

# **APLIKASI CFG DAN PDA PADA PENGENALAN EKSPRESI MATEMATIKA**

## **LAPORAN TUGAS BESAR II**

**Mata Kuliah Teori Bahasa Formal dan Otomata Semester I Tahun Akademik  
2018/2019**

Oleh :

**Ignatius Timothy Manullang 13517044**

**Juro Sutantra 13517113**



**PROGRAM STUDI TEKNIK INFORMATIKA  
INSTITUT TEKNOLOGI BANDUNG  
BANDUNG  
2018**

# **BAB I**

## **PERMASALAHAN**

### **1.1. Penjelasan 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.

Pada tugas kedua TBFO ini, anda diminta untuk membuat sebuah kalkulator sederhana, yang menggunakan implementasi tata bahasa bebas konteks (CFG) dan/atau pushdown automata (PDA). Bila kalkulator diberikan sebuah ekspresi matematika, program harus bisa mengenali apakah ekspresi tersebut valid atau tidak ( syntax error ). Bila ekspresi tersebut sudah valid, program akan menghitung nilai dari ekspresi tersebut dengan mengubah terlebih dahulu setiap simbol terminal (angka) menjadi nilai numerik yang bersesuaian. Program juga harus dapat mengenali apakah ekspresi tersebut mungkin dihitung atau tidak ( math error ).

Contoh ekspresi matematika yang valid adalah  $(-457.01+1280) * (35.7-11.0233)/(-6.1450)$  (setelah di- enter akan menampilkan hasil perhitungan ekspresi tersebut yaitu -3304.91) . Contoh ekspresi tidak valid adalah  $3*+-12/(57)$  (setelah di- enter akan ditampilkan pesan “S YNTAX ERROR” ), atau  $(-5)^{(2/3)}$  (setelah di-enter akan ditampilkan pesan (“MATH ERROR”))

### **1.2 Batasan Masalah**

Terminal hanya 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 implementasi fungsi pangkat, perhatikan bahwa terdapat kemungkinan implementasi fungsi pangkat negatif dan pangkat pecahan (akar). Silahkan gunakan notasi  $9^{(0.5)}$  untuk menuliskan notasi akar 2 dan notasi  $2^{(-1)}$  untuk menuliskan notasi  $\frac{1}{2}$  pada command prompt.

Dalam implementasi perhitungan pada kalkulator, gunakan aturan sebagai berikut.

1. Operasi yang berada dalam kurung dikerjakan lebih dahulu.
2. Perhatikan urutan eksekusi operasi .
3. Kerjakan berurutan dari kiri (Contoh:  $23 + 12 - 16 = 35 - 16 = 19$  atau  $8 * 5 : 2 = 40 : 2 = 20$  atau  $72 : 2 - 11 = 36 - 11 = 25$ ), kecuali untuk pangkat 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 dan PDA yang dibuat terlebih dahulu menggunakan teori yang telah dipelajari.

## BAB II

### GAMBARAN UMUM

#### 2.1 Gambaran Umum Solusi

Secara umum, program akan menerima sebuah input string, yang hanya berisi simbol-simbol seperti dalam batasan masalah, setelah itu masukan tersebut akan dibaca sebagai string, lalu dilakukan validasi dengan sebuah pemodelan PDA. Apabila input tidak valid akan mengeluarkan “SYNTAX ERROR”. Jika input telah valid maka akan dilakukan kalkulasi. Dalam proses kalkulasi string akan dibaca satu persatu dengan algoritma *recursive descent* yang sudah dimodifikasi sedikit.

#### 2.2 Gambaran Umum Validasi

Proses validasi dilaksanakan berdasarkan PDA yang telah disiapkan. Secara garis besar PDA hanya menerima string yang memiliki pola angka dilanjutkan dengan operan, sehingga seharusnya tidak dibutuhkan stack. Tetapi masalah muncul dalam penerimaan tanda kurung. Oleh sebab itu, stack digunakan untuk mengetahui bahwa setiap tanda kurung buka punya kurung tutup. Sehingga skema umum PDA menjadi angka dilanjutkan operan atau tanda kurung. Jika bertemu tanda kurung buka, maka push tanda tersebut ke stack dan harus kembali ke state awal. Jika bertemu tanda kurung tutup, pop tanda kurung buka dari stack dan boleh berulang ke state awal, operan. String diterima jika di dalam stack hanya terdapat simbol awal, dan diakhiri oleh angka atau kurung tutup.

