Kantanat Siripipatworakun (5931005521)

Retro Basics Project Report

Description

1. Scanner : Using fstream to open and read the data from the file, reading word by word. Each word(token) are stored as *string* in vector *tok*.

2. Parser: Use algorithm based on PowerPoint Parsing slide 15 with some modification. The normal flow is as follows:

> Push a start symbol into *stk* (a provided stack)
> Start counter i at 0.
> While *stk* is not empty and iteration in *tok* is not finished :
>
> Pop *stk*
>
> Check validity of top of *stk* and tok[i] by:
>
> 1. Check if top of *stk* can be parsed anyway into *tok[i]* using parsing table.
> 2. If it cannot be parsed, the program rejects and ends immediately.
> 3. If top of *stk* is non-terminal and the parsing table entry corresponding to the valid parsing is A -> $X_1X_2X_3 \dots X_n$. Push $X_nX_{n-1}X_{n-2} \dots X_3X_2X_1$ into *stk*.
> 4. If top of *stk* is terminal, generate an output B-code corresponding to the current token and increment counter by 1 (point to the next token).
>
> If *stk* is not empty or some tokens are not iterated on, the program rejects.
> Otherwise, the parsing is finished successfully.

Rewritten Grammar

```
pgm := line pgm | EOF
line := line_num stmt
stmt := asgmnt | if | print | goto | stop
asgmnt := id = exp
exp := term exp'
exp' := + term | - term | λ
term := id | const
if := IF cond line_num
cond := term cond'
cond' := < term | = term
print := PRINT id
goto := GOTO line_num
stop := STOP
```

Kantanat Siripipatworakun (5931005521)

First set & Follow set

Non-terminal	First()	Follow()
pgm	line_num, EOF	$
line	line_num	line_num, EOF
stmt	id, IF, PRINT, GOTO, STOP	line_num, EOF
asgmnt	id	line_num, EOF
exp	id, const	line_num, EOF
exp’	+, -, λ	line_num, EOF
term	id, const	+, -, line_num, EOF
if	IF	line_num, EOF
cond	id, const	line_num
cond’	<, =	line_num
print	PRINT	line_num, EOF
goto	GOTO	line_num, EOF
stop	STOP	line_num, EOF

Split Grammar

1. pgm := line pgm
2. pgm := EOF
3. line := line_num stmt
4. stmt := asgmnt
5. stmt := if
6. stmt := print
7. stmt := goto
8. stmt := stop
9. asgmnt := id = exp
10. exp := term exp’
11. exp’ := + term
12. exp’ := - term
13. exp’ := λ
14. term := id
15. term := const
16. if := IF cond line_num
17. cond := term cond’
18. cond’ := < term
19. cond’ := = term
20. print := PRINT id
21. goto := GOTO line_num
22. stop := STOP

Kantanat Siripipatworakun (5931005521)

Parsing Table

	line_num	id	IF	PRINT	GOTO	STOP	+	-	const	<	=	EOF	$
pgm	1											2	
line	3												
stmt		4	5	6	7	8							
asgmnt		9											
exp		10							10				
exp’	13						11	12				13	
term		14							15				
if			16										
cond									17				
cond’										18	19		
print				20									
goto					21								
stop						22							

