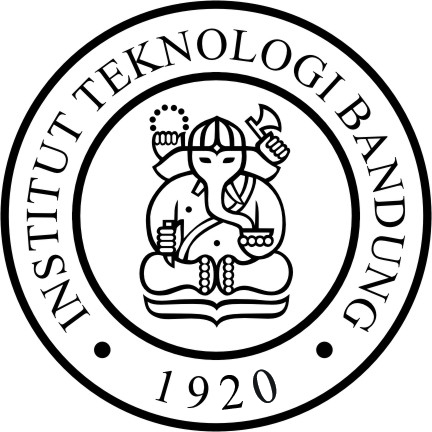
LAPORAN

TUGAS 2 IF2220 TEORI BAHASA FORMAL & OTOMATA

Aplikasi CFG dan PDA pada pengenalan ekspresi matematika



DISUSUN OLEH :

Kevin Jonathan Koswara – 13515052

Aulia Ichsan Rifkyano - 13515100

INSTITUT TEKNOLOGI BANDUNG

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

TEKNIK INFORMATIKA

2016

DAFTAR ISI

DAFTAR ISI.......................................................................................................................... ii

1. DESKRIPSI PERSOALAN............................................................................................... 1

2. JAWAB PERSOALAN...................................................................................................... 1

3. DAFTAR ASUMSI............................................................................................................ 2

4. CONTEXT FREE GRAMMAR......................................................................................... 2

5. SOURE CODE................................................................................................................... 3

a. kalkulator.c..................................................................................................................... 3

b. check.c........................................................................................................................... 6

c. check.h............................................................................................................................ 9

d. mesinkata.c..................................................................................................................... 9

e. mesinkata.h.................................................................................................................... 10

f. boolean.h........................................................................................................................ 10

g. stacklist.c......................................................................................................................... 11

h. stacklist.h........................................................................................................................ 12

6. CONTOH MASUKAN DAN KELUARAN...................................................................... 14

1. DESKRIPSI PERSOALAN

Desain sebuah program untuk mengenali dan menghitung ekspresi matematika menggunakan tata bahasa bebas konteks. Bila diberikan sebuah ekspresi matematika, maka program anda harus bisa mengenali apakah ekspresi tersebut valid atau tidak (sintaks error), kalau sudah valid maka program tersebut menghitung nilai dari ekspresi tersebut dengan mengubah terlebih dahulu setiap simbol terminal (angka) menjadi nilai numerik yang bersesuian. Contoh ekspresi matematika yang valid adalah (-457.01+1280) \* (35.7-11.0233)/(-6.1450) (setelah dienter akan menampilkan hasil perhitungan ekspresi tersebut yaitu -3304.91) . Contoh ekspresi tidak valid adalah 3\*+-12/(57) (setelah di enter akan ditampilkan pesan “SINTAK ERROR”

Batasan masalah:

Terminal terdiri dari {+, -, \*, /, (, ), 0, ...,9, .(tanda koma desimal)). Operator hanya terdiri dari { ‘+’, ‘-‘ , ‘\*,’ ‘/’ , ( ) }, operator ‘\*’ dan ‘/’ mempunyai prioritas lebih tinggi dari ‘+’ dan ‘–‘ kecuali kalau ada tanda ( ). Operan terdiri dari bilangan bulat dan bilangan desimal (baik positif atau negatif) dari digit {1....9,0}.

Program merupakan implementasi dari tatabahasa dan PDA yang dibuat terlebih dahulu (disain sendiri) menggunakan teori yang telah dipelajari.

1. JAWAB PERSOALAN

Program mula-mula akan mengecek terlebih dahulu sintaks input yang diberikan user dngan implementasi CFG yang sesuai pada bagian 4. Jika input user tidak memenuhi CFG maka akan keluar pesan “SYNTAX ERROR”. Jika input user memenuhi CFG barulah program akan masuk ke bagian perhitungan.

