Course: Compiler and Programming Languages

Course Code: CPSC-323-0 (13759)

Term: Fall 2023

Compiler Design Final Project

Group members:

Diego Vela

Ruben Garcia

Dan-Albert Solis

Method Used: Predictive Parsing Table

Language Used: C++

Compiler Design Final Project

1) Original Program

Text file: “finalv1.txt”

```
program f2023;
(* This program computes and prints the value
of an expression *)
var
   (* declare variables *)
   a1 , b2a , c, ba : integer ;
begin
     a1 = 3 ;
     b2a = 4 ;
     c = 5 ;
   write ( c ); (* display c *)

      (* compute the value of the expression *)
      ba = a1 * ( b2a + 2 * c) ;
      write ( “value=”, ba ) ; (* print the value of ba*)
   end.
```

Text file: “finalf23.txt”

```
program f2023;
var
a1 , b2a , c, ba : integer ;
begin
a1 = 3 ;
b2a = 4 ;
c = 5 ;
write ( c );
ba = a1 * ( b2a + 2 * c) ;
write ( “value=”, ba ) ;
end.
```

2) Original grammar

<prog>	➜ **program** <identifier>; **var** <dec-list> **begin** <stat-list> **end.**
<identifier>	➜ <letter>{<letter>|<digit>}
<dec-list>	➜ <dec> : <type> ;
<dec>	➜ <identifier>,<dec>| < identifier >
<type>	➜ **integer**
<stat-list>	➜ <stat> | <stat> <stat-list>
<stat>	➜ <write> | <assign>
<write>	➜ **write** (<str> < identifier >);
<str>	➜ ”value=”, | λ
<assign>	➜ <identifier> = <expr>;
<expr>	➜ <expr> + <term> | <expr> - <term> | <term>
<term>	➜ <term> * <factor> | <term> / <factor>| <factor>
<factor>	➜ <identifier> | <number> | (<expr>)
<number>	➜ <sign><digit>{ <digit> }
<sign>	➜ + | - | λ
<digit>	➜ 0|1|2|...|9v
<letter>	➜ a|b|c|d|w|f

3) Original Grammar in BNF Form

State New Name	State	BNF Grammar
P	<prog>	➜ program <identifier>; var <dec-list> begin <stat-list> end.
I	<identifier>	➜ <letter> <IB>
IB	<IB>	➜ <letter> <IB>
IB	<IB>	➜ <digit> <IB>
IB	<IB>	➜ λ
DL	<dec-list>	➜ <dec> : <type> ;
DC	<dec>	➜ <identifier>, <dec>
DC	<dec>	➜ < identifier >
TP	<type>	➜ integer
SL	<stat-list>	➜ <stat>
SL	<stat-list>	➜ <stat> <stat-list>
ST	<stat>	➜ <write>
ST	<stat>	➜ <assign>
W	<write>	➜ write (<str> < identifier >);

SR	<str>	➜ ”value=”
SR	<str>	➜ λ
A	<assign>	➜ < identifier > = <expr>;
E	<expr>	➜ <expr> + <term>
E	<expr>	➜ <expr> - <term>
E	<expr>	➜ <term>
T	<term>	➜ <term> * <factor>
T	<term>	➜ <term> / <factor>
T	<term>	➜ <factor>
F	<factor>	➜ < identifier >
F	<factor>	➜ <number>
F	<factor>	➜ <expr>
N	<number>	➜ <sign> <digit> <NB>
NB	<NB>	➜ <digit> <NB>
NB	<NB>	➜ λ
S	<sign>	➜ +
S	<sign>	➜ -
S	<sign>	➜ λ

D	<digit>	➜ 0
D	<digit>	➜ 1
D	<digit>	➜ 2
D	<digit>	➜ 3
D	<digit>	➜ 4
D	<digit>	➜ 5
D	<digit>	➜ 6
D	<digit>	➜ 7
D	<digit>	➜ 8
D	<digit>	➜ 9
L	<letter>	➜ a
L	<letter>	➜ b
L	<letter>	➜ c
L	<letter>	➜ d
L	<letter>	➜ w
L	<letter>	➜ f

4) Preparing BNF Grammar for Predictive Parsing Table

State	BNF Grammar
P	➜ program I ; var DL begin SL end.
I	➜ L IB
IB	➜ L IB
IB	➜ D IB
IB	➜ λ
DL	➜ DC : TP ;
DC	➜ I, DC
DC	➜ I
TP	➜ integer
SL	➜ ST
SL	➜ ST SL
ST	➜ W
ST	➜ A
W	➜ write (SR I);