Namun di dalam program kami, stack dalam PDA diubah menjadi sebuah variabel yang bertipe integer saja, karena pada hakikatnya kami tidak perlu menyimpan string “(”, karena pada saat kalkulasi akan di parse ulang. Sehingga agar program lebih efisien kami hanya menggunakan sebuah variabel integer dan menambahnya pada saat stack di push, dan mengurangnya saat stack di pop. PDA secara lebih detail akan dibahas dalam Bab III.

### 2.3 Gambaran Umum *Parsing* untuk kalkulasi

Untuk perhitungan digunakan algoritma *recursive descent* dimana, program akan membaca input satu persatu, dengan analisis kondisi yang sesuai. Namun, ada sedikit modifikasi yang diberikan, karena pita input kita telah divalidasi menggunakan PDA, sehingga input untuk kalkulasi dapat diasumsikan selalu benar. Oleh sebab itu, dalam algoritma *recursive descent* tidak dibutuhkan lagi output “SYNTAX ERROR”. Sehingga faktor kelemahan *recursive descent* dalam pesan kesalahan tereliminasi.

Secara umum, kami akan menginisialisasi program kami dengan angka yang didapat dari fungsi parser angka. Setelah itu program kami akan memanggil fungsi parser lain ketika bertemu suatu operator. Dimana fungsi parser ini meminta parameter basis yang merupakan hasil angka dari ruas kirinya. Fungsi parser terbagi menjadi 6 kali, tambah, bagi, kurang, kurung, dan eksponen. Di dalamnya kemungkinan terjadi pemanggilan fungsi parser lain atau dirinya sendiri sesuai dengan tingkat operasi. Seperti kurung, memiliki tingkat operasi tertinggi, sehingga dia harus menghitung hasil yang ada di dalam kurung baru di kembalikan ke awal. Jika terdapat operasi kali, tambah, dan atau bagi, maka fungsi kurung akan memanggil fungsi parser lain. Setiap fungsi parser akan memanggil parser angka, untuk menentukan nilai angka di sebelah kanannya.

Namun implementasi program kami masih menggunakan beberapa skema iteratif, seperti pada count fungsi pemicu parser dan fungsi kurung. Mungkin dengan melihat *source code* kami, bisa membuat pemahaman di atas menjadi lebih jelas.

## BAB III

### CFG DAN PDA

#### 3.1. Context Free Grammar

$G = (V, T, P, E)$

Dimana  $V = \{\text{kurung, Tamkur, VTamkur, kalbag, Vkalbag, Eksp, B, R, I, Decimal, Int, v, Complex}\}$

Dan  $T = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, *, ./, ^, (, ), i\}$

Dengan P adalah produksi di bawah ini :

