Dokumentasi
Tugas Besar II IF2220 Teori Bahasa Formal dan Otomata

Aplikasi CFG dan PDA pada Pengenalan Ekspresi Matematika

oleh

Adyaksa Wisanggeni / 13517091
Aidil Rezjki Suljztan Syawaludin / 13517070
Edward Alexander Jaya / 13517115

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2018

1. Deskripsi Persoalan

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 besar kedua untuk matakuliah **Teori Bahasa Formal dan Otomata** ini, akan dibuat sebuah kalkulator sederhana, yang menggunakan implementasi tata bahasa bebas konteks (CFG) dan/atau *pushdown automata* (PDA). Ketika kalkulator tersebut diberikan sebuah ekspresi matematika, program harus bisa mengenali apakah ekspresi tersebut valid atau tidak (*syntax error*).

Bila ekspresi yang diterima oleh program sudah 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 mungkin dihitung atau tidak (*math error*).

Contoh ekspresi matematika yang valid adalah sebagai berikut.

(-457.01+1280) * (35.7-11.0233)/(-6.1450)

Setelah ekspresi tersebut dimasukkan ke program, maka program akan menampilkan hasil perhitungan ekspresi tersebut yaitu -3304.91).

Contoh ekspresi yang tidak valid adalah sebagai berikut.

3*+-12/(57)

Setelah ekspresi tersebut dimasukkan ke program,program akan menampilkan pesan “SYNTAX ERROR”).

Pada Tugas Besar kali ini, 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 ½ 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, kecuali untuk operator pangkat.

(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.

Adapun pada Tugas Besar kali ini, terdapat soal bonus, yaitu dapat menerima masukan imajiner dan mengeluarkan hasil imajiner. Kami tidak seluruhnya mengerjakan Bonus pada Tugas Besar kali ini, karena hanya ekspresi seperti

(-5)^(2/3)

yang dapat diatasi oleh program bilangan imajiner kami.

2. Deskripsi Solusi

Pada pembuatan program kalkulator ini, dibuat algoritma program yang merupakan implementasi dari tata bahasa bebas konteks (*Context Free Grammar*). Program ini akan menerima sebuah masukan dari pengguna yang akan dibaca sebagai sebuah *string*. Setelah itu, *string* tersebut akan dievaluasi oleh implementasi dari tata bahasa bebas konteks.

Adapun *Context Free Grammar* yang kami gunakan dikembangkan lagi menjadi teknik *recursive descent parsing*, sebuah teknik yang menggunakan beberapa prosedur yang memanggil prosedur lainnya (mirip dengan rekursi).

Adapun prosedur-prosedur yang kami gunakan dalam tugas besar ini adalah *number*, *item*, *factor, term*, dan *expression*. *Number* mendefinisikan prosedur untuk mengecek apakah suatu string merupakan angka atau bukan. *Item* mendefinisikan prosedur untuk mengecek angka yang dimasukkan baik berupa integer ataupun *double* melalui *Number*, lalu mengecek ekspresi seperti '(', dan mengecek masukkan acak (tentunya tidak sahih). *Factor* mendefinisikan prosedur untuk mengecek *Item* dan melakukan *Item* dipangkatkan. *Term* mendefinisikan prosedur untuk melakukan perkalian atau pembagian terhadap *Factor*. Sedangkan *Expression* mendefinisikan prosedur untuk melakukan penjumlahan atau pengurangan terhadap *Term*.

Ketika *string* yang dimasukkan oleh pengguna termasuk ke dalam bahasa yang diterima oleh tata bahasa bebas konteks tersebut, maka akan dilakukan penghitungan terhadap ekspresi matematika tersebut. Jika *string* tersebut tidak termasuk bahasa yang diterima oleh tata bahasa bebas konteks pada program, maka akan ditampilkan pesan "Ekspresi tidak valid" di layar.

3. Notasi Tata Bahasa Bebas Konteks

3.1 Notasi

Pada program kalkulator ini digunakan tata bahasa bebas konteks yang memenuhi notasi berikut.

$$G = (V, \Sigma, R, S)$$

Sebuah tata bahasa bebas konteks terdiri atas beberapa hal, yaitu:

1. V, yang merupakan himpunan terbatas dari variabel yang menyatakan status, frasa, atau klausa pada sebuah tata bahasa bebas konteks.
2. Σ, yang merupakan himpunan terbatas dari terminal. Terminal menyatakan alfabet dari suatu tata bahasa bebas konteks. Terminal merupakan penyusun dari kalimat yang termasuk dalam bahasa dari suatu tata bahasa bebas konteks.
3. R, yang merupakan fungsi produksi dari sebuah tata bahasa bebas konteks.
4. S, yang merupakan variabel awal atau status awal dari suatu tata bahasa bebas konteks.

