TUGAS BESAR 2

IF2220 – TEORI BAHASA FORMAL DAN AUTOMATA

Aplikasi CFG dan PDA pada Pengenalan Ekspresi Matematika

Disusun oleh:

Lydia Astrella Wiguna (13517019)

Karina Iswara (13517031)

Saskia Imani (13517142)

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK INFORMATIKA DAN ELEKTRO

INSTITUT TEKNOLOGI BANDUNG

Deskripsi Umum

a. Permasalahan

Kalkulator atau mesin hitung 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 sekarang ini, kalkulator sering dimasukkan sebagai fungsi tambahan daripada komputer, ponsel, bahkan sampai jam tangan.

Pada tugas ini, kelompok diminta untuk membuat sebuah kalkulator sederhana yang menggunakan implementasi *context-free grammar* (CFG) atau *pushdown automata* (PDA). Bila kalkulator diberikan sebuah ekspresi matematika, program harus bisa menentukan apakah ekspresi tersebut valid atau tidak (*syntax error*). Bila ekspresi 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 menentukan apakah ekspresi mungkin dihitung atau tidak (*math error*).

Terminal terdiri dari simbol aritmetika biasa (+, -, *, /), simbol pangkat (^), tanda negatif (-), angka (0, 1, 2, 3, 4, 5, 6, 7, 8, 9), desimal (.), dan tanda kurung (()). Tidak ada spasi antar *token*. Digunakan notasi 9^(0.5) untuk menuliskan notasi akar 2 dan notasi 2^(-1) untuk menuliskan notasi 1⁄2 pada *command prompt*. Dalam implementasi perhitungan pada kalkulator, digunakan aturan sebagai berikut.

1. Operasi yang berada dalam kurung dikerjakan lebih dahulu.
2. Urutan eksekusi operasi mengikuti urutan eksekusi matematika.
3. Dikerjakan berurutan dari kiri, kecuali untuk pangkat.

Operan terdiri dari bilangan riil (positif dan negatif) dari digit (0, 1, 2, 3, 4, 5, 6, 7, 8, 9). Kelompok membuat program sebagai implementasi dari CFG yang dibuat kelompok terlebih dahulu, menggunakan teori yang telah dipelajari.

b. Solusi

Program dibuat dalam bahasa pemrograman prosedural, yaitu C, dengan menggunakan nama variabel/fungsi, indentasi, dan komentar yang jelas. Program menerima masukan dan menampilkan keluaran pada terminal. Keluaran berupa hasil perhitungan valid jika notasi perhitungan valid, tulisan "SYNTAX ERROR" apabila notasi perhitungan tidak valid, atau tulisan "MATH ERROR" apabila ekspresi tidak mungkin dihitung. Algoritma implementasi yang digunakan oleh program kelompok adalah CYK.

CFG (Context-Free Grammar)

a. Dasar Teori

Context-free grammar (CFG) atau tata bebas konteks adalah notasi formal untuk menggambarkan definisi rekursif dari suatu Bahasa. Dengan pendekatan formal tersebut, *compiler* suatu bahasa pemrograman dapat dibuat lebih mudah dan menghindari ambiguitas ketika melakukan *parsing* bahasa tersebut. Proses *parsing* adalah proses pembacaan *string* dalam bahasa sesuai CFG tertentu. Proses ini harus mematuhi aturan produksi dalam CFG tersebut.

CFG menjadi dasar dalam pembentukan suatu *parser* / proses analisis sintaksis, yang merupakan bagian penting dalam suatu *compiler* ataupun *grammar checker*. Bagian sintak dalam suatu *compiler* atau *grammar checker* umumnya didefinisikan dalam tata bahasa bebas konteks. Pohon penurunan (*derivation tree* / *parse tree*) berguna untuk menggambarkan simbol-simbol variabel menjadi simbol-simbol terminal setiap simbol variabel akan di turunkan menjadi terminal sampai tidak ada yang belum tergantikan.