SR	➜”value=”
SR	➜ λ
A	➜ I = E;
E	➜ E + T
E	➜ E – T
E	➜ T
T	➜ T * F
T	➜ T / F
T	➜ F
F	➜ I
F	➜ N
F	➜ E
N	➜ S D NB
NB	➜ D NB
NB	➜ λ
S	➜ +
S	➜ -
S	➜ λ

D	➜ 0
D	➜ 1
D	➜ 2
D	➜ 3
D	➜ 4
D	➜ 5
D	➜ 6
D	➜ 7
D	➜ 8
D	➜ 9
L	➜ a
L	➜ b
L	➜ c
L	➜ d
L	➜ w
L	➜ f

5) BNF Grammar Removing Left-Recursion

State	BNF Grammar
P	➜ program I PB
PB	➜ ; PC
PC	➜ var DL PD
PD	➜ begin SL PE
PE	➜ end.
I	➜ L IB
IB	➜ L IB
IB	➜ D IB
IB	➜ λ
DL	➜ DC DLB
DLB	➜ : TP DLC
DLC	➜ ;
DC	➜ I, DCB
DCB	➜ , DCC
DCB	➜ λ
TP	➜ integer

SL	➜ ST SLB
SLB	➜ SL
SLB	➜ λ
ST	➜ W
ST	➜ A
W	➜ write WB
WB	➜ (WC
WC	➜ SR WD
WD	➜ I WE
WE	➜) WF
WF	➜ ;
SR	➜”value=”
SR	➜ λ
A	➜ I AB
AB	➜ = E AC
AC	➜ ;
E	➜ T EB
EB	➜ + T EB

EB	➜ – T EB
EB	➜ λ
T	➜ F TB
TB	➜ * F TB
TB	➜ / F TB
TB	➜ λ
F	➜ (E)
F	➜ I
F	➜ N
N	➜ S D NB
NB	➜ D NB
NB	➜ λ
S	➜ +
S	➜ -
S	➜ λ
D	➜ 0
D	➜ 1
D	➜ 2

D	➜ 3
D	➜ 4
D	➜ 5
D	➜ 6
D	➜ 7
D	➜ 8
D	➜ 9
L	➜ a
L	➜ b
L	➜ c
L	➜ d
L	➜ w
L	➜ f

6) FIRST Table

State	FIRST
P	➜ program
PB	➜ ;
PC	➜ var
PD	➜ begin
PE	➜ end.
I	➜ a b c d w f
IB	➜ a b c d w f 0 1 2 3 4 5 6 7 8 9 λ
DL	➜ a b c d w f
DLB	➜ :
DLC	➜ ;
DC	➜ a b c d w f
DCB	➜ , λ
TP	➜ integer
SL	➜ write a b c d w f
SLB	➜ write a b c d w f λ
ST	➜ write a b c d w f

W	➜ write
WB	➜ (
WC	➜ “values=” λ
WD	➜ a b c d w f
WE	➜)
WF	➜ ;
SR	➜”value=”
SR	➜ λ
A	➜ a b c d w f
AB	➜ =
AC	➜ ;
E	➜ (a b c d w f + - 0 1 2 3 4 5 6 7 8 9
EB	➜ + - λ
T	➜ (a b c d w f + - 0 1 2 3 4 5 6 7 8 9
TB	➜ * / λ
F	➜ (a b c d w f + - 0 1 2 3 4 5 6 7 8 9
N	➜ + - 0 1 2 3 4 5 6 7 8 9
NB	➜ 0 1 2 3 4 5 6 7 8 9 λ

S	➜ + - λ
D	➜ 0 1 2 3 4 5 6 7 8 9
L	➜ a b c d w f

FOLLOW Table

State	FOLLOW
P	➜ $
PB	➜ $
PC	➜ $
PD	➜ $
PE	➜ $
I	➜ ; ,) = : * / + -
IB	➜ ; ,) = : * / + -
DL	➜ begin
DLB	➜ begin
DLC	➜ begin
DC	➜ :

DCB	➜ :
TP	➜ ;
SL	➜ end.
SLB	➜ end.
ST	➜ write a b c d w f end.
W	➜ write a b c d w f end.
WB	➜ write a b c d w f end.
WC	➜ write a b c d w f end.
WD	➜ write a b c d w f end.
WE	➜ write a b c d w f end.
WF	➜ write a b c d w f end.
SR	➜ write a b c d w f end.
SR	➜ write a b c d w f end.
A	➜ write a b c d w f end.
AB	➜ write a b c d w f end.
AC	➜ write a b c d w f end.
E	➜ ;)
EB	➜ ;)

T	→ + - ;)
TB	→ + - ;)
F	→ * / + - ;)
N	→ * / + - ;)
NB	→ * / + - ;)
S	→ 0 1 2 3 4 5 6 7 8 9
D	→ a b c d w f 0 1 2 3 4 5 6 7 8 9 ; ,) = : * / + -
L	→ a b c d w f 0 1 2 3 4 5 6 7 8 9 ; ,) = : * / + -