$E \rightarrow \text{Ikurung}$

$E \rightarrow \text{Tamkur}$

$\text{kurung} \rightarrow (E)\text{kurung}$

$\text{kurung} \rightarrow (E)$

$\text{kurung} \rightarrow I$

$\text{Tamkur} \rightarrow \text{VTamkur} + \text{Tamkur}$

$\text{Tamkur} \rightarrow \text{VTamkur} - \text{Tamkur}$

$\text{Tamkur} \rightarrow \text{VTamkur}$

$\text{VTamkur} \rightarrow \text{kalbag}$

$\text{VTamkur} \rightarrow \text{kurung}$

$\text{VTamkur} \rightarrow \text{Eksp}$

$\text{VTamkur} \rightarrow I$

$\text{kalbag} \rightarrow \text{Vkalbag} * \text{kalbag}$

$\text{kalbag} \rightarrow \text{Vkalbag} / \text{kalbag}$

$\text{Vkalbag} \rightarrow \text{kurung}$

$\text{Vkalbag} \rightarrow \text{Eksp}$

$\text{Vkalbag} \rightarrow I$

$\text{Eksp} \rightarrow B^*R$

$B \rightarrow \text{kurung}$

$B \rightarrow I$

$R \rightarrow \text{Eksp}$

$R \rightarrow \text{kurung}$

$R \rightarrow I$

$I \rightarrow \text{Decimal}$

$I \rightarrow \text{Int}$

$I \rightarrow \text{Complex}$

$\text{Decimal} \rightarrow \text{Int.Int}$

$\text{Int} \rightarrow 0v$

$\text{Int} \rightarrow 1v$

$\text{Int} \rightarrow 2v$

$\text{Int} \rightarrow 3v$

$\text{Int} \rightarrow 4v$

$\text{Int} \rightarrow 5v$

$\text{Int} \rightarrow 6v$

$\text{Int} \rightarrow 7v$

$\text{Int} \rightarrow 8v$

$\text{Int} \rightarrow 9v$

$v \rightarrow \epsilon$

$v \rightarrow \text{Int}$

$\text{Complex} \rightarrow vi$

### 3.2. Pushdown Automata

$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, q_{12})$

$\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, *, /, ^, (, ), i\}$

$$\Gamma = \{ \langle, Z0 \}$$

Transition Table :

	$\rightarrow q_0$	$q_1$	$q_2$	$q_3$	$q_4$	$q_5$	$q_6$	$q_7$	$q_8$	$q_9$	$q_{10}$	$q_{11}$	$*q_{12}$
0..9, Z0	$q_1, Z0$	$q_1, Z0$	-	-	$q_4 / q_5, Z0$	-	-	-	-	-	-	-	-
(, Z0	$q_3, (Z0$	$q_3, (Z0$	-	-	-	$q_3, (Z0$	-	-	-	-	-	-	-
) , Z0	-	-	-	-	-	-	-	-	-	-	-	-	-
+, Z0	-	$q_9, Z0$	$q_9, Z0$	-	-	$q_9, Z0$	$q_9, Z0$	-	-	-	-	-	-
-, Z0	-	$q_7, Z0$	$q_7, Z0$	-	-	$q_7, Z0$	$q_7, Z0$	-	-	-	-	-	-
*, Z0	-	$q_8, Z0$	$q_8, Z0$	-	-	$q_8, Z0$	$q_8, Z0$	-	-	-	-	-	-
/, Z0	-	$q_{11}, Z0$	$q_{11}, Z0$	-	-	$q_{11}, Z0$	$q_{11}, Z0$	-	-	-	-	-	-
^, Z0	-	$q_{10}, Z0$	$q_{10}, Z0$	-	-	$q_{10}, Z0$	$q_{10}, Z0$	-	-	-	-	-	-
., Z0	-	$q_4, Z0$	-	-	-	-	-	-	-	-	-	-	-
i, Z0	$q_2, Z0$	$q_2, Z0$	-	-	-	$q_2, Z0$	-	-	-	-	-	-	-
0..9,(	$q_1, ($	$q_1, ($	-	-	-	-	-	-	-	-	-	-	-
(,(	$q_3, (($	$q_3, (($	$q_3, (($	-	-	$q_3, (($	-	-	-	-	-	-	-
) ,(	-	$q_6, \epsilon$	$q_6, \epsilon$	-	-	$q_6, \epsilon$	$q_6, \epsilon$	-	-	-	-	-	-
+, (	-	$q_9, ($	$q_9, ($	-	-	$q_9, ($	$q_9, ($	-	-	-	-	-	-
-, (	-	$q_7, ($	$q_7, ($	-	-	$q_7, ($	$q_7, ($	-	-	-	-	-	-
*, (	-	$q_8, ($	$q_8, ($	-	-	$q_8, ($	$q_8, ($	-	-	-	-	-	-
/, (	-	$q_{11}, ($	$q_{11}, ($	-	-	$q_{11}, ($	$q_{11}, ($	-	-	-	-	-	-
^, (	-	$q_{10}, ($	$q_{10}, ($	-	-	$q_{10}, ($	$q_{10}, ($	-	-	-	-	-	-
., (	-	$q_4, ($	-	-	-	-	-	-	-	-	-	-	-
i, (	$q_2, ($	$q_2, ($	-	-	-	$q_2, ($	-	-	-	-	-	-	-
$\epsilon, Z0$	-	$*q_{12}, Z0$	$*q_{12}, Z0$	-	-	$*q_{12}, Z0$	$q_0 / *q_{12}, Z0$	$q_0, Z0$	$q_0, Z0$	$q_0, Z0$	$q_0, Z0$	$q_0, Z0$	-
$\epsilon, ($	-	-	-	$q_0, ($	-	-	$q_0, ($	$q_0, ($	$q_0, ($	$q_0, ($	$q_0, ($	$q_0, ($	-