Sedangkan metode *Cocke-Younger-Kasami* (CYK) adalah suatu metode parsing yang digunakan dengan CFG. Metode CYK bersifat *bottom-up*, yang artinya *parsing* dimulai dari setiap masukan yang sudah berupa anggota terminal CFG, lalu dicocokkan dengan simbol variabel yang ada pada aturan produksi CFG tersebut. Jika pada tahap akhir *parsing* diperoleh simbol variabel Start (umumnya dituliskan sebagai S), maka ekspresi masukan tersebut memenuhi aturan produksi suatu CFG. Pada metode ini, CFG harus dalam bentuk *Chomsky normal form* (CNF), yaitu setiap variabel pada aturan produksi menghasilkan rangkaian simbol lainnya, anggota terminal, atau epsilon (ε).

b. Implementasi

Dalam tugas ini, kelompok menggunakan aturan produksi CFG sebagai berikut untuk memeriksa kurung berpasangan dalam ekspresi matematik yang dimasukkan:

- D → EF | EG
- E → (
- F → GD
- G → EH |)
- H → GG

Sedangkan untuk memeriksa kesalahan sintak lain selain kesalahan tanda kurung, digunakan aturan produksi CFG sebagai berikut:

- S → II | IJ | IK | NI | NJ | NK | JI | JK | K | OI | OK | PI | PK | KJ | KL | KM
- I → II | IJ | IK | NI | NJ | NK
- J → JI | JK | OI | OK | PI | PK
- K → KJ | KL | KM | bilangan riil
- L → LI | LJ | LK | QI | QJ | QK | RI | RJ | RK | TI | TJ | TK
- M → MJ | ML | MM |)
- N → (
- O → -

- P → +
- Q → *
- R → /
- T → ^

Sehingga contoh penerapan dari CFG tersebut menggunakan metode CYK adalah:

Masukan: 2+3-5

S, K				
S	S			
S, K		S, K		
	S, J		S, J	
K	P	K	O	K
2	+	3	-	5

➔ Karena tahap akhir CYK menghasilkan S, maka masukan memenuhi aturan produksi CFG.

Dalam pembuatan CFG, kelompok mengasumsikan bahwa input .<angka> dan <angka>+–<angka> akan menghasilkan SYNTAX ERROR. Asumsi-asumsi lainnya disamakan dengan hasil yang diperoleh pada kalkulator CASIO fx-991ES.

Source Code

Kode program dibuat menggunakan bahasa C dan mengimplementasikan CFG dengan metode CYK, dengan menggunakan bantuan tipe data berupa *array* dan matriks yang keduanya berelemen *string*. Kode program dimulai dengan penerimaan masukan dari pengguna berupa *string*.

Kemudian, program melaksanakan tiga tahap untuk melakukan perhitungan berdasarkan string yang telah dimasukkan oleh pengguna. Ketiga tahap tersebut antara lain penerjemahan, di mana program membagi *string* berdasarkan tipe *substring* masing-masing ke dalam sebuah *array*; kemudian tahap pengecekan sintak, ketika CFG dan metode CYK diimplementasikan bersama tipe data matriks; dan yang terakhir adalah tahap perhitungan, ketika program sudah memeriksa sintak dari permasalahan yang dimasukkan pengguna dan memproses setiap elemen *array* untuk menghasilkan bilangan riil yang merupakan solusi dari permasalahan tersebut.

a. Penerjemahan

Dalam eksekusinya, tahap ini menggunakan sebuah *header* yang bernama parse.h dan *body* yang bernama parse.c.