7) The Predictive Parsing Table chart

State	BNF Grammar		
P	→ program I PB	FIRST	program
PB	→ ; PC	FIRST	;
PC	→ var DL PD	FIRST	var
PD	→ begin SL PE	FIRST	begin
PE	→ end.	FIRST	end.
I	→ L IB	FIRST (L)	a b c d w f

IB	➜ L IB	FIRST (L)	a b c d w f
IB	➜ D IB	FIRST (D)	0 1 2 3 4 5 6 7 8 9
IB	➜ λ	FOLLOW (IB)	; ,) = : * / + -
DL	➜ DC DLB	FIRST (DC)	a b c d w f
DLB	➜ : TP DLC	FIRST	:
DLC	➜ ;	FIRST	;
DC	➜ I, DCB	FIRST (I)	a b c d w f
DCB	➜ , DC	FIRST	,
DCB	➜ λ	FOLLOW (DCB)	:
TP	➜ integer	FIRST	integer
SL	➜ ST SLB	FIRST (ST)	write a b c d w f
SLB	➜ SL	FIRST (SL)	write a b c d w f
SLB	➜ λ	FOLLOW (SLB)	end.
ST	➜ W	FIRST (W)	write
ST	➜ A	FIRST (A)	a b c d w f
W	➜ write WB	FIRST	write
WB	➜ (WC	FIRST	(
WC	➜ SR WD	FIRST (SR)	“value=”
WC	➜ WD	FOLLOW (WC)	a b c d w f

WD	➜ I WE	FIRST (I)	a b c d w f
WE	➜) WF	FIRST	)
WF	➜ ;	FIRST	;
SR	➜”value=”	FIRST	“value=”
SR	➜ λ	FOLLOW (SR)	end.
A	➜ I AB	FIRST (I)	a b c d w f
AB	➜ = E AC	FIRST	=
AC	➜ ;	FIRST	;
E	➜ T EB	FIRST (T)	(a b c d w f + - 0 1 2 3 4 5 6 7 8 9
EB	➜ + T EB	FIRST	+
EB	➜ – T EB	FIRST	-
EB	➜ λ	FOLLOW (EB)	;)
T	➜ F TB	FIRST (F)	(a b c d w f + - 0 1 2 3 4 5 6 7 8 9
TB	➜ * F TB	FIRST	*
TB	➜ / F TB	FIRST	/
TB	➜ λ	FOLLOW (TB)	+ - ;)
F	➜ (E)	FIRST	(
F	➜ I	FIRST (I)	a b c d w f

F	➔ N	FIRST (N)	+ - 0 1 2 3 4 5 6 7 8 9
N	➔ S D NB	FIRST (S D)	+ - 0 1 2 3 4 5 6 7 8 9
NB	➔ D NB	FIRST (D)	0 1 2 3 4 5 6 7 8 9
NB	➔ λ	FOLLOW (NB)	* / + - ;)
S	➔ +	FIRST	+
S	➔ -	FIRST	-
S	➔ λ	FOLLOW (S)	0 1 2 3 4 5 6 7 8 9
D	➔ 0	FIRST	0
D	➔ 1	FIRST	1
D	➔ 2	FIRST	2
D	➔ 3	FIRST	3
D	➔ 4	FIRST	4
D	➔ 5	FIRST	5
D	➔ 6	FIRST	6
D	➔ 7	FIRST	7
D	➔ 8	FIRST	8
D	➔ 9	FIRST	9
L	➔ a	FIRST	a
L	➔ b	FIRST	b