Pada bagian perhitungan, program mengimplementasikan PDA menggunakan stack yang menampung token (berisi karakter dan nilai). Proses PDA didefinisikan sebagai berikut :

* Jika karakter yang diperiksa adalah angka, maka program akan ‘mengambil’ 1 blok angka tersebut dan menyimpannya ke dalam stack. Misalkan input user adalah “123.45+54.321”, maka proses *push* pertama akan memasukkan nilai ‘123.45’ ke dalam stack.
* Jika karakter yang diperiksa adalah ‘+’, maka program akan mengabaikannya.
* Jika karakter yang diperiksa adalah ‘-‘, maka angka berikutnya yang di *push* akan dikali dengan -1.
* Jika karakter yang diperiksa adalah ‘\*’ atau ‘/’, maka program akan mem-*pop­­* angka terakhir dari stack, kemudian mengalikan atau membaginya dengan angka berikutnya, kemudian hasilnya di-*push* ke dalam stack.
* Jika karakter yang diperiksa adalah ‘(‘, maka program akan memasukkan tanda ‘(‘ ke dalam stack lalu melakukan rekursi seluruh proses ini.
* Jika karakter yang diperiksa adalah ‘)’, maka program akan mem-*pop* seluruh isi stack dan menjumlahkannya hingga menemukan tanda ‘(‘, kemudian hasilnya di-*push* ke dalam stack. Jika sebelum tanda ‘(‘ terdapat operator ‘-‘, ‘\*’, atau ‘/’, maka angka hasil penjumlahan tersebut akan diperlakukan sebagai ‘angka berikutnya’ sesuai penjelasan di atas. Misalkan input user adalah “5\*(1+2+3)”, maka operator \* akan mengalikan 5 dengan hasil dari (1+2+3).

1. DAFTAR ASUMSI

* Program akan menganggap “.5” salah, sintaks yang diperbolehkan adalah “0.5”.
* Penulisan angka negatif dengan “-1” diperbolehkan di awal perhitungan atau setelah tanda ‘(‘, tetapi tidak boleh diletakkan setelah operator lain seperti “1\*-1”, sintaks yang diperbolehkan adalah “1\*(-1)”.
* Hasil yang ditampilkan dibulatkan 2 angka di belakang koma jika 0.01 <= hasil < 1,000,000,000.
* Terdapat kesalahan konversi pada presisi yang kecil karena masalah representasi data dalam komputer.
* Dalam memasukkan input tidak boleh ada spasi.

1. CONTEXT FREE GRAMMAR (CFG)

Context Free Grammar yang digunakan dalam program ini adalah sebagai berikut :

1. SOURCE CODE
   1. kalkulator.c

#include <stdio.h>

#include <math.h>

#include "mesinkata.h"

#include "stacklist.h"

#include "check.h"

#define Karakter(i) input.TabKata[(i)]

int i;

Kata input;

Stack S;

double TakeAngka() {

double angka = 0;

int j;

do {

angka = angka\*10 + (double) ((int) Karakter(i) - (int) '0');

i++;

} while ((i <= input.Length) && ((int) Karakter(i) >= (int) '0') && ((int) Karakter(i) <= (int) '9'));

if (Karakter(i) == '.') {

i++;

j = 0;

while ((i <= input.Length) && ((int) Karakter(i) >= (int) '0') && ((int) Karakter(i) <= (int) '9')) {

j++;

angka = angka + (double) ((int) Karakter(i) - (int) '0') / pow(10,j);

i++;

}

}

return angka;

}

double proses() {

double angka,sum,temp;

boolean minus,kali,bagi;

infotype X;

minus = false;

kali = false;

bagi = false;

while ((i <= input.Length) && (Karakter(i) != ')')) {

if ((Karakter(i) >= '0') && (Karakter(i) <= '9')) {

angka = TakeAngka();

if (minus) {

angka \*= -1;

minus = false;

}

else if (kali) {

Pop(&S,&X);

angka \*= X.val;

kali = false;