3.2 Variabel

Berikut adalah variabel-variabel yang digunakan dalam tata bahasa bebas konteks pada program kalkulator ini.

{**E**, **T**, **F**, **I**, **N**, **A**, **B**, **C**}

3.3 Terminal

Berikut adalah himpunan dari terminal yang diterima oleh tata bahasa bebas konteks pada program kalkulator ini.

{**0**, **1**, **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**, **.** , +, **-**, ***, /, ^, **(**, **)**, }

3.4 Variabel Awal

Pada tata bahasa bebas konteks yang digunakan oleh program kalkulator ini, variabel yang merupakan variabel dan menandakan status awal adalah variabel **E**.

3.5 Fungsi Produksi

Pada tata bahasa bebas konteks yang digunakan oleh program kalkulator ini, terdapat beberapa fungsi produksi yang akan menghasilkan bahasa atau ekspresi yang diterima oleh tata bahasa bebas konteks tersebut. Adapun fungsi produksi tersebut adalah sebagai berikut.

Fungsi Produksi
E -> T | T + | T - **T -> F | F * | F /** **F -> I | I ^ F** **I -> N | (E) | (- E)** **N -> A | A . B | C . B | C** **A -> 1B | 2B | 3B | 4B | 5B | 6B | 7B | 8B | 9B** **B -> CB | C** C -> **0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9**

4. Kode Sumber

4.1 Kompilasi

Pada pembuatan program kalkulator yang mengimplementasikan tata bahasa bebas konteks ini, digunakan bahasa pemrograman C. Untuk melakukan kompilasi kode sumber yang ditulis dalam bahasa C, digunakan *compiler* yaitu GNU Compiler Collection C (GCC C). Kompilasi pada kode sumber dilakukan dengan perintah berikut.

gcc main.c -o main.exe

Perintah tersebut dimasukan pada *terminal*, *command prompt*, ataupun *powershell*. Setelah perintah tersebut dijalankan, maka akan dibuat sebuah *file* bernama *main.exe* yang merupakan hasil kompilasi dari kode sumber pada *file main.c*. Untuk menjalankan program tersebut, dapat dilakukan dengan menjalankan *main.exe* dengan langsung, ataupun dari *terminal*, *command prompt*, maupun *powershell*.

4.2 Kode Sumber

Pada program ini, hanya terdapat satu *file* yang merupakan keseluruhan dari kode sumber untuk program ini. Adapun *file* tersebut adalah *main.c*. Kode sumber program *main.c* dilampirkan di bawah ini:

File main.c

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

const double EPS = 0.0000000001;

char str[105], arr[105];
int isValid, mathError;

int tipe(char X){
    switch(X){
        case '+': return 1; break;
        case '-': return 1; break;
        case '*': return 2; break;
        case '/': return 2; break;
        case '^': return 3; break;
        case '(': return 4; break;
        case ')': return 5; break;
        default: if (X <= '9' && X >= '0') return 0;
                else return 6;
    }
}

complex modified_cpow(complex a, complex b){
    if (fabs(creal(a)) <= EPS && fabs(creal(b)) <= EPS && fabs(cimag(a)) <= EPS && fabs(cimag(b)) <= EPS){
        return 1;
    } else return cpow(a,b);
}

void inc(int *idx){
    do{
        (*idx)++;
    } while (str[*idx] == ' ');
}

void outputComplex(complex X){
    printf("%.8f + %.8fi\n", creal(X), cimag(X));
}

complex parse_expression(int *idx);