$Q = \{$

$q_0$  = karakter pada pita bebas

$q_2$  = karakter pada pita i

$q_1$  = karakter pada pita 0-9

$q_3$  = karakter pada pita (



q4 = karakter pada pita .

q8 = karakter pada pita +

q5 = karakter pada pita 0-9 tetapi pita telah melewati .

q9 = karakter pada pita x

q10 = karakter pada pita ^

q6 = karakter pada pita )

q11 = karakter pada pita /

q7 = karakter pada pita -

q12 = karakter pada pita }

Catatan : PDA hanya digunakan untuk validasi, sehingga stack hanya diisi kurung, untuk mengetahui jumlah kurung buka dan tutup seimbang. Pada proses ini pita input asli dicopy untuk dibaca oleh PDA. Sedangkan untuk kalkulasi menggunakan pita input yang asli.

## BAB IV

### IMPLEMENTASI PROGRAM

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <complex.h>

/*VARIABEL GLOBAL*/
char * x;
/*DEKLARASI FUNGSI DAN PROSEDUR*/
char *inputString(FILE* fp, size_t size);
int IsValidSyntax();
int KarToInteger(char c);
double complex count();
double complex salinDouble();
double complex tambah(double complex basis);
double complex kali(double complex basis);
double complex kurung();
double complex kurang(double complex basis);
double complex exponen(double complex basis);
double complex bagi(double complex basis);
int main()
/*Program utama secara garis besar dia akan menerima sebuah input dan
melakukan validasi apakah syntax benar.
jika benar maka mengeluarkan hasil perhitungan, bisa terjadi
Mengeluarkan "MATH ERROR" jika hasilnya infinity atau NAN.
jika salah maka mengeluarkan "SYNTAX ERROR" ke layar*/
// Input example:
/* 123(1)+(1+2)+3*6+1/(((1+3)/4)+1) = 144.5
 * 5/0 = MATH ERROR */
{
    do{
        x = inputString(stdin, 10);
        if (*x == '\\0')
        {
            printf("0");
        }
        else
        {
            if (IsValidSyntax()){
                double complex out=count();
                if (isfinite(creal(out)) && isfinite(cimag(out))){
                    if (cimag(out) <(-0) ){
                        if (creal(out)==0)
                            printf("-%gi\\n", -cimag(out));
                        else
```

```

        printf("%g - %gi \n",creal(out),-
        cimag(out));
    }
    else if (cimag(out) ==0 || cimag(out)==-0)
        printf("%g\n",creal(out));
    else{
        if (creal(out)==0)
            printf("%gi\n",cimag(out));
        else
            printf("%g + %gi
\n",creal(out),cimag(out));
    }
}
else
    printf("MATH ERROR\n");
}
else{
    printf("SYNTAX ERROR\n");
}
}
}while (*x!='o' && *x!='e' && *x!='k');
return 0;
}
/*REALISASI PROSEDUR*/
char *inputString(FILE* fp, size_t size){
//pembacaan input string yang diletakkan pada sebuah array character
dengan alokasi dinamik, sehingga tidak ada batas panjang string.
    char *str;
    int ch;
    size_t len = 0;
    str = realloc(NULL, sizeof(char)*size);//size is start size
    if(!str)return str;
    while(EOF!=(ch=fgetc(fp)) && ch != '\n'){
        str[len++]=ch;
        if(len==size){
            str = realloc(str, sizeof(char)*(size+=16));
            if(!str)return str;
        }
    }
    str[len++]='\0';

    return realloc(str, sizeof(char)*len);
}

int IsValidSyntax(){
/*Mengecek apakah inputan sudah valid, dengan metode PDA namun stacknya
diganti hanya dengan sebuah variabel, karena isi stack pasti tidak lebih
dari 1 buah karakter.*/
    char* P = x;
    int jmlkurung=0, valid=1;

```