i. Fungsi **IsOp** dan **IsNum**

Kedua fungsi ini digunakan untuk mengecek apakah karakter yang sedang dibaca oleh program merupakan operasi atau angka. Fungsi IsOp mengembalikan *true* jika karakter merupakan tanda kurung buka atau kurung tutup ((), tanda plus (+), tanda minus (-), tanda kali (*), tanda bagi (/), atau tanda pangkat (^). Sedangkan fungsi IsNum mengembalikan *true* jika karakter merupakan bilangan antara 0 sampai 9 atau tanda titik (.) yang menyatakan pecahan.

```
boolean IsOp (char c) {
    return ((c == '(') || (c == ')') || (c == '+') || (c == '-') || (c
== '*') || (c == '/') || (c == '^'));
}

boolean IsNum (char c) {
    return ((c == '0') || (c == '1') || (c == '2') || (c == '3') || (c
== '4') || (c == '5') || (c == '6') || (c == '7') || (c == '8') || (c
== '9') || (c == '.'));
}
```

ii. Prosedur **parse**

Prosedur ini merupakan prosedur yang menjalankan proses penerjemahan *string* masukan pengguna menjadi komponen-komponennya dan memasukkannya ke dalam sebuah *array* berelemen *string*. Jika karakter yang dibaca merupakan tanda operasi, maka program memasukkannya ke dalam sebuah sel pada *array*. Jika karakter yang dibaca merupakan sebuah bilangan dari 0 sampai 9 atau tanda titik (.).

```
void parse (string s, arrString * arr) {
```

```
    int i, j;
    i = 0;
    j = 0;
    (*arr).Neff = 0;

    while (i < strlen(s)) {
        string temp = "";

        if (IsOp(s[i])) {
            temp[0] = s[i];
            (*arr).T[j][0] = temp[0];
            (*arr).Neff++;
            i++;
            j++;

        } else if (IsNum(s[i])) {
            temp[0] = s[i];
            int k = 1;
            i++;
            while ((IsNum(s[i])) && (i < strlen(s))) {
                temp[k] = s[i];
                i++;
                k++;
            }
            for (int a = 0; a <= strlen(temp); a++) {
                (*arr).T[j][a] = temp[a];
            }
            (*arr).Neff++;
            j++;

        } else {
            temp[0] = s[i];
            (*arr).T[j][0] = temp[0];
            (*arr).Neff++;
            i++;
            j++;
        }
    }
}
```

iii. Fungsi **toDouble**

Fungsi ini digunakan untuk menerjemahkan *string* yang berupa kumpulan angka dan mengembalikan hasil berupa bilangan riil (*double*).

```
double toDouble (string s) {
    int i;
    double x;
```

```
    double dec;

    x = 0;
    i = 0;
    while (s[i] != '.' && i<strlen(s)) {
        x = (x * 10) + (s[i] - '0');
        i++;
    }

    dec = 1;
    i++;

    while (i < strlen(s)) {
        dec = dec * 10;
        x = (x * 10) + (s[i] - '0');
        i++;
    }

    x = x / dec;
    return x;
}
```

b. Pengecekan Sintaks

Dalam eksekusinya, tahap ini menggunakan sebuah *header* yang bernama syntax.h dan *body* yang bernama syntax.c.

i. Fungsi **MakeBracketsArr**

Fungsi ini membaca *array* dari hasil tahap penerjemahan dan membuat *array* yang isinya hanya tanda kurung untuk dicek apakah semua tanda kurung sudah memiliki pasangannya.

```
arrString MakeBracketsArr (arrString arr) {
    arrString arrbracket;
    int i,j;
    j=0;
    arrbracket.Neff = 0;
    for (i=0;i<arr.Neff;i++){
        if (arr.T[i][0] == '(' || arr.T[i][0] == ')'){
            arrbracket.T[j][0] = arr.T[i][0];
            arrbracket.Neff +=1;
            j++;
        }
    }
    return arrbracket;
}
```

ii. Fungsi **VariableBrackets**

Fungsi ini mencari variabel asal dari sebuah *string* yang berbentuk 2 variabel atau 1 terminal dari aturan produksi kurung berpasangan, dan mengembalikan variabel asal jika ada atau 'Z' bila tidak ada. Contoh: Jika A → AB | a, maka VariableBrackets("AB") akan menghasilkan 'A'.

```
char VariableBrackets (string s){
    if (strcmp(s,"EF") == 0 || strcmp(s,"EG") == 0){
        return 'D';
    }else if(strcmp(s,"EH")==0 || strcmp(s,")")==0){
        return 'G';
    }else if (strcmp(s,"GD")==0){
        return 'F';
    }else if (strcmp(s,"(")==0){
        return 'E';
    }else if (strcmp(s,"GG")==0){
        return 'H';
    }else{
        return 'Z';
    }
}
```

