -<expression>
6. <expression> 🡪 (<expression>)
7. <expression> 🡪 <expression>^<expression>
8. <expression> 🡪 <expression>\*<expression> | <expression>/<expression>
9. <expression> 🡪 <expression>+<expression> | <expression>-<expression>
10. I 🡪 II
11. I 🡪 0|1|2|3|4|5|6|7|8|9

# **Source Code**

**Calculator.c**

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <math.h>

#include "mesintoken.h"

#include "mesinkar.h"

#include <complex.h>

double complex result=0+0\*I;

double complex PlusMinusFunction(void);

double complex MultDivFunction(void);

double complex PowerFunction(void);

double complex NegativeFunction(void);

double complex NumParenthesesFunction(void);

double complex MakeCMPLX(double a, double b){

return (a + b \* I);

}

void error(){

if (isinf(result) || isnan(result))

printf("Math Error!\n");

else

printf("Syntax Error!\n");

exit(1);

}

void ADVREAD( char targetchar){

if (CToken.tkn==targetchar){

ADVTOKEN();

}else{

error();

}

}

int main(){

printf("CALCULATOR IS READY! INPUT EXPRESSION : ");

STARTTOKEN();

result = PlusMinusFunction();

if ((CToken.tkn =='\n') && !isinf(result) && !isnan(result)){

printf("The result is... \*Ba dum tss!\* = ");

if (round(cimag(result))==0){

printf("%.2f\n",creal(result));

}else if (round(creal(result))==0){

if (round(cimag(result))==1){

printf("i\n");

}else{

printf("%.2fi\n",cimag(result));

}

}else{

if (round(cimag(result))==1){

printf("%.2f+i\n",creal(result));

}else{

printf("%.2f%+.2fi\n",creal(result), cimag(result));

}

}

}

else{

error();

}

return 0;

}

double complex PlusMinusFunction(void){

double complex temp = MultDivFunction();

while (CToken.tkn=='+' || CToken.tkn =='-'){

if(CToken.tkn =='+'){

ADVREAD('+');

temp += MultDivFunction();

}else{

ADVREAD('-');

temp -= MultDivFunction();

}

}

return temp;

}

double complex MultDivFunction(void){

double complex temp = PowerFunction();

while (CToken.tkn =='\*' || CToken.tkn =='/'){

if(CToken.tkn =='\*'){

ADVREAD('\*');

temp \*= PowerFunction();

}

else{

ADVREAD('/');

temp \*= 1/PowerFunction();

}

}

return temp;

}

double complex PowerFunction(void){

double complex temp = NegativeFunction();

while (CToken.tkn =='^'){

ADVREAD('^');

temp = cpow(temp,PowerFunction());

}

return temp;

}

double complex NegativeFunction(void){

double complex temp;

if(CToken.tkn == '-'){

ADVREAD('-');

temp=(-1)\*NumParenthesesFunction();

if (creal(temp)==0){

temp = MakeCMPLX(0,cimag(temp));

}

if (cimag(temp)==0){

temp = MakeCMPLX(creal(temp),0);

}

return temp;

}

temp=NumParenthesesFunction();

return temp;

}

double complex NumParenthesesFunction(void){

double complex temp;

if (CToken.tkn =='('){

ADVREAD('(');

temp = PlusMinusFunction();

ADVREAD(')');

}

else if(CToken.tkn == 'i'){

temp = MakeCMPLX(0,CToken.val);

ADVREAD('i');

}

else if (CToken.tkn == 'b')

{

temp = MakeCMPLX(CToken.val,0);

ADVTOKEN();

}

else{

error();

}

if (CToken.tkn == 'i'){

ADVREAD('i');

return MakeCMPLX(0,temp);

}else{

return temp;

}

}

**mesintoken.h**

/\* File: mesintoken.h \*/

/\* Definisi Mesin Token: Model Akuisisi Versi I \*/

#ifndef \_\_MESINTOKEN\_H\_\_

#define \_\_MESINTOKEN\_H\_\_

#include "boolean.h"

#include "mesinkar.h"

#define NMax 50

#define BLANK ' '

typedef struct {

char tkn; /\* token yang dibaca \*/

double val; /\* nilai bilangan jika tkn = 'b' \*/

} Token;

/\* State Mesin Kata \*/

extern boolean EndToken;

extern Token CToken;

void IgnoreBlank();

/\* Mengabaikan satu atau beberapa BLANK

I.S. : CC sembarang

F.S. : CC ≠ BLANK atau CC = MARK \*/

void STARTTOKEN();

/\* I.S. : CC sembarang

F.S. : EndToken = true, dan CC = MARK;

atau EndToken = false, CToken adalah Token yang sudah diakuisisi,

CC karakter pertama sesudah karakter terakhir Token \*/

void ADVTOKEN();