complex parse_number(int *idx){
    char t = str[*idx];
    char strcomplex[105];
    memset(strcomplex, 0, sizeof(strcomplex));
    int idxcomplex = 0;
    int countdot = 0;
    int isFirstNumber = 1;
    strcomplex[idxcomplex] = t;
    while ((tipe(t) == 0) || (t == '.')) {
        (*idx)++;
        t = str[*idx];
        if(isFirstNumber && tipe(t) == 0 && strcomplex[0] == '0'){
            isValid = 0;
        }
        isFirstNumber = 0;
        if ((tipe(t) == 0) || (t == '.')) {
            inc(&idxcomplex);
            strcomplex[idxcomplex] = t;
            if (t == '.')
                countdot = countdot + 1;
```

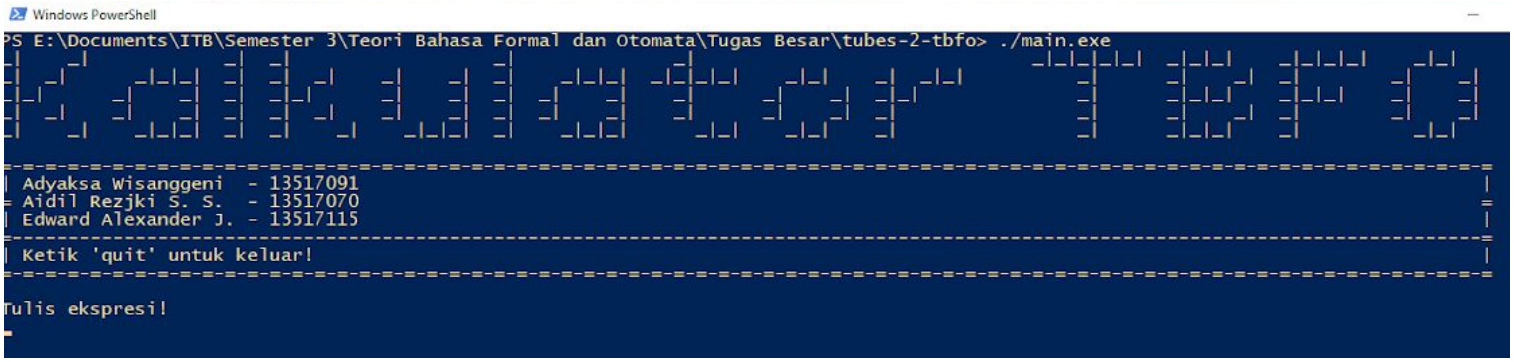
```
            if (countdot > 1)
                isValid = 0;
        }
    }
    if (strcomplex[idxcomplex] == '.'){
        isValid = 0;
        return 0;
    }
    else {
        return strtod(strcomplex,NULL);
    }
}

complex parse_item(int *idx){
    if (*idx == strlen(str)){
        isValid = 0;
    }
    char t = str[*idx];
    complex result = 0;
    if (tipe(t) == 0){
        result = parse_number(idx);
    } else if(t == '('){
        inc(idx);
        int prev = *idx;
        t = str[*idx];
        if (t == '-'){
            inc(idx);
            prev = *idx;
            result = -parse_expression(idx);
            if (cimag(result) < EPS){
                result = creal(result);
            }
            if (prev == *idx){
                isValid = 0;
                return result;
            }
        } else{
            result = parse_expression(idx);
            if (prev == *idx){
                isValid = 0;
                return result;
            }
        }

        if (prev == *idx){
            isValid = 0;
            return result;
        }
        if (str[*idx] == ')')
            inc(idx);
        else {
            isValid = 0;
            return result;
        }
    } else isValid = 0;
    return result;
}

complex parse_factor(int *idx){
    if (*idx == strlen(str)){
        isValid = 0;
    }
    char t = str[*idx];
    complex result;
    int prev = *idx;
    result = parse_item(idx);
    if (prev == *idx){
        isValid = 0;
        return result;
```