iii. Fungsi **BracketsCheck**

Fungsi ini mengecek apakah tanda kurung pada *array* hasil fungsi MakeBrackets sudah berpasangan dengan menggunakan algoritma CYK, dengan bantuan tipe data berupa matriks. Fungsi mengembalikan nilai *true* jika tanda kurung berpasangan, dan mengembalikan *false* jika terdapat tanda kurung yang tidak berpasangan.

```
boolean BracketsCheck (arrString arr){
    MATRIKSCYK M;
    int i,j,k,l,m,n,o,p,kurang;
    string temp;
    char c;
    boolean found;

    if (arr.Neff == 0){
        return true;
    }else{
        M.NBrsEff = arr.Neff;
        M.NKolEff = arr.Neff;
        for (i=0;i<arr.Neff;i++){
            M.Mem[i][i][0] = VariableBrackets(arr.T[i]);
        }

        kurang=1;
        for (m=1;m<arr.Neff;m++){
            for (j=m;j<arr.Neff;j++){
```

```
                i=j-kurang;
                n=0;
                k=i;
                l=k+1;
                while(l<=j){
                    if (strlen(M.Mem[i][k])!=0 &&
                    strlen(M.Mem[l][j])!=0){
                        for (o=0;o<strlen(M.Mem[i][k]);o++){
                            for(p=0;p<strlen(M.Mem[l][j]);p++){
                                temp[0]=M.Mem[i][k][o];
                                temp[1]=M.Mem[l][j][p];
                                c = VariableBrackets(temp);
                                if (c!='Z'){
                                    M.Mem[i][j][n]=c;
                                    n++;
                                }
                            }
                        }
                    }
                    k++;
                    l++;
                }
            }
            kurang++;
        }

        found=false;
        i=0;
        while(i<strlen(M.Mem[0][arr.Neff-1]) && !found){
            if (M.Mem[0][arr.Neff-1][i] == 'D'){
                found=true;
            }else{
                i++;
            }
        }

        if (found){
            return true;
        }else{
            return false;
        }
    }
}
```