L	➔ c	FIRST	c
L	➔ d	FIRST	d
L	➔ w	FIRST	w
L	➔ f	FIRST	f

8) Part I Program

```
#include <iostream>
#include <string>
#include <algorithm>
#include <sstream>
#include <bits/stdc++.h>

using namespace std;

int main() {

    ifstream inputFile("finalv1.txt");
    ofstream outFile("finalf23.txt");
    string t, q;

    if (!inputFile.is_open() || !outFile.is_open()) {
        std::cerr << "Error opening files!\n";
        return EXIT_FAILURE;
    }


    std::string word;
    std::regex regexMultipleSpaces("\\s+");
    bool isComment = false;
    while(getline(inputFile, word)){    //Get entire line
instead so the output stays consistent

        //Delete Comments
        if (isComment) {
            size_t commEnd = word.find("*)");
            if (commEnd != std::string::npos) {
                word = word.substr(commEnd+2);
                isComment = false;
            } else {
                isComment = true;
                continue;
            }
        } else {
            size_t commStart = word.find("(*");
            if (commStart != std::string::npos) {
                string temp;
                // Keep only the content before the specific
character
```

```
                temp = word.substr(0, commStart);
                //Check if comment continues to another line
                size_t commEnd = word.find("*)");
                if (commEnd != std::string::npos) {
                    temp += word.substr(commEnd+2);
                } else {
                    isComment = true;
                    continue;
                }
                word = temp;
            }
        }

        //Check if line is empty
        if (word.empty()) {continue;}

        //Delete extra whitespaces
        std::string modWord = std::regex_replace(word,
regexMultipleSpaces, " ");

        //Delete preceding whitespaces
        size_t start = modWord.find_first_not_of(" \t");
        if (start != std::string::npos) {
            // Extract the substring starting from the first
non-space character
            outFile << modWord.substr(start) << std::endl; //
write modified line to temporary file
        }
    }
    /*if (isInsideComment(word)){
        //continue;
    //}

    //do processing of the tokens here
    //outFile << word << std::endl;
    //std::cout << word << std::endl;


    }*/

    return 0;
}
```

Given finalv1.txt

```
program f2023;
(* This program computes and prints the value
of an expression *)
var
   (* declare variables *)
   a1 , b2a , c, ba : integer ;
begin
     a1 = 3 ;
     b2a = 4 ;
     c = 5 ;
   write ( c ); (* display c *)

       (* compute the value of the expression *)
       ba = a1 * ( b2a + 2 * c) ;
       write ( “value=”, ba ) ; (* print the value of ba*)
   end.
```

Part I Program Sample Run

Console

```
"C:\Users\Dan\Documents\CPSC_323\Assignments\cpsc 323 final
proj\cmake-build-debug\cpsc_323_final_proj.exe"

Process finished with exit code 0
```

Output File

```
program f2023;
var
a1 , b2a , c, ba : integer ;
begin
a1 = 3 ;
b2a = 4 ;
c = 5 ;
write ( c );
ba = a1 * ( b2a + 2 * c) ;
write ( “value=”, ba ) ;
end.
```