Code : Use C++

As the code is very long (almost 500 lines), here below is the link to code and attachments(such as file reports, .exe, .c and inputs) :

```
//Retro Basic
//Scanner and Parser
//Kantanat Siripipatworakun (ID:
5931005521)

#include<stdio.h>
#include<string>
#include<stack>
#include<fstream>
#include<vector>
#include<fstream>
#include<iostream>

#define s_pgm 1
#define s_line 2
#define s_stmt 3
#define s_asgmnt 4
#define s_exp 5
#define s_expp 6
#define s_term 7
#define s_if 8
#define s_cond 9
#define s_condp 10
#define s_print 11
#define s_goto 12
#define s_stop 13
#define s_line_num 14
#define s_id 15
#define s_IF 16
#define s_PRINT 17
#define s_GOTO 18
#define s_STOP 19
#define s_plus 20
#define s_minus 21
#define s_const 22
#define s_less 23
#define s_equal 24
#define s_goto_num 25
#define s_EOF 26
using namespace std;
stack <int> stk;
vector <string> tok;
int i=0;

void pushstk(int k){
    switch(k){
    case 1:
        stk.push(s_pgm);
        stk.push(s_line);
        break;
    case 3:
        stk.push(s_stmt);
        stk.push(s_line_num);
        break;
```

```
case 4:
    stk.push(s_asgmnt);
    break;
case 5:
    stk.push(s_if);
    break;
case 6:
    stk.push(s_print);
    break;
case 7:
    stk.push(s_goto);
    break;
case 8:
    stk.push(s_stop);
    break;
case 9:
    stk.push(s_exp);
    stk.push(s_equal);
    stk.push(s_id);
    break;
case 10:
    stk.push(s_expp);
    stk.push(s_term);
    break;
case 11:
    stk.push(s_term);
    stk.push(s_plus);
    break;
case 12:
    stk.push(s_term);
    stk.push(s_minus);
    break;
case 14:
    stk.push(s_id);
    break;
case 15:
    stk.push(s_const);
    break;
case 16:
    stk.push(s_goto_num);
    stk.push(s_cond);
    stk.push(s_IF);
    break;
case 17:
    stk.push(s_condp);
    stk.push(s_term);
    break;
case 18:
    stk.push(s_term);
    stk.push(s_less);
    break;
case 19:
    stk.push(s_term);
    stk.push(s_equal);
    break;
case 20:
    stk.push(s_id);
        stk.push(s_PRINT);
        break;
    case 21:
        stk.push(s_line_num);
        stk.push(s_GOTO);
        break;
    case 22:
        stk.push(s_STOP);
        break; }
}

int stoiAble(string s){
    int len = s.length();
    for(int i=0; i<len; i++){
        if(s[i] < 48 || s[i] > 57)
            return 0; }
    return 1; }

int isLineNum(string s){
    if(!stoiAble(s))
        return 0;
    int i = stoi(s);
    if(i>=1 && i<=1000)
        return 1;
    return 0; }

int isConst(string s){
    if(!stoiAble(s))
        return 0;
    int i = stoi(s);
    if(i>=0 && i<=100)
        return 1;
    return 0; }

int isId(string s){
    if(s.length() != 1)
        return 0;
    if(s[0]>='A' && s[0]<='Z')
        return 1;
    return 0; }

int d_pgm(){
    if(i == tok.size())
stk.push(s_EOF);
    else if(isLineNum(tok[i]))
        pushstk(1);
    else return 0;
    return 1;}

int d_line(){
    if(isLineNum(tok[i]))
        pushstk(3);
    else
        return 0;
    return 1; }

int d_stmt(){
```

```
    if(isId(tok[i]))
        pushstk(4);
    else if(tok[i]=="IF")
        pushstk(5);
    else if(tok[i]=="PRINT")
        pushstk(6);
    else if(tok[i]=="GOTO")
        pushstk(7);
    else if(tok[i]=="STOP")
        pushstk(8);
    else
        return 0;
    return 1; }

int d_asgmnt(){
    if(isId(tok[i]))
        pushstk(9);
    else
        return 0;
    return 1; }

int d_exp(){

if(isId(tok[i])||isConst(tok[i]))
        pushstk(10);
    else if (i==tok.size())
stk.push(s_EOF);
    else
        return 0;
    return 1; }

int d_expp(){
    if(tok[i]=="+")
        pushstk(11);
    else if(tok[i]=="-")
        pushstk(12);
    return 1; }

int d_term(){
    if(isId(tok[i]))
        pushstk(14);
    else if(isConst(tok[i]))
        pushstk(15);
    else
        return 0;
    return 1; }

int d_if(){
    if(tok[i]=="IF")
        pushstk(16);
    else
        return 0;
    return 1; }

int d_cond(){

if(isId(tok[i])||isConst(tok[i]))
        pushstk(17);
    else
        return 0;
    return 1; }

int d_condp(){
    if(tok[i]=="<")
        pushstk(18);
    else if(tok[i]=="=")
        pushstk(19);
    else
        return 0;
    return 1; }

int d_print(){
    if(tok[i]=="PRINT")
        pushstk(20);
    else
        return 0;
    return 1; }

int d_goto(){
    if(tok[i]=="GOTO")
        pushstk(21);