iv. Prosedur **Variable**

Prosedur ini mencari variabel asal dari sebuah *string* yang berbentuk 2 variabel atau 1 terminal dari aturan produksi kurung berpasangan, dan mengembalikan variabel asal jika ada atau 'Z' bila tidak ada. Contoh: Jika A → AB | a, maka Variable("AB") akan menghasilkan 'A'.

```
void Variable (string s, string *temp, int *neff){
    int i;

    i=0;
    *neff =0;

    if (strcmp(s,"II") == 0 || strcmp(s,"IJ") == 0 || strcmp(s,"IK")
    == 0 ||strcmp(s,"NI") == 0 || strcmp(s,"NJ") == 0 ||
    strcmp(s,"NK") == 0 || strcmp(s,"JI") == 0 || strcmp(s,"JK") == 0
    || strcmp(s,"OI") == 0 || strcmp(s,"OK") == 0 || strcmp(s,"PI") ==
    0 || strcmp(s,"PK") == 0 || strcmp(s,"KJ") == 0 || strcmp(s,"KL")
    == 0 || strcmp(s,"KM") == 0 || IsNum(s[0])){
        *temp[i]='S';
        *neff += 1;
        i++;
    }

    if(strcmp(s,"II")==0 || strcmp(s,"IJ")==0||strcmp(s,"IK") == 0 ||
    strcmp(s,"NI") == 0 || strcmp(s,"NJ") == 0 || strcmp(s,"NK") ==
    0){
             *temp[i]='I';
             *neff +=1;
             i++;
    }

    if (strcmp(s,"JI")==0 || strcmp(s,"JK") == 0 || strcmp(s,"OI") ==
    0 || strcmp(s,"OK") == 0||strcmp(s,"PI") == 0 || strcmp(s,"PK") ==
    0){
        (*temp)[i]='J';
        *neff +=1;
        i++;
    }

    if (strcmp(s,"KJ")==0 || strcmp(s,"KL") == 0 || strcmp(s,"KM") ==
    0 || IsNum(s[0])){
         (*temp)[i]='K';
        *neff +=1;
        i++;
    }

    if (strcmp(s,"LI")==0 || strcmp(s,"LJ") == 0 || strcmp(s,"LK") ==
    0 || strcmp(s,"QI") == 0 || strcmp(s,"QJ") == 0 || strcmp(s,"QK")
```

```
== 0 || strcmp(s,"RI") == 0 || strcmp(s,"RJ") == 0 ||
strcmp(s,"RK") == 0 || strcmp(s,"TI") == 0 || strcmp(s,"TJ") == 0
|| strcmp(s,"TK") == 0){
    (*temp)[i]='L';
    *neff +=1;
    i++;
}

if(strcmp(s,"MJ")==0 || strcmp(s,"ML") == 0 || strcmp(s,"MM") == 0
|| strcmp(s,")") == 0){
    (*temp)[i]='M';
    *neff +=1;
    i++;
}

if (strcmp(s,"(") == 0){
    (*temp)[i]='N';
    *neff +=1;
    i++;
}

if(strcmp(s,"-") == 0){
    (*temp)[i]='O';
    *neff +=1;
    i++;
}

if(strcmp(s,"+") == 0){
    (*temp)[i]='P';
    *neff +=1;
    i++;
}

if(strcmp(s,"*") == 0){
    (*temp)[i]='Q';
    *neff +=1;
    i++;
}

if(strcmp(s,"/") == 0){
    (*temp)[i]='R';
    *neff +=1;
    i++;
}

if(strcmp(s,"^") == 0){
    (*temp)[i]='T';
    *neff +=1;
```

```
        i++;
    }

    if(*neff ==0){
        (*temp)[0]='Z';
        *neff +=1;
    }
}
```

v. Fungsi **InputCheck**

Fungsi ini mengecek masukan yang sudah diterjemahkan dengan menggunakan algoritma CYK dan mengembalikan nilai *true* jika input diterima. Fungsi mengembalikan *false* jika input tidak diterima.

```
boolean InputCheck (arrString arr){
    MATRIKSCYK M;
    int i,j,k,l,m,n,o,p,q,kurang,neff;
    string temp,s;
    boolean found;

    if (arr.Neff == 0){
        return true;
    }else{
        M.NBrsEff = arr.Neff;
        M.NKolEff = arr.Neff;
        for (i=0;i<arr.Neff;i++){
            n=0;
            Variable(arr.T[i],&s,&neff);
            for (q=0; q<neff;q++){
                if (s[q] != 'Z'){
                    M.Mem[i][i][n] = s[q];
                    n++;
                }
            }
        }
        kurang=1;
        for (m=1;m<arr.Neff;m++){
            for (j=m;j<arr.Neff;j++){
                i=j-kurang;
                n=0;
                k=i;
                l=k+1;
                while(l<=j){
                    if (strlen(M.Mem[i][k]) != 0 &&
                    strlen(M.Mem[l][j]) !=0){
                        for (o=0;o<strlen(M.Mem[i][k]);o++){
                            for(p=0;p<strlen(M.Mem[l][j]);p++){
```

```
                                temp[0]=M.Mem[i][k][o];
                                temp[1]=M.Mem[l][j][p];
                                Variable(temp,&s,&neff);
                                for(q=0;q<neff;q++){
                                    if (s[q]!='Z'){
                                        M.Mem[i][j][n]=s[q];
                                        n++;
                                    }
                                }
                            }
                        }
                    }
                    k++;
                    l++;
                }
            }
            kurang++;
        }
        found=false;
        i=0;
        while(i<strlen(M.Mem[0][arr.Neff-1]) && !found){
            if (M.Mem[0][arr.Neff-1][i] == 'S'){
                found=true;
            }else{
                i++;
            }
        }
        if (found){
            return true;
        }else{
            return false;
        }
    }
}
```