Compiler Program

```
/*  Final Version: 21
    Programmers: Diego Vela, Ruben Garcia, Dan Solis
    Description: Simple Compiler Program.
*/
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
#include <map>
#include <unordered_map>
#include <stack>
#include <sstream>
#include <fstream>
#include <bits/stdc++.h>

using namespace std;
const int nun = 2147483647;

void createFile();
void createStack(vector<string> *myStack);
bool checkGrammar(vector<string> *myStack);
void compileMe(vector<string> *program);
int evaluate(vector<string> expression);

int main() {

    //finalf23.txt
    createFile();

    //Create a string array of file "finalf23"
    vector<string> program;
    createStack(&program);

    //Check the grammar
    if (program[0]!= "program") {
        std::cout << "Expected program\n";
        std::cout << "Failed to Compile..." << std::endl;
        return EXIT_SUCCESS;
    } else if (program.back() == ".") {
        program.pop_back();
        program.back() += ".";
    }


    if (checkGrammar(&program)) {
        reverse(program.begin(), program.end());
        std::cout << "Now Compiling...\n" << std::endl;
        compileMe(&program);
    } else {
        std::cout << "Failed to Compile..." << std::endl;
    }

    return EXIT_SUCCESS;
}

/* ===== START OF FUNCTIONS ===== */

//Helper function for evaluate
bool isOperator(const string &token) {
    return (token == "+" || token == "-" || token == "*" || token == "/");
}

//Helper function for evaluate
int performOperation(int operand1, int operand2, const string &op) {
    if (op == "+") {
        return operand1 + operand2;
    } else if (op == "-") {
        return operand1 - operand2;
    } else if (op == "*") {
        return operand1 * operand2;
    } else if (op == "/") {
        return operand1 / operand2;
    }
    return 0;
}

//Helper function for evaluate
int evaluateExpression(const vector<string> &expression) {
    stack<int> numbers;
    stack<string> ops;

    unordered_map<string, int> precedence;
    precedence["+"] = precedence["-"] = 1;
    precedence["*"] = precedence["/"] = 2;

    for (const string &token : expression) {
        if (isdigit(token[0])) {
            numbers.push(stoi(token));
        } else if (isOperator(token)) {
            while (!ops.empty() && precedence[ops.top()] >= precedence[token])
{
                int operand2 = numbers.top();
                numbers.pop();
                int operand1 = numbers.top();
                numbers.pop();
                string op = ops.top();
                ops.pop();
                numbers.push(performOperation(operand1, operand2, op));
            }
            ops.push(token);
        }
    }

    while (!ops.empty()) {
        int operand2 = numbers.top();
        numbers.pop();
        int operand1 = numbers.top();
        numbers.pop();
        string op = ops.top();
        ops.pop();
        numbers.push(performOperation(operand1, operand2, op));
    }

    return numbers.top();
}

//Evaluates an expression with respect to PEMDAS
int evaluate(vector<string> expression) {
    vector<string> group;
    int left = 0;
    int right = 0;
    int count = 0;
    int tempTotal = 0;

    while (expression.begin()+count != expression.end() ) {
        if (expression[count] == "(") {
            left++;
            expression.erase(expression.begin()+count);
            while (left != right && count < expression.size()) {
                if(expression[count] == "(") {
                    left++;
                    expression.erase(expression.begin()+count);
                } else if (expression[count] == ")") {
                    right++;
                    expression.erase(expression.begin()+count);
                } else {
                    group.push_back(expression[count]);
                    expression.erase(expression.begin()+count);
                }
            }
            expression.insert(expression.begin()+count,
to_string(evaluate(group)));
            left = 0;
            right = 0;
            group.clear();
        }
        count++;
    }
    return evaluateExpression(expression);
}

//Creates a usable file given a file name to open