/\* I.S. : CC adalah karakter pertama kata yang akan diakuisisi

F.S. : CToken adalah Token terakhir yang sudah diakuisisi,

CC adalah karakter pertama dari kata berikutnya, mungkin MARK

Jika CC = MARK, maka EndToken = true

Proses : Akuisisi kata menggunakan procedure SalinKata \*/

void SalinToken();

/\* Mengakuisisi Token dan menyimpan hasilnya dalam CToken

I.S. : CC adalah karakter pertama dari Token

F.S. : CToken berisi Token yang sudah diakuisisi;

CC = BLANK atau CC = MARK;

CC adalah karakter sesudah karakter terakhir yang diakuisisi.

Jika panjang kata melebihi NMax, maka sisa kata "dipotong" \*/

#endif

**mesintoken.c**

#include "mesintoken.h"

#include "mesinkar.h"

#include <stdio.h>

#include "boolean.h"

#include <stdlib.h>

#include <math.h>

boolean EndToken;

Token CToken;

int fraction;

boolean dotExist;

void IgnoreBlank(){

while ((CC== BLANK) && (CC != MARK)){

ADV();

}

}

void STARTTOKEN(){

IgnoreBlank();

START();

IgnoreBlank();

if (CC == MARK)

EndToken = true;

else{

EndToken = false;

SalinToken();

}

}

void ADVTOKEN(){

ADV();

IgnoreBlank();

SalinToken();

IgnoreBlank();

}

void SalinToken(){

CToken.val = 0;

fraction = 0;

dotExist = false;

if (CC == '+'){

CToken.tkn = '+';

CToken.val = -1;

}

else if (CC == '-'){

CToken.tkn = '-';

CToken.val = -1;

}

else if (CC == '/'){

CToken.tkn = '/';

CToken.val = -1;

}

else if (CC == '\*'){

CToken.tkn = '\*';

CToken.val = -1;

}

else if (CC == '^'){

CToken.tkn = '^';

CToken.val = -1;

}

else if (CC == '('){

CToken.tkn = '(';

CToken.val = -1;

}

else if (CC == ')'){

CToken.tkn = ')';

CToken.val = -1;

}

else if (CC == '\n'){

CToken.tkn = '\n';

CToken.val = -1;

}

else if (CC=='i'){

CToken.tkn='i';

CToken.val=1;

}

else if (CC >= '0' && CC <= '9' ){

CToken.tkn = 'b';

for(;;){

if (CC == '.'){

if (dotExist){

printf("SYNTAX ERROR!\n");

exit(1);

}

else{

dotExist = true;

}

IgnoreBlank();

ADV();

IgnoreBlank();

}

else{

if (!dotExist){

CToken.val = (CToken.val \*10) + (CC-'0');

}else{

fraction ++;

CToken.val = CToken.val + ((CC-'0')/(pow(10.0,fraction)));

}

IgnoreBlank();

ADV();

IgnoreBlank();

if ( CC== MARK || CC == BLANK || !((CC >= '0' && CC <= '9') || CC == '.')) {

BACKCHAR();

break;

}

}

}

}else{

CToken.tkn='e';

CToken.val=-1;

}

}

**mesinkar.h**

/\* File: mesinkar.h \*/

/\* Definisi Mesin Karakter \*/

#ifndef \_\_MESIN\_KAR\_H\_

#define \_\_MESIN\_KAR\_H\_

#include "boolean.h"

#define MARK '\n'

/\* State Mesin \*/

extern char CC;

extern boolean EOP;

void BACKCHAR();

void START();

/\* Mesin siap dioperasikan. Pita disiapkan untuk dibaca.

Karakter pertama yang ada pada pita posisinya adalah pada jendela.

I.S. : sembarang

F.S. : CC adalah karakter pertama pada pita

Jika CC != MARK maka EOP akan padam (false)

Jika CC = MARK maka EOP akan menyala (true) \*/

void ADV();

/\* Pita dimajukan satu karakter.

I.S. : Karakter pada jendela = CC, CC != MARK

F.S. : CC adalah karakter berikutnya dari CC yang lama,

CC mungkin = MARK

Jika CC = MARK maka EOP akan menyala (true) \*/

#endif

**mesinkar.c**

/\* File: mesinkar.c \*/

/\* Implementasi Mesin Karakter \*/

#include "mesinkar.h"

#include <stdio.h>

char CC;

boolean EOP;

static FILE \* pita;

static char\* ekspresi;

static int retval;

void START() {

CC = getchar();

}

void BACKCHAR(){

ungetc(CC,stdin);

}

void ADV() {

CC = getchar();

}

# **Tampilan Layar**