vi. Fungsi **IsNumValid**

Fungsi ini memeriksa banyak tanda titik (.) pada angka. Angka valid bila banyak tanda titik 0 atau 1. Angka tidak valid bila banyak tanda titik lebih dari 1. Fungsi mengembalikan nilai *false* bila ada angka yang tidak valid, *true* jika semua angka valid.

```
boolean IsNumValid (arrString arr){
    boolean valid;
    int count;
    int i,j;

    valid = true;
    i=0;
```

```
    while(i<arr.Neff && valid){
        if (IsNum(arr.T[i][0])){
            j=0;
            count=0;
            while (j<strlen(arr.T[i])){
                if (arr.T[i][j] == '.'){
                    count+=1;
                }
                j++;
            }
            if(count>1){
                valid = false;
            }
        }
        i++;
    }
    return valid;
}
```

c. Perhitungan

Dalam eksekusinya, tahap ini menggunakan sebuah *header* yang bernama calculate.h dan *body* yang bernama calculate.c.

i. Fungsi **PrioBracket**

Fungsi ini mencari dan mengembalikan indeks *array* yang akan dihitung dahulu menurut prioritas berdasarkan tanda kurung.

```
int PrioBracket (arrString arr){
    int i,prio;
    i = 0;
    prio = 0;
    while (i<arr.Neff && arr.T[i][0] != ')'){
        if (arr.T[i][0] == '('){
            prio=i+1; //setelah '(' pasti ada karakter lain
        }
        i++;
    }
    return prio;
}
```

ii. Fungsi **PrioOperator**

Fungsi ini mencari dan mengembalikan nilai indeks operator yang pertama harus dihitung.

```
int PrioOperator (arrString arr, int i){
    int j,prio;
```

```
    boolean found1,found2,found3;
    found1 = false;
    found2 = false;
    found3 = false;
    prio=i;
    j=i;
    while(j<arr.Neff && arr.T[j][0] != ')'){
        if (arr.T[j][0] == '^'){
            prio = j;
            found1=true;
        }
        j++;
    }
    if (!found1){
        j=i;
        while(j<arr.Neff && arr.T[j][0] != ')' && !found2){
            if (arr.T[j][0] == '*' || arr.T[j][0] == '/'){
                prio = j;
                found2=true;
            }
            j++;
        }
        if(!found2){
            j=i;
            while(j<arr.Neff && arr.T[j][0] != ')' && !found3){
                if (arr.T[j][0] == '+' || arr.T[j][0] == '-'){
                    prio = j;
                    found3=true;
                }
                j++;
            }
        }
    }
    return prio;
}
```

iii. Prosedur **toArrDouble**

Prosedur ini membentuk sebuah array berelemen double yang berisi seluruh angka dari arrString, dengan indeks yang sama dengan arrString.

```
void toArrDouble (arrString arr, arrDouble *arrd){
    int i;
    (*arrd).Neff = arr.Neff;
    for (i=0;i<arr.Neff;i++){
        if(IsNum(arr.T[i][0])){
             (*arrd).T[i] = toDouble(arr.T[i]);
        }
```

```
    }
}
```

iv. Prosedur **DelTwo**

Prosedur ini digunakan untuk menghapus karater di antara suatu indeks i, mengubah nilai elemen berindeks i menjadi X, dan kemudian menggeser elemen-elemen lainnya.

```
void DelTwo (arrString *arr, int i, char X){
    int j;
    (*arr).T[i][0] = X;
    strcpy((*arr).T[i-1],(*arr).T[i]);
    for (j=i;j<((*arr).Neff-2);j++){
        strcpy((*arr).T[j],(*arr).T[j+2]);
    }
    (*arr).Neff -= 2;
}
```