void createFile() {
    ifstream inputFile("finalv1.txt");
    ofstream outFile("finalf23.txt");
    string t, q;

    if (!inputFile.is_open() || !outFile.is_open()) {
        cerr << "Error opening files!\n";
        exit(1);
    }


    std::string word;
    std::regex regexMultipleSpaces("\\s+");
    bool isComment = false;
    while(getline(inputFile, word)){    //Get entire line instead so the
output stays consistent

        //Delete Comments
        if (isComment) {
            size_t commEnd = word.find("*)");
            if (commEnd != std::string::npos) {
                word = word.substr(commEnd+2);
                isComment = false;
            } else {
                isComment = true;
                continue;
            }
        } else {
            size_t commStart = word.find("(*");
            if (commStart != std::string::npos) {
                string temp;
                // Keep only the content before the specific character
                temp = word.substr(0, commStart);
                //Check if comment continues to another line
                size_t commEnd = word.find("*)");
                if (commEnd != std::string::npos) {
                    temp += word.substr(commEnd+2);
                } else {
                    isComment = true;
                    continue;
                }
                word = temp;
            }
        }

        //Check if line is empty
        if (word.empty()) {continue;}

        //Delete extra whitespaces
        std::string modWord = std::regex_replace(word, regexMultipleSpaces, "
");
```

```
        //Delete preceding whitespaces
        size_t start = modWord.find_first_not_of(" \t");
        if (start != std::string::npos) {
            // Extract the substring starting from the first non-space
character
            outFile << modWord.substr(start) << std::endl; // write modified
line to temporary file
        }
    }
    inputFile.close();
    outFile.close();
}

//Parsing table helper
string parse(string stackVal, string readVal) {
     //Step 1: Create a 2D vector and populate it (let -1 = blank)
    vector<vector<string>> ppTable {
               //program       var         begin      end.   integer   write
"values"   +        -        *        /        =        (       )      ,
;      :          0
        /*P  */{"program I PB","p"        ,"p"       ,"p"   ,"p"      ,"p"
,"p"       ,"p"     ,"p"     ,"p"     ,"p"     ,"p"     ,"p"    ,"p"   ,"p"
,"p"   ,"p"       ,"p"
,"p","p","p","p","p","p","p","p","p","p","p","p","p","p","p"},
        /*PB */{"|"           ,"|"        ,"|"       ,"|"   ,"|"      ,"|"
,"|"       ,"|"     ,"|"     ,"|"     ,"|"     ,"|"     ,"|"    ,"|"   ,"|"
,"; PC","|"       ,"|"
,"|","|","|","|","|","|","|","|","|","|","|","|","|","|","|"},
        /*PC */{"v"           ,"var DL PD","v"       ,"v"   ,"v"      ,"v"
,"v"       ,"v"     ,"v"     ,"v"     ,"v"     ,"v"     ,"v"    ,"v"   ,"v"
,"v"   ,"v"       ,"v"
,"v","v","v","v","v","v","v","v","v","v","v","v","v","v","v" },
        /*PD */{"g"           ,"g"        ,"begin SL PE","g"   ,"g"      ,"g"
,"g"       ,"g"     ,"g"     ,"g"     ,"g"     ,"g"     ,"g"    ,"g"   ,"g"
,"g"   ,"g"       ,"g"
,"g","g","g","g","g","g","g","g","g","g","g","g","g","g","g" },
        /*PE */{"|"           ,"|"        ,"|"       ,"end.","|"      ,"|"
,"|"       ,"|"     ,"|"     ,"|"     ,"|"     ,"|"     ,"|"    ,"|"   ,"|"
,"|"   ,"|"       ,"|"
,"|","|","|","|","|","|","|","|","|","|","|","|","|","|","|" },
        /*I  */{"|"           ,"|"        ,"|"       ,"|"   ,"|"      ,"|"
,"|"       ,"|"     ,"|"     ,"|"     ,"|"     ,"|"     ,"|"    ,"|"   ,"|"
,"i"   ,"|"       ,"|"     ,"|","|","|","|","|","|","|","|","|","L IB","L
IB","L IB","L IB","L IB","L IB" },
        /*IB */{"|"           ,"z"        ,"|"       ,"|"   ,"|"      ,"|"
,"|"       ,"λ"     ,"λ"     ,"λ"     ,"λ"     ,"λ"     ,"|"    ,"λ"   ,"λ"
,"λ"   ,"λ"       ,"D IB"  ,"D IB","D IB","D IB","D IB","D IB","D IB","D
IB","D IB","D IB","L IB","L IB","L IB","L IB","L IB","L IB" },
        /*DL */{"|"           ,"|"        ,"|"       ,"|"   ,"|"      ,"|"