```

char temp;
if (*P == '\0') {
    return 1 ;
}
else if (*(P+1)=='\0'){
    valid = *P=='-' || (*P>='0' && *P<='9') || (*P=='i');
    return valid;
}
else{
    temp = *P;
    P++;
    if (temp!='.'){
        while (valid && *P!='\0') {
            if (temp=='(' && (*P=='/' || *P=='+' || *P=='*' ||
                *P=='^' || *P==')' ))
                valid = 0;
            else if (temp=='(' && (*P=='-' || *P=='(' || (*P>='0' &&
                *P<='9') || *P=='i'))
                jmlkurung++;
            else if ((temp=='/' || temp == '+' || temp == '*' ||
                temp == '^' || temp == '-') && (*P=='/' || *P=='+' || *P=='*' || *P=='^' ||
                *P==')' || *P=='-' || *P=='.'))
                valid =0;
            else if (temp==')')
                jmlkurung--;
            else if (temp=='.' && !(*P>='0' && *P<='9'))
                valid =0;
            else if (temp=='i' && ((*P>='0' &&
                *P<='9') || (*P=='.')))
                valid = 0;
            temp = *P;
            P++;
        }
        if (temp==')')
            jmlkurung--;
        else if (temp == '(')
            jmlkurung++;
        else if (temp == '*' || temp=='/' || temp=='+' || temp == '-'
            || temp == '^')
            valid =0;
        return valid && jmlkurung==0;
    }
    else
        return 0;
}
}

int KarToInteger(char c){
/*PREKONDISI inputan harus antara 0-9.
fungsi untuk mengubah karakter menjadi integer*/

```

```

switch (c) {
    case '0' :
        return 0;
    case '1' :
        return 1;
    case '2' :
        return 2;
    case '3' :
        return 3;
    case '4' :
        return 4;
    case '5' :
        return 5;
    case '6' :
        return 6;
    case '7' :
        return 7;
    case '8' :
        return 8;
    case '9' :
        return 9;
    default :
    {
        printf("SYNTAX ERROR\n");
        exit(0);
    }
}

double complex salinDouble(){
/*PREKONDISI character yang sedang ditunjuk variabel global x adalah
angka, huruf 'i', atau tanda negatif.
fungsi yang berguna untuk membaca sebuah inputan bilangan real atau
imajiner*/
    char temp;
    double CInt=0;
    double complex retval;
    int neg=0;
    if (*x=='-'){
        neg=1;
        x++;
    }
    while (*x>='0'&&*x<='9'){
        temp=*x;
        x++;
        if (*x>='0'&&*x<='9')
            CInt= (CInt + KarToInteger(temp))*10;
        else
            CInt= (CInt + KarToInteger(temp));
    }
}

```

```

    if (*x=='.'){
        x++;
        int i=1;
        while (*x>='0'&&*x<='9'){
            CInt= (CInt + KarToInteger(*x)*pow(10,-i));
            i++;
            x++;
        }
    }
    else if (*x=='i'){
        if (CInt==0){
            if (*(x-1)=='0')
                retval = 0+0*I;
            else
                retval = 0+I;
        }
        else
            retval = 0+CInt*I;
        x++;
        return retval;
    }
    if (!neg) {
        retval = CInt + 0*I;
        return retval;
    }
    else {
        retval = -CInt + 0*I;
        return retval;
    }
}

double complex kurung(){
    /*PREKONDISI character pada indeks ke x adalah '('
    menghasilkan nilai dari awal kurung sampai kurung berakhir.*/
    x++;
    double complex tmp=0,temp=0;
    if (*x=='('){
        tmp = kurung();
    }
    if ((*x>='0' && *x<='9') || *x=='-' || *x=='i')
        temp = salinDouble();
    else
        temp = tmp;
    while (*x!=''){
        if (*x=='*')
            temp = kali(temp);
        else if (*x=='+')
            temp = tambah(temp);
        else if (*x=='-')
            temp = kurang(temp);
    }
}

```