v. Prosedur DelTwoDouble

Prosedur ini digunakan untuk menghapus letak elemen di antara suatu indeks i agar indeks angka tetap terjaga sama dengan arrString. Nilai elemen pada indeks i diubah menjadi X.

```
void DelTwoDouble (arrDouble *arrd, int i, double X){
    int j;
    (*arrd).T[i] = X;
    (*arrd).T[i-1]=(*arrd).T[i];
    for (j=i;j<((*arrd).Neff-2);j++){
        (*arrd).T[j]=(*arrd).T[j+2];
    }
    (*arrd).Neff -= 2;
}
```

vi. Prosedur **DelOne**

Prosedur ini menghapus elemen setelah suatu indeks i, mengubah nilai elemen pada indeks i menjadi X, dan menggeser elemen-elemen lainnya.

```
void DelOne (arrString *arr, int i, char X){
    int j;
    (*arr).T[i][0] = X;
    for (j=i+1;j<((*arr).Neff-1);j++){
        strcpy((*arr).T[j],(*arr).T[j+1]);
    }
    (*arr).Neff -= 1;
}
```

vii. Prosedur DelOneDouble

Prosedur ini menghapus elemen setelah suatu indeks i agar indeks angka tetap terjaga sama dengan arrString, mengubah elemen pada indeks i menjadi X, dan menggeser elemen lainnya.

```
void DelOneDouble (arrDouble *arrd, int i, double X){
    int j;
    (*arrd).T[i] = X;
    for (j=i+1;j<((*arrd).Neff-1);j++){
        (*arrd).T[j] =(*arrd).T[j+1];
    }
    (*arrd).Neff -= 1;
}
```

viii. Fungsi lilCalc dan Calculate

Kedua fungsi ini digunakan untuk melakukan proses perhitungan, dengan bantuan sebuah array bertipe string dan array bertipe double. Fungsi mengembalikan MATH ERROR jika teradapat ekspresi matematika yang tidak dapat dihitung, dan mengembalikan hasil perhitungan bila semua ekspresi matematika terbukti sahih. Fungsi lilCalc adalah sub-fungsi dari fungsi Calculate.

```
double lilCalc (double angka1, double angka2, char Op){
    double result;
    if (Op == '^') {
        result = (float) pow((double) angka1,angka2);
    } else
    if (Op == '/') {
        result = (angka1 / angka2);
    }else
    if (Op == '*') {
        result = (angka1 * angka2);
    } else
    if (Op == '+') {
        result = (angka1 + angka2);
    } else
    if (Op == '-') {
        result = (angka1 - angka2);
    }
    return result;
}

void calculate (arrString *arr,arrDouble *arrd, boolean *error){
    int i,j;
    float temp;
    char tempc;
    *error=false;
    while ((*arr).Neff > 1) {
        i = PrioBracket(*arr);
```

```
j = PrioOperator(*arr, i);
if (((*arr).T[j][0] == '^') || ((*arr).T[j][0] == '/')) {
    if ((*arr).T[j][0] == '^') {
        if (fmod((1/(*arrd).T[j+1]), 2.0) == 0 && (*arrd).T[j-
        1] <0) {
            printf("\n");
            printf("MATH ERROR\n");
            (*arr).Neff = 1;
            *error=true;
        }
    }else
    if ((*arr).T[j][0] == '/') {
        if ((*arrd).T[j+1] == 0) {
            printf("\n");
            printf("MATH ERROR\n");
            (*arr).Neff = 1;
            *error=true;
        }
    }
}
if((*arr).Neff != 1) {
    if ((*arr).T[j-1][0] == '(') {
        if (IsNum((*arr).T[j+1][0])) {
            if((*arr).T[j][0] == '-'){
                temp = ((*arrd).T[j+1]) * -1;
            }else{
                temp = ((*arrd).T[j+1]);
            }
            tempc = '0';
            DelOne(arr,j,tempc);
            DelOneDouble(arrd,j,temp);
        } else
        if ((*arr).T[j+1][0] == ')') {
            temp = ((*arrd).T[j]);
            tempc = '0';
            DelTwo(arr,j, tempc);
            DelTwoDouble(arrd,j,temp);
        }
        //else if (IsOp((*arr).T[j+1][0])){}
    } else
    if (IsNum((*arr).T[j-1][0])) {
        temp = lilCalc (((*arrd).T[j-1]), ((*arrd).T[j+1]),
        (*arr).T[j][0]);
        tempc = '0';
        DelTwo(arr,j,tempc);
        DelTwoDouble(arrd,j,temp);
    } else
    if (j==0){
```

```
                if(j+1<(*arr).Neff){
                    if((*arr).T[j][0] == '-'){
                        temp = ((*arrd).T[j+1]) * -1;
                    }else{
                        temp = ((*arrd).T[j+1]);
                    }
                    tempc = '0';
                    DelOne(arr,j,tempc);
                    DelOneDouble(arrd,j,temp);
                }
            }
        }
    }
}
```