    else
        return 0;
    return 1; }

int d_stop(){
    if(tok[i]=="STOP")
        pushstk(22);
    else
        return 0;
    return 1; }

int d_line_num(){
    if(isLineNum(tok[i]))
        printf("\n10 %d
",stoi(tok[i]));
    else
        return 0;
    return 1; }

int d_id(){
    if(isId(tok[i]))
        printf("11 %d
",tok[i][0]+1-'A');
    else
        return 0;
    return 1;}

int d_IF(){
    if(tok[i]=="IF")
        printf("13 0 ");
    else
        return 0;
    return 1; }
```

```
int d_PRINT(){
    if(tok[i]=="PRINT")
        printf("15 0 ");
    else
        return 0;
    return 1; }

int d_GOTO(){
    if(tok[i]=="GOTO" &&
isLineNum(tok[i+1])){
        stk.pop();
        printf("14 %d
",stoi(tok[i+1]));
        i++; }
    else
        return 0;
    return 1; }

int d_STOP(){
    if(tok[i]=="STOP")
        printf("16 0 ");
    else
        return 0;
    return 1; }

int d_plus(){
    if(tok[i]=="+")
        printf("17 1 ");
    else
        return 0;
    return 1; }

int d_minus(){
    if(tok[i]=="-")
        printf("17 2 ");
    else
        return 0;
    return 1; }

int d_const(){
    if(isConst(tok[i]))
        printf("12 %d
",stoi(tok[i]));
    else
        return 0;
    return 1; }

int d_less(){
    if(tok[i]=="<")
        printf("17 3 ");
    else
        return 0;
    return 1; }

int d_equal(){
    if(tok[i]=="=")
        printf("17 4 ");
    else
        return 0;
    return 1; }

int d_goto_num(){
    if(isLineNum(tok[i]))
        printf("14 %d
",stoi(tok[i]));
    else
        return 0;
    return 1; }

int d_EOF(){
    if(i==tok.size()){
        printf("\n0");
        return 1; }
    return 0; }

void reject(){
    printf("\nPARSING FAILED:
incorrect syntax");
}
void accept(){
    printf("\nPARSING SUCCESSFUL");
}

int main(int argc, char *argv[]){
    fstream file;
    string fname = argv[1];
    string str;
    file.open(fname.c_str());
    while(file >> str){
        tok.push_back(str);
    }
    file.close();

    stk.push(s_pgm);
    int ret = 1;
    while(!stk.empty()&&(i <=
tok.size())){
        int top = stk.top();
        stk.pop();
        switch(top){
        case(s_pgm):
            ret = d_pgm();
            break;
        case(s_line):
            ret = d_line();
            break;
        case(s_stmt):
            ret = d_stmt();
            break;
        case(s_asgmnt):
            ret = d_asgmnt();
            break;
        case(s_exp):
```

```
            ret = d_exp();
            break;
        case(s_expp):
            ret = d_expp();
            break;
        case(s_term):
            ret = d_term();
            break;
        case(s_if):
            ret = d_if();
            break;
        case(s_cond):
            ret = d_cond();
            break;
        case(s_condp):
            ret = d_condp();
            break;
        case(s_print):
            ret = d_print();
            break;
        case(s_goto):
            ret = d_goto();
            break;
        case(s_stop):
            ret = d_stop();
            break;
        case(s_line_num):
            ret = d_line_num();
            i++;
            break;
        case(s_id):
            ret = d_id();
            i++;
            break;
        case(s_IF):
            ret = d_IF();
            i++;
            break;
        case(s_PRINT):
            ret = d_PRINT();
            i++;
            break;
        case(s_GOTO):
            ret = d_GOTO();
            i++;
            break;
        case(s_STOP):
            ret = d_STOP();
            i++;
            break;
        case(s_plus):
            ret = d_plus();
            i++;
            break;
        case(s_minus):
            ret = d_minus();
            i++;
            break;
        case(s_const):
            ret = d_const();
            i++;
            break;
        case(s_less):
            ret = d_less();
            i++;
            break;
        case(s_equal):
            ret = d_equal();
            i++;
            break;
        case(s_goto_num):
            ret = d_goto_num();
            i++;
            break;
        case(s_EOF):
            ret = d_EOF();
            break; }
        if(ret == 0){
            reject();
            return 0; }
    }
    if(!stk.empty() ||
i!=tok.size()){
        reject(); }
    else accept();
    return 0;
}
```