}

else if (bagi) {

Pop(&S,&X);

angka = X.val / angka;

bagi = false;

}

X.op = 'b';

X.val = angka;

Push(&S,X);

}

else if (Karakter(i) == '+') {

i++;

}

else if (Karakter(i) == '-') {

i++;

minus = true;

}

else if (Karakter(i) == '\*') {

i++;

kali = true;

}

else if (Karakter(i) == '/') {

i++;

bagi = true;

}

else if (Karakter(i) == '(') {

X.op = '(';

X.val = 0;

Push(&S,X);

i++;

angka = proses();

if (minus) {

angka \*= -1;

minus = false;

}

else if (kali) {

Pop(&S,&X);

angka \*= X.val;

kali = false;

}

else if (bagi) {

Pop(&S,&X);

angka = X.val / angka;

bagi = false;

}

X.op = 'b';

X.val = angka;

Push(&S,X);

}

if (Karakter(i) == ')') {

angka = 0;

Pop(&S,&X);

while (X.op != '(') {

angka += X.val;

Pop(&S,&X);

}

}

}

i++;

return angka;

}

int main() {

float sum;

infotype X;

i = 1;

CreateEmpty(&S);

input = InputKata();

if (checkSemua(input)){

input.Length++;

input.TabKata[input.Length] = ')';

X.op = '(';

X.val = 0;

Push(&S,X);

sum = proses();

if (((sum >= 0.01) && (sum < 1000000000)) || ((sum <= -0.01) && (sum > -1000000000))) {

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

}

else {

printf("%.10e\n",sum);

}

}

else {

printf("SYNTAX ERROR\n");

}

printf("Press ENTER to continue");

while (getchar() != '\n');

return 0;

}

* 1. check.c

#include <stdio.h>

#include <stdlib.h>

#include "mesinkata.h"

#include "check.h"

boolean checkKurung(Kata kata){

char cc;

int ci=0,n=1;

boolean valid=true;

while (n<=kata.Length && valid){

if ( kata.TabKata[n] == '(' || kata.TabKata[n] == ')'){

if (kata.TabKata[n] == '('){

ci++;

}else{

ci--;

}

if (ci<0){

valid=false;

}

}

n++;

}

if (ci==0){

return valid;

}else{

return false;

}

}

boolean isOperator(char cc){

return (cc=='+' || cc=='-' || cc=='\*' || cc=='/');

}

boolean checkOperator(Kata kata){

char cc;

int ci=0,n=1;

boolean valid=true;

while (n<=kata.Length && valid){

if (isOperator(kata.TabKata[n])){

if (kata.TabKata[n]=='-'){

if (n == 1) {

if (!(kata.TabKata[n+1]=='(' || (kata.TabKata[n+1]>='0' && kata.TabKata[n+1] <= '9'))) {

valid = false;

}

}

else {

if (!(kata.TabKata[n-1]=='(' || kata.TabKata[n-1]==')' || (kata.TabKata[n-1]>='0' && kata.TabKata[n-1] <= '9'))) {

valid = false;

}

if (!(kata.TabKata[n+1]=='(' || (kata.TabKata[n+1]>='0' && kata.TabKata[n+1] <= '9'))) {

valid = false;

}

}

}

else {

if (n == 1) {

valid = false;

}

else {

if (!(kata.TabKata[n-1]==')' || (kata.TabKata[n-1]>='0' && kata.TabKata[n-1] <= '9'))) {

valid = false;

}

if (!(kata.TabKata[n+1]=='(' || (kata.TabKata[n+1]>='0' && kata.TabKata[n+1] <= '9'))) {

valid = false;

}

}

}

}

else if (kata.TabKata[n] == '(') {

if (n == 1) {

if (!(kata.TabKata[n+1]=='(' || kata.TabKata[n+1]=='-' || (kata.TabKata[n+1]>='0' && kata.TabKata[n+1] <= '9'))) {

valid = false;