```

        else if (*x=='/')
            temp = bagi(temp);
        else if (*x=='^')
            temp = exponen(temp);
        else if (*x=='(')
            temp = temp*kurung();
    }
    x++;

    if (!(*x>='0' && *x<='9'))
        return temp;
    else {
        tmp=salinDouble();
        return (temp*tmp);
    }
}

double complex exponen(double complex basis){
/*PREKONDISI basis terdefinisi sebagai angka sebelum operator '^' dan
isi indeks x sedang berada pada character '^'
fungsi ini akan mengembalikan nilai exponen dari basis dipangkatkan
sesuatu angka dikanannya, bila angka dikanan ternyata bertemu eksponen
maka proses akan direkursifkan, sampai bertemu operator yang bukan
eksponen*/
    x++;
    double complex tmp;
    if (*x=='('){
        tmp=kurung();
    }
    else
        tmp = salinDouble();
    if (*x!='^') {
        return cpow(basis,tmp);
    }
    else{
        return cpow(basis,exponen(tmp));
    }
}

double complex tambah(double complex basis){
/*PREKONDISI basis terdefinisi sebagai angka sebelum operator '+' dan
isi indeks x sedang berada pada character '+'
fungsi ini akan mengembalikan penambahan antara basis dengan angka
dibelakang operator +. bisa jadi ada perkalian dan/atau pembagian
dan/atau eksponen sehingga harus dikerjakan terlebih dahulu
*/
    x++;
    double complex tmp;
    if (*x=='('){
        tmp = kurung();
    }

```

```

else
    tmp = salinDouble();
if (*x!='+') {
    while (*x!='\0' && *x!='-' && *x!='+' && *x!='') {
        if (*x=='^')
            tmp = exponen(tmp);
        else if (*x=='*')
            tmp = kali(tmp);
        else if (*x=='/')
            tmp = bagi(tmp);
    }
    return basis+tmp;
}
else{
    return basis+tambah(tmp);
}
}

double complex kali(double complex basis){
/*PREKONDISI basis terdefinisi dan merupakan angka sebelum operator -
dan isi indeks x adalah karakter '*'
fungsi ini mengembalikan hasil perkalian antara basis dengan angka
disebelah kanan operan. bisa saja bertemu dengan eksponen, maka harus
dikerjakan terlebih dahulu sampai eksponen selesai*/
    x++;
    double complex tmp;
    if (*x=='('){
        tmp=kurung();
    }
    else
        tmp = salinDouble();
    if (*x!='*') {
        if (*x=='^')
            return basis*exponen(tmp);
        else
            return basis*tmp;
    }
    else{
        return basis*kali(tmp);
    }
}

double complex bagi(double complex basis){
/*PREKONDISI basis terdefinisi dan merupakan angka sebelum operator -
dan isi indeks x adalah karakter '/'
fungsi ini mengembalikan hasil pembagian antara basis dengan angka
disebelah kanan operan. bisa saja bertemu dengan eksponen, maka harus
dikerjakan terlebih dahulu sampai eksponen selesai*/

```



```

x++;
double complex tmp;
if (*x=='('){
    tmp = kurung();
}
else
    tmp = salinDouble();

if (*x!='/' ) {
    if (*x=='^')
        return basis/exponen(tmp);
    else
        return basis/tmp;
}
else{
    return basis/bagi(tmp);
}
}