d. Program Utama

```
int main(){
    string input;
    arrString arr;
    arrString arrbracket;
    arrDouble arrd;
    boolean valid_num;
    boolean valid_bracket;
    boolean valid_input;
    boolean math_error;

    /*menerima input dari pengguna dan mem-parse input*/
    printf("Silakan masukkan perhitungan yang akan dikalkulasi:\n");
    scanf("%s",input);
    parse(input, &arr);

    /*mengecek banyak '.' pada setiap angka*/
    valid_num = IsNumValid(arr);
    if(!valid_num){
        printf("\n");
        printf("SYNTAX ERROR\n");
    }else{
        /*jika semua angka memiliki banyak '.' <=1*/

        /*mengecek tanda kurung berpasangan*/
        arrbracket = MakeBracketsArr(arr);
        valid_bracket = BracketsCheck(arrbracket);
        if(!valid_bracket){
            printf("\n");
            printf("SYNTAX ERROR\n");
        }else{ //semua tanda kurung berpasangan
```

```
            /*mengecek syntax error lainnya*/
            valid_input = InputCheck(arr);
            if(!valid_input){
                printf("\n");
                printf("SYNTAX ERROR\n");
            }else{
                /*lolos syntax error, melakukan kalkulasi*/
                toArrDouble(arr,&arrd);
                calculate(&arr,&arrd,&math_error);
                if(!math_error){
                    printf("\n");
                    if(arrd.T[0] == -0){
                        printf("%f\n",arrd.T[0]*-1);
                    }else{
                        printf("%f\n",arrd.T[0]);
                    }
                }
            }
        }
    }
    return 0;
}
```

Contoh Masukan dan Keluaran

Uji Coba 1: Sintak matematis yang sahih.

```
Silakan masukkan perhitungan yang akan dikalkulasi:
200.5*750-237^1
150138.000000
```

Uji Coba 2: Penggunaan simbol operasi yang salah.

```
Silakan masukkan perhitungan yang akan dikalkulasi:
3+-5*2/7
SYNTAX ERROR
```

Uji Coba 3: Penggunaan tanda titik yang salah.

```
Silakan masukkan perhitungan yang akan dikalkulasi:
400..0*200
SYNTAX ERROR
```

Uji Coba 4: Tanda kurung yang tidak berpasangan.

```
Silakan masukkan perhitungan yang akan dikalkulasi:
(-2)*3+((5)
SYNTAX ERROR
```

Uji Coba 5: Pembagian dengan 0.

```
Silakan masukkan perhitungan yang akan dikalkulasi:
1+(2/0)-1
MATH ERROR
```

Uji Coba 6: Akar imajiner.

```
Silakan masukkan perhitungan yang akan dikalkulasi:
(-2)^(1/2)
MATH ERROR
```