}

}

else {

if (!(kata.TabKata[n-1]=='(' || isOperator(kata.TabKata[n-1]))) {

valid = false;

}

if (!(kata.TabKata[n+1]=='(' || kata.TabKata[n+1]=='-' || (kata.TabKata[n+1]>='0' && kata.TabKata[n+1] <= '9'))) {

valid = false;

}

}

}

else if (kata.TabKata[n] == ')') {

if (n == 1) {

valid = false;

}

else if (n == kata.Length) {

if (!(kata.TabKata[n-1]==')' || (kata.TabKata[n-1]>='0' && kata.TabKata[n-1] <= '9'))) {

valid = false;

}

}

else {

if (!(kata.TabKata[n-1]==')' || (kata.TabKata[n-1]>='0' && kata.TabKata[n-1] <= '9'))) {

valid = false;

}

if (!(kata.TabKata[n+1]==')' || isOperator(kata.TabKata[n+1]))) {

valid = false;

}

}

}

n++;

}

return valid;

}

boolean checkAngka(Kata kata){

char cc;

int ci=0,n=1;

boolean valid=true;

while (n<=kata.Length && valid){

if ((kata.TabKata[n]>= '0') && (kata.TabKata[n]<= '9')){

while ((kata.TabKata[n]>= '0') && (kata.TabKata[n]<= '9')) {

n++;

}

if (kata.TabKata[n] == '.') {

n++;

if ((kata.TabKata[n]>= '0') && (kata.TabKata[n]<= '9')) {

while ((kata.TabKata[n]>= '0') && (kata.TabKata[n]<= '9')) {

n++;

}

if (kata.TabKata[n] == '.') {

valid = false;

}

}

else {

valid = false;

}

}

}

else if (kata.TabKata[n] == '.') {

valid = false;

}

n++;

}

return valid;

}

boolean checkSemua(Kata kata) {

return (checkKurung(kata) && checkOperator(kata) && checkAngka(kata));

}

* 1. check.h

#ifndef \_CHECK

#define \_CHECK

#include "boolean.h"

boolean checkKurung(Kata kata);

boolean isOperator(char cc);

boolean checkOperator(Kata kata);

boolean checkAngka(Kata kata);

boolean checkSemua(Kata kata);

#endif

* 1. mesinkata.c

/\* File: mesinkata.c \*/

#include "mesinkata.h"

#include "mesinkar.h"

#include <stdio.h>

Kata InputKata()

// Fugsi untuk menerima input dari pengguna

{

Kata input;

char c;

int i = 0;

do {

scanf("%c",&c);

if (c != '\n') {

i++;

input.TabKata[i] = c;

}

} while (c != '\n');

input.Length = i;

return input;

}

* 1. mesinkata.h

/\* File: mesinkata.h \*/

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

#ifndef \_\_MESINKATA\_H\_\_

#define \_\_MESINKATA\_H\_\_

#include "boolean.h"

#define NMax 50

typedef struct {

char TabKata[NMax+1]; /\* container penyimpan kata \*/

int Length;

} Kata;

Kata InputKata();

// Fugsi untuk menerima input dari pengguna

#endif

* 1. boolean.h

/\* Definisi type boolean \*/

#ifndef \_BOOLEAN\_h

#define \_BOOLEAN\_h

#define boolean unsigned char

#define true 1

#define false 0

#endif

* 1. stacklist.c

#include <stdio.h>

#include <stdlib.h>

#include "boolean.h"

#include "stacklist.h"

/\* Prototype manajemen memori \*/

void Alokasi (address \*P, infotype X)

/\* I.S. Sembarang \*/

/\* F.S. Alamat P dialokasi, jika berhasil maka Info(P)=X dan

Next(P)=Nil \*/

/\* P=Nil jika alokasi gagal \*/

{

\*P = (address) malloc(sizeof(ElmtStack));

if ((\*P) != Nil) { //Alokasi berhasil

Info(\*P) = X;

Next(\*P) = Nil;

}

}

void Dealokasi (address P)