double complex kurang(double complex basis){
/*PREKONDISI basis terdefinisi dan merupakan angka sebelum operator -
dan isi indeks x adalah karakter '-'
fungsi ini akan mengembalikan pengurangan antara basis dengan angka
dibelakang operator -. bisa jadi ada perkalian dan/atau pembagian
dan/atau eksponen sehingga harus dikerjakan terlebih dahulu*/
x++;
double complex tmp;
if (*x=='('){
    tmp=kurung();
}
else
    tmp = salinDouble();
if (*x!='-') {
    while (*x!='\0'&& *x!='-'&& *x!='+' && *x!='') {
        if (*x=='^')
            tmp =exponen(tmp);
        else if (*x=='*')
            tmp = kali(tmp);
        else if (*x=='/')
            tmp = bagi(tmp);
    }
    return basis-tmp;
}
else{
    return basis-kurang(tmp);
}
}
}

```

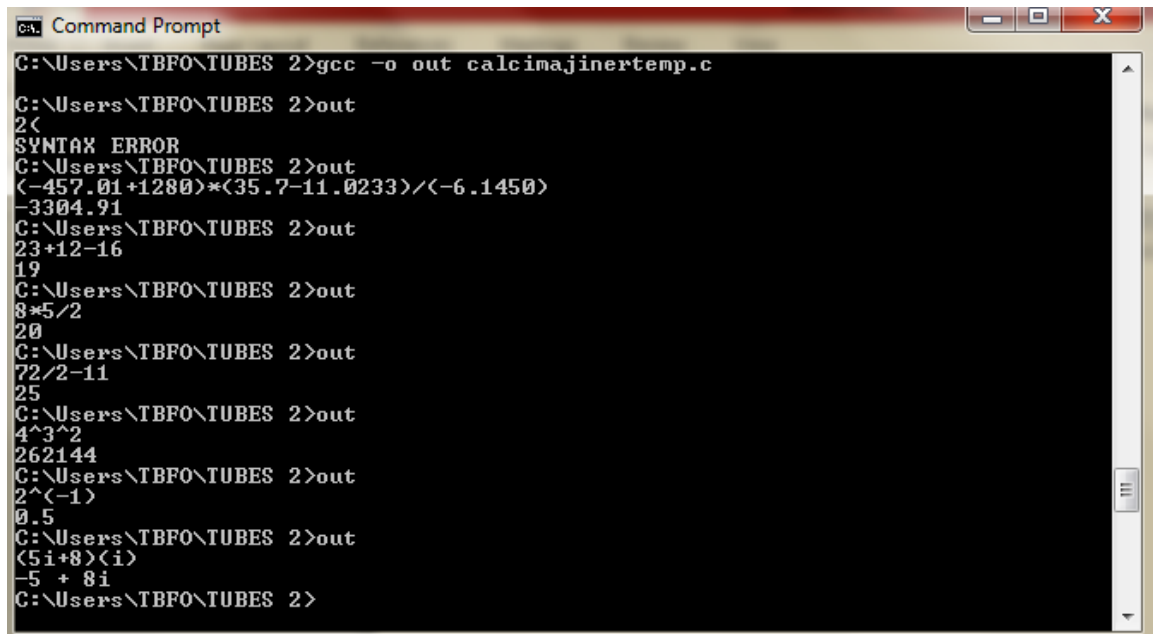
```

double complex count(){
/*PREKONDISI isi x tidak mungkin string yang tidak valid.
melakukan perhitungan hingga array habis, atau bertemu akhir dari
string. dan mengembalikan hasil perhitungan tersebut.*/
    double complex tmp;
    if (*x!='(')
        tmp=salinDouble();
    else
        tmp=kurung();
    while (*x!='\0'){
        if (*x=='*')
            tmp = kali(tmp);
        else if (*x=='+')
            tmp = tambah(tmp);
        else if (*x=='-')
            tmp = kurang(tmp);
        else if (*x=='/')
            tmp = bagi(tmp);
        else if (*x=='^')
            tmp = exponen(tmp);
        else if (*x=='(')
            tmp = kurung() * tmp;
    }
    return tmp;
}

```

## BAB V

### EKSPERIMEN



```
C:\Users\TBFO\TUBES 2>gcc -o out calcimajinertemp.c
C:\Users\TBFO\TUBES 2>out
2<
SYNTAX ERROR
C:\Users\TBFO\TUBES 2>out

$$\frac{(-457.01 + 1280) * (35.7 - 11.0233)}{-6.1450}$$

-3304.91
C:\Users\TBFO\TUBES 2>out

$$23 + 12 - 16$$

19
C:\Users\TBFO\TUBES 2>out

$$8 * 5 / 2$$

20
C:\Users\TBFO\TUBES 2>out

$$72 / 2 - 11$$

25
C:\Users\TBFO\TUBES 2>out

$$4^3^2$$

262144
C:\Users\TBFO\TUBES 2>out

$$2^{(-1)}$$

0.5
C:\Users\TBFO\TUBES 2>out

$$(5i + 8)(i)$$

-5 + 8i
C:\Users\TBFO\TUBES 2>
```