,"|"       ,"|"     ,"|"     ,"|"     ,"|"     ,"|"     ,"|"    ,"|"   ,"|"
,"|"   ,"|"       ,"|"     ,"|","|","|","|","|","|","|","|","|","DC DLB","DC
DLB","DC DLB","DC DLB","DC DLB","DC DLB" },
        /*DLB*/{"|"           ,"|"        ,"|"       ,"|"   ,"|"      ,"|"
,"|"       ,"|"     ,"|"     ,"|"     ,"|"     ,"|"     ,"|"    ,"|"   ,"|"
,"|"   ,": TP DLC","|"
,"|","|","|","|","|","|","|","|","|","|","|","|","|","|","|" },
        /*DLC*/{"|"           ,"|"        ,"z"       ,"|"   ,"|"      ,"|"
,"|"       ,"|"     ,"|"     ,"|"     ,"|"     ,"|"     ,"|"    ,"|"   ,"|"
,";"   ,"|"       ,"|"
,"|","|","|","|","|","|","|","|","|","|","|","|","|","|","|" },
        /*DC */{"|"           ,"|"        ,"|"       ,"|"   ,"|"      ,"|"
,"|"       ,"|"     ,"|"     ,"|"     ,"|"     ,"|"     ,"|"    ,"|"   ,"|"
,"|"   ,"|"       ,"|"     ,"|","|","|","|","|","|","|","|","|"," I DCB ","I
DCB","I DCB","I DCB ","I DCB","I DCB" },
        /*DCB*/{"|"           ,"|"        ,"|"       ,"|"   ,"|"      ,"|"
,"|"       ,"|"     ,"|"     ,"|"     ,"|"     ,"|"     ,"|"    ,"|"   ,",
DC","|"   ,"λ"      ,"|"
,"|","|","|","|","|","|","|","|","|","|","|","|","|","|","|" },
        /*TP */{"|"           ,"|"        ,"|"       ,"|"   ,"integer","|"
,"|"       ,"|"     ,"|"     ,"|"     ,"|"     ,"|"     ,"|"    ,"|"   ,"|"
,"|"   ,"|"       ,"|"
,"|","|","|","|","|","|","|","|","|","|","|","|","|","|","|" },
        /*SL */{"|"           ,"|"        ,"|"       ,"|"   ,"|"      ,"ST
SLB"  ,"|"       ,"|"     ,"|"     ,"|"     ,"|"     ,"|"     ,"|"    ,"|"
,"|"   ,"|"   ,"|"       ,"|"     ,"|","|","|","|","|","|","|","|","|"," ST
SLB "," ST SLB "," ST SLB "," ST SLB "," ST SLB "," ST SLB " },
        /*SLB*/{"e"           ,"|"        ,"|"       ,"λ"   ,"|"      ,"SL"
,"|"       ,"|"     ,"|"     ,"|"     ,"|"     ,"|"     ,"|"    ,"|"   ,"|"
,"|"   ,"|"       ,"|"     ,"|","|","|","|","|","|","|","|","|"," SL "," SL
"," SL "," SL "," SL "," SL " },
        /*ST */{"|"           ,"|"        ,"|"       ,"|"   ,"|"      ,"W"
,"|"       ,"|"     ,"|"     ,"|"     ,"|"     ,"|"     ,"|"    ,"|"   ,"|"
,"|"   ,"|"       ,"|"     ,"|","|","|","|","|","|","|","|","|"," A "," A ","
A "," A "," A "," A " },
        /*W  */{"|"           ,"|"        ,"|"       ,"|"   ,"|"      ,"write
WB","|"       ,"|"     ,"|"     ,"|"     ,"|"     ,"|"     ,"|"    ,"|"   ,"|"
,"|"   ,"|"       ,"|"
,"|","|","|","|","|","|","|","|","|","|","|","|","|","|","|" },
        /*WB */{"|"           ,"|"        ,"|"       ,"|"   ,"|"      ,"|"
,"|"       ,"|"     ,"|"     ,"|"     ,"|"     ,"|"     ,"( WC" ,"|"   ,"|"
,"|"   ,"|"       ,"|"
,"|","|","|","|","|","|","|","|","|","|","|","|","|","|","|" },
        /*WC */{"|"           ,"|"        ,"|"       ,"|"   ,"|"      ,"|"
,"SR WD"   ,"|"     ,"|"     ,"|"     ,"|"     ,"|"     ,"|"    ,"|"   ,"|"
,"|"   ,"|"       ,"|"
,"|","|","|","|","|","|","|","|","|","WD","WD","WD","WD","WD","WD" },
        /*WD */{"|"           ,"|"        ,"|"       ,"|"   ,"|"      ,"|"
,"|"       ,"|"     ,"|"     ,"|"     ,"|"     ,"|"     ,"|"    ,"|"   ,"|"
,"|"   ,"|"       ,"|"     ,"|","|","|","|","|","|","|","|","|"," I WE "," I
WE "," I WE "," I WE "," I WE "," I WE " },
        /*WE */{"|"           ,"|"        ,"|"       ,"|"   ,"|"      ,"|"
,"|"       ,"|"     ,"|"     ,"|"     ,"|"     ,"|"     ,"|"    ,") WF","|"
,"|"   ,"|"       ,"|"
,"|","|","|","|","|","|","|","|","|","|","|","|","|","|","|" },
        /*WF */{"|"           ,"|"        ,"|"       ,"|"   ,"|"      ,"z"
,"|"       ,"|"     ,"|"     ,"|"     ,"|"     ,"|"     ,"|"    ,"|"   ,"|"
,";"   ,"|"        ,"|"
,"|","|","|","|","|","|","|","|","|","z","z","z","z","z","z" },
        /*SR */{"|"           ,"|"        ,"|"       ,"λ"   ,"|"      ,"|"
,"“value=” ","|"      ,"|"     ,"|"     ,"|"     ,"|"     ,"|"    ,"|"   ,"|"
,"|"   ,"|"        ,"|"
,"|","|","|","|","|","|","|","|","|","|","|","|","|","|","|" },
        /*A  */{"|"           ,"|"        ,"|"       ,"|"   ,"|"      ,"|"
,"|"       ,"|"     ,"|"     ,"|"     ,"|"     ,"|"     ,"|"    ,"|"   ,"|"
,"|"   ,"|"        ,"|"     ,"|","|","|","|","|","|","|","|","|"," I AB "," I
AB "," I AB "," I AB "," I AB "," I AB " },