```
134.        }
135.        t = str[*idx];
136.        if (t == '^'){
137.            inc(idx);
138.            int prev = *idx;
139.            result = modified_cpow(result,parse_factor(idx));
140.            if (prev == *idx){
141.                isValid = 0;
142.                return result;
143.            }
144.        }
145.        return result;
146.    }
147.
148.    complex parse_term(int *idx){
149.        if (*idx == strlen(str)){
150.            isValid = 0;
151.        }
152.        complex result = parse_factor(idx);
153.        char t = str[*idx];
154.        while (tipe(t) == 2){
155.            inc(idx);
156.            int prev = *idx;
157.            complex rhs = parse_factor(idx);
158.            if (prev == *idx){
159.                isValid = 0;
160.                return result;
161.            }
162.
163.            if (t == '*') result = result*rhs;
164.            else if (rhs != 0) result = result/rhs;
165.            else if (t == '/'){
166.                mathError = 1;
167.            } else {
168.                isValid = 0;
169.            }
170.            t = str[*idx];
171.        }
172.        return result;
173.    }
174.
175.    complex parse_expression(int *idx){
176.        complex result = parse_term(idx);
177.        char t = str[*idx];
178.        while (tipe(t) == 1){
179.            inc(idx);
180.            int prev = *idx;
181.            complex rhs = parse_term(idx);
182.            if (prev == *idx){
183.                isValid = 0;
184.                return result;
185.            }
186.
187.            if (t == '+') result = result+rhs;
188.            else if (t == '-') result = result-rhs;
189.            else isValid = 0;
190.            t = str[*idx];
191.        }
192.        return result;
193.    }
194.