/\* I.S. P adalah hasil alokasi, P != Nil \*/

/\* F.S. Alamat P didealokasi, dikembalikan ke sistem \*/

{

free(P);

}

/\* \*\*\*\*\*\*\*\*\* PROTOTYPE REPRESENTASI LOJIK STACK \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

boolean IsEmpty (Stack S)

/\* Mengirim true jika Stack kosong: TOP(S) = Nil \*/

{

return Top(S) == Nil;

}

void CreateEmpty (Stack \* S)

/\* I.S. sembarang \*/

/\* F.S. Membuat sebuah stack S yang kosong \*/

{

Top(\*S) = Nil;

}

void Push (Stack \* S, infotype X)

/\* Menambahkan X sebagai elemen Stack S \*/

/\* I.S. S mungkin kosong, X terdefinisi \*/

/\* F.S. X menjadi TOP yang baru jika alokasi X berhasil, \*/

/\* jika tidak, S tetap \*/

/\* Pada dasarnya adalah operasi Insert First pada list linier \*/

{

address P;

Alokasi (&P,X);

if (P!=Nil) {

if (IsEmpty(\*S)) {

Top(\*S) = P;

}

else {

Next(P) = Top(\*S);

Top(\*S) = P;

}

}

}

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, \*/

/\* elemen TOP yang lama didealokasi \*/

/\* Pada dasarnya adalah operasi Delete First pada list linier \*/

{

\*X = InfoTop(\*S);

Top(\*S) = Next(Top(\*S));

}

* 1. stacklist.h

/\* File : stacklist.h \*/

#ifndef \_STACKLIST\_H

#define \_STACKLIST\_H

#include "boolean.h"

#include <stdlib.h>

/\* Konstanta \*/

#define Nil NULL

/\* Deklarasi infotype \*/

typedef struct {

float val;

char op;

} infotype;

/\* Stack dengan representasi berkait dengan pointer \*/

typedef struct tElmtStack \* address;

typedef struct tElmtStack {

infotype Info;

address Next;

} ElmtStack;

/\* Type stack dengan ciri TOP : \*/

typedef struct {

address TOP; /\* alamat TOP: elemen puncak \*/

} Stack;

/\* Selektor \*/

#define Top(S) (S).TOP

#define InfoTop(S) (S).TOP->Info

#define ValTop(S) (S).TOP->Info.val

#define OpTop(S) (S).TOP->Info.op

#define Next(P) (P)->Next

#define Info(P) (P)->Info

/\* Prototype manajemen memori \*/

void Alokasi (address \*P, infotype X);

/\* I.S. Sembarang \*/

/\* F.S. Alamat P dialokasi, jika berhasil maka Info(P)=X dan Next(P)=Nil \*/

/\* P=Nil jika alokasi gagal \*/

void Dealokasi (address P);

/\* I.S. P adalah hasil alokasi, P != Nil \*/

/\* F.S. Alamat P didealokasi, dikembalikan ke sistem \*/

/\* \*\*\*\*\*\*\*\*\* PROTOTYPE REPRESENTASI LOJIK STACK \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

boolean IsEmpty (Stack S);

/\* Mengirim true jika Stack kosong: TOP(S) = Nil \*/

void CreateEmpty (Stack \* S);

/\* I.S. sembarang \*/

/\* F.S. Membuat sebuah stack S yang kosong \*/

void Push (Stack \* S, infotype X);

/\* Menambahkan X sebagai elemen Stack S \*/

/\* I.S. S mungkin kosong, X terdefinisi \*/

/\* F.S. X menjadi TOP yang baru jika alokasi X berhasil, \*/

/\* jika tidak, S tetap \*/

/\* Pada dasarnya adalah operasi Insert First pada list linier \*/

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, \*/

/\* elemen TOP yang lama didealokasi \*/

/\* Pada dasarnya adalah operasi Delete First pada list linier \*/

#endif

1. CONTOH MASUKAN DAN KELUARAN