195.    int main(){
196.        //Bagian ini hanya hiasan.
197.        printf("_|    _|            _|  _|                _|              _|
    _|_|_|_|_|  _|_|_|    _|_|_|_|    _|_|    \n");
198.        printf("_|  _|      _|_|_|  _|  _|  _|    _|    _|  _|    _|_|_|  _|_|_|_|
    _|_|    _|  _|_|          _|      _|  _|  _|      _|    _|  _|  \n");
199.        printf("_|_|      _|    _|  _|  _|_|      _|    _|  _|  _|    _|    _|
    _|    _|  _|_|            _|      _|_|_|    _|_|_|    _|    _| \n");
```

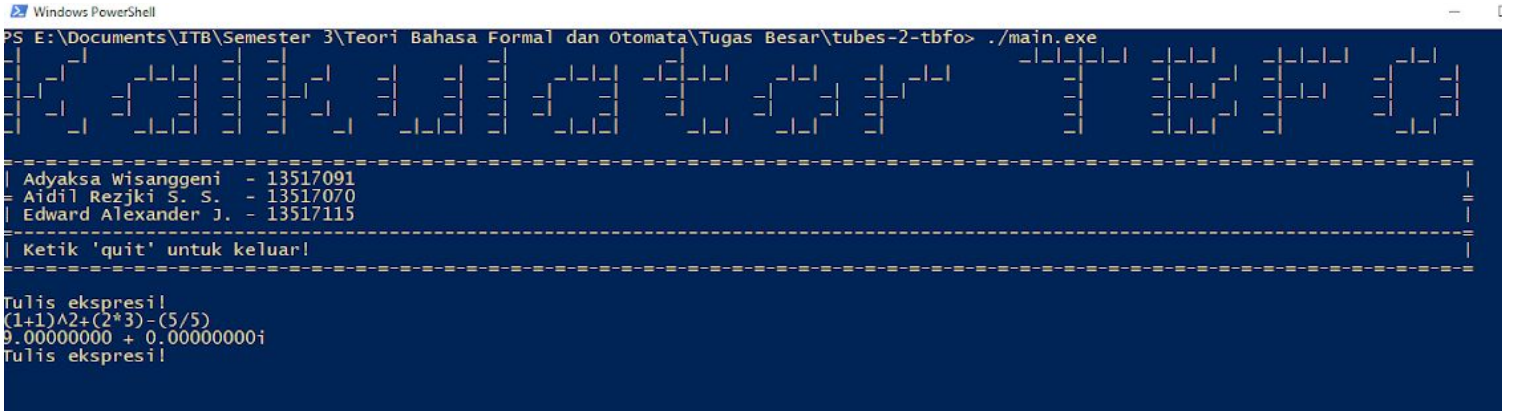
```
200.        printf("_|  _|    _|    _|  _|  _|  _|    _|    _|  _|  _|    _|    _|
    _|    _|  _|                  _|      _|    _|  _|        _|    _|  \n");
201.        printf("_|    _|    _|_|_|  _|  _|    _|    _|_|_|  _|    _|_|_|      _|_|
    _|_|    _|                _|      _|_|_|    _|          _|_|   \n");
202.        printf("
    \n");
203.
    printf("=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
    =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=\n");
204.        printf("| Adyaksa Wisanggeni  - 13517091
    |\n");
205.        printf("= Aidil Rezjki S. S.  - 13517070
    =\n");
206.        printf("| Edward Alexander J. - 13517115
    |\n");
207.
    printf("=-----------------------------------------------------------------------------
    ----------------------------------------------------------=\n");
208.        printf("| Ketik 'quit' untuk keluar!
    |\n");
209.
    printf("=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
    =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=\n");
210.        printf("
    \n");
211.        //Akhir dari hiasan.
212.
213.        while(strcmp(str, "quit") != 0){
214.            printf("Tulis ekspresi!\n");
215.            gets(str);
216.            if (strcmp(str, "quit") != 0){
217.                int x = strlen(str);
218.                int idx = 0;
219.                isValid = 1;
220.                complex res = parse_expression(&idx);
221.
222.                if(isValid && idx == strlen(str)){
223.                    if(mathError) printf ("Math error\n");
224.                    else outputComplex(res);
225.                } else printf("Ekspresi tidak valid\n");
226.            }else {
227.
228.            }
229.        }
230.
231.    }
```

5. Uji Coba Program

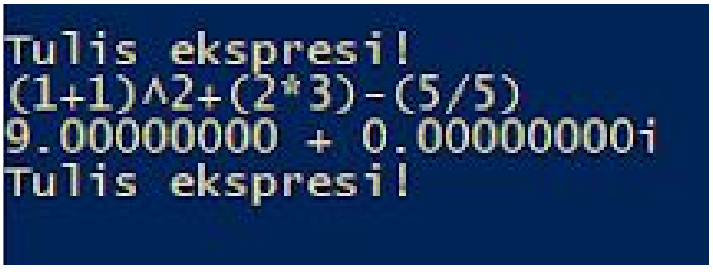
Pada bagian ini akan dilakukan uji coba terhadap program kalkulator ini. Hasil dari uji coba tersebut adalah sebagai berikut.



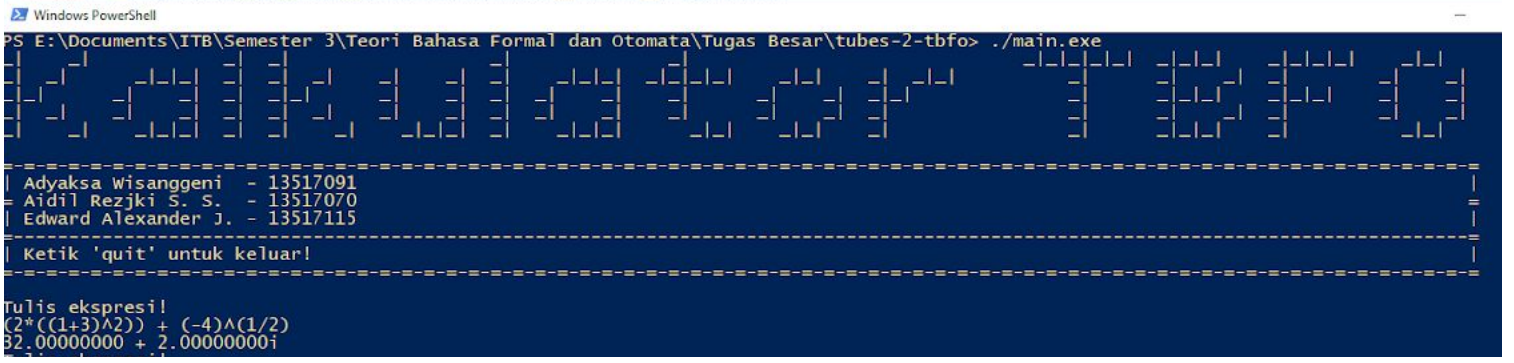
Gambar 5.1 Tampilan awal program.



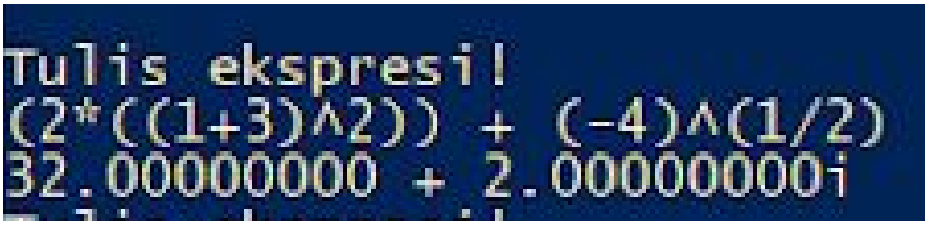
Gambar 5.2 Uji coba untuk ekspresi (1+1)^2+(2*3)-(5/5) yang hasilnya bilangan riil yaitu 9.



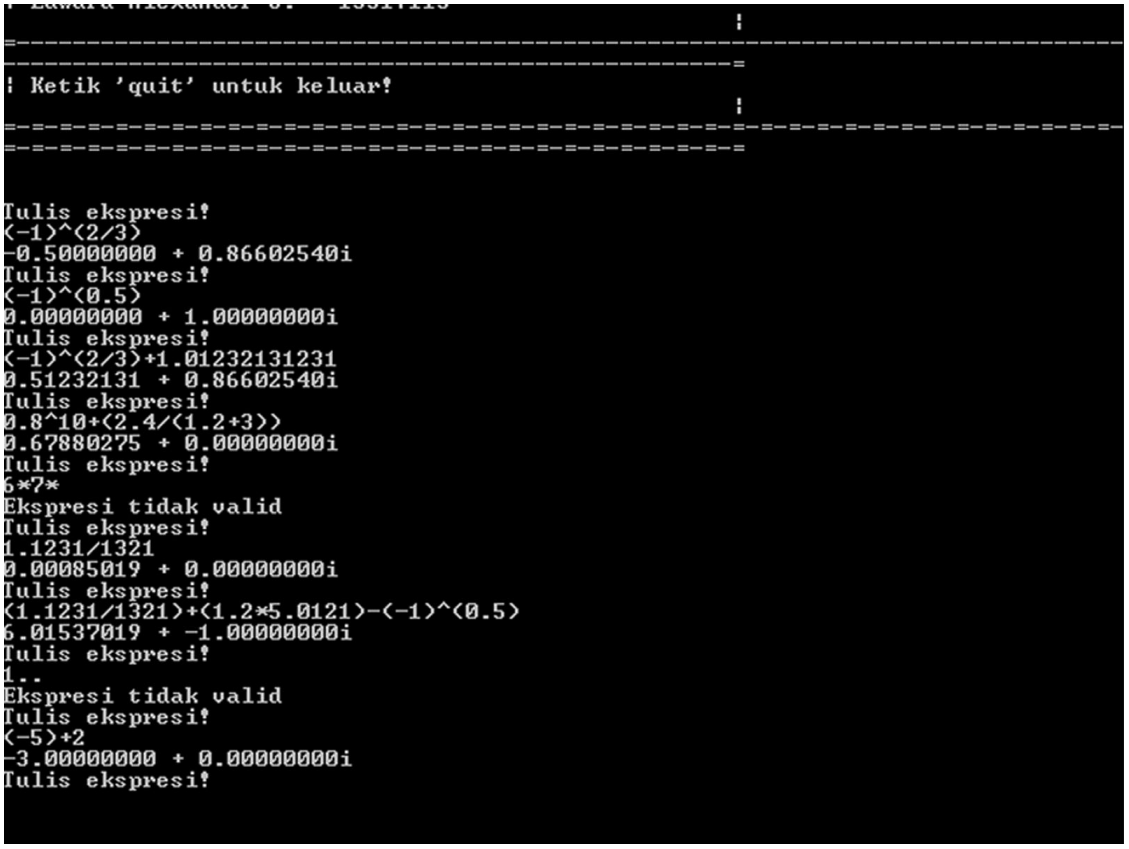
Gambar 5.3 Hasil dari ekspresi (1+1)^2+(2*3)-(5/5).



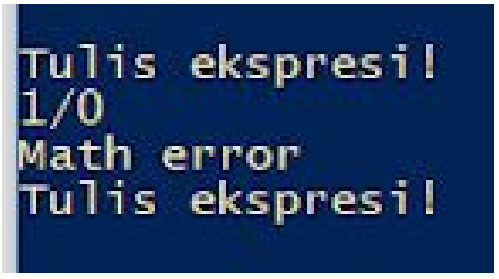
Gambar 5.4 Uji coba untuk ekspresi (2*((1+3)^2)) + (-4)^(1/2) yang melibatkan bilangan imajiner.



Gambar 5.5 Hasil dari ekspresi (2*((1+3)^2)) + (-4)^(1/2).



Gambar 5.6 Hasil dari beberapa ekspresi.



Gambar 5.7 Hasil dari ekspresi 1/0.

6. Daftar Pustaka dan Referensi

Context-free Grammar. (n.d.). Di *Wikipedia*. Diakses pada November 21, 2018, dari https://en.wikipedia.org/wiki/Context-free_grammar#Examples_of_languages_that_are_not_context_free

Recursive Descent Parser. (n.d.). Di *Wikipedia.* Diakses pada November 21, 2018, dari https://en.wikipedia.org/wiki/Recursive_descent_parser

Cheong, Otfried. (n.d.). Implementing a Calculator. Diakses pada November 21, 2018, dari http://otfried.org/courses/cs206/notes/calculator.pdf