        /*AB */{"|"           ,"|"        ,"|"       ,"|"   ,"|"      ,"|"
,"|"       ,"|"     ,"|"     ,"|"     ,"|"     ,"= E AC","|"    ,"y"   ,"|"
,"|"   ,"|"        ,"|"
,"|","|","|","|","|","|","|","|","|","|","|","|","|","|","|" },
        /*AC */{"|"           ,"|"        ,"|"       ,"|"   ,"|"      ,"|"
,"|"       ,"|"     ,"|"     ,"|"     ,"|"     ,"|"     ,"|"    ,"|"   ,"|"
,";"   ,"|"        ,"|"
,"|","|","|","|","|","|","|","|","|","|","|","|","|","|","|" },
        /*E  */{"|"           ,"|"        ,"|"       ,"|"   ,"|"      ,"|"
,"|"       ,"T EB"  ,"T EB"  ,"|"     ,"|"     ,"|"     ,"T EB" ,"|"   ,"|"
,"|"   ,"|"        ,"T EB"  ," T EB "," T EB "," T EB "," T EB "," T EB "," T
EB "," T EB "," T EB "," T EB "," T EB "," T EB "," T EB "," T EB "," T EB ","
T EB " },
        /*EB */{"|"           ,"|"        ,"|"       ,"|"   ,"|"      ,"|"
,"|"       ,"+ T EB","- T EB","|"     ,"|"     ,"|"     ,"|"    ,"λ"   ,"|"
,"λ"   ,"|"        ,"|"
,"|","|","|","|","|","|","|","|","|","|","|","|","|","|","|" },
        /*T  */{"|"           ,"|"        ,"|"       ,"|"   ,"|"      ,"|"
,"|"       ,"F TB"  ,"F TB"  ,"|"     ,"|"     ,"|"     ,"F TB" ,"|"   ,"|"
,"|"   ,"|"        ,"F TB"  ," F TB "," F TB "," F TB "," F TB "," F TB "," F
TB "," F TB "," F TB "," F TB "," F TB "," F TB "," F TB "," F TB "," F TB ","
F TB " },
        /*TB */{"|"           ,"|"        ,"|"       ,"|"   ,"|"      ,"|"
,"|"       ,"λ"     ,"λ"     ,"* F TB","/ F TB","|"     ,"|"    ,"λ"   ,"|"
,"λ"   ,"|"        ,"|"
,"|","|","|","|","|","|","|","|","|","z","z","z","z","z","z" },
        /*F  */{"|"           ,"|"        ,"|"       ,"|"   ,"|"      ,"|"
,"|"       ,"N"     ,"N"     ,"|"     ,"|"     ,"|"     ,"( E )","|"   ,"|"
,"|"   ,"|"        ,"D"
,"D","D","D","D","D","D","D","D","D","I","I","I","I","I","I" },
        /*N  */{"|"           ,"|"        ,"|"       ,"|"   ,"|"      ,"|"
,"|"       ,"S D NB","S D NB","|"     ,"|"     ,"|"     ,"|"    ,"|"   ,"|"
,"|"   ,"|"        ,"S D NB"," S D NB "," S D NB "," S D NB "," S D NB "," S D
NB "," S D NB "," S D NB "," S D NB "," S D NB ","|","|","|","|","|","|" },
        /*NB */{"|"           ,"|"        ,"|"       ,"|"   ,"|"      ,"|"
,"|"       ,"λ"     ,"λ"     ,"λ"     ,"λ"     ,"|"     ,"|"    ,"λ"   ,"|"
,"λ"   ,"|"        ," D NB "," D NB "," D NB "," D NB "," D NB "," D NB "," D
NB "," D NB "," D NB "," D NB ","|","|","|","|","|","|" },
        /*S  */{"|"           ,"|"        ,"|"       ,"|"   ,"|"      ,"|"
,"|"       ,"+"     ,"-"     ,"|"     ,"|"     ,"|"     ,"|"    ,"|"   ,"|"
```

```
,"|"   ,"|"       ," λ "," λ "," λ "," λ "," λ "," λ "," λ "," λ "," λ "," λ
","|","|","|","|","|","|" },
        /*D  */{"|"           ,"|"        ,"|"       ,"|"    ,"|"      ,"|"
,"|"       ,"|"     ,"|"     ,"|"     ,"|"     ,"|"     ,"|"    ,"|"   ,"|"
,"|"   ,"|"       ," 0 "," 1 "," 2 "," 3 "," 4 "," 5 "," 6 "," 7 "," 8 "," 9
","|","|","|","|","|","|" },
        /*L  */{"|"           ,"|"        ,"|"       ,"|"    ,"|"      ,"|"
,"|"       ,"|"     ,"|"     ,"|"     ,"|"     ,"|"     ,"|"    ,"|"   ,"|"
,"|"   ,"|"       ,"|","|","|","|","|","|","|","|","|","|"," a "," b "," c ","
d "," w "," f " }
    };

    //Step 2:Create a Map of corresponding values
    map<string,int> myMap;

    myMap["program"] = 0; myMap["var"] = 1; myMap["begin"] = 2; myMap["end."]
= 3;
    myMap["integer"] = 4; myMap["write"] = 5; myMap["“value=”"] = 6;
myMap["+"] = 7;
    myMap["-"] = 8; myMap["*"] = 9; myMap["/"] = 10; myMap["="] =11;
    myMap["("] = 12; myMap[")"] = 13; myMap[","] = 14; myMap[";"] = 15;
    myMap[":"] = 16; myMap["0"] = 17; myMap["1"] = 18;
    myMap["2"] = 19; myMap["3"] = 20; myMap["4"] = 21;
    myMap["5"] = 22; myMap["6"] = 23; myMap["7"] = 24; myMap["8"] = 25;
    myMap["9"] = 26; myMap["a"] = 27; myMap["b"] = 28; myMap["c"] = 29;
    myMap["d"] = 30; myMap["w"] = 31; myMap["f"] = 32;

    myMap["P"] = 0; myMap["PB"] = 1; myMap["PC"] = 2; myMap["PD"] = 3;
    myMap["PE"] = 4; myMap["I"] = 5; myMap["IB"] = 6; myMap["DL"] = 7;
    myMap["DLB"] = 8; myMap["DLC"] = 9; myMap["DC"] = 10; myMap["DCB"] = 11;
    myMap["TP"] = 12; myMap["SL"] = 13; myMap["SLB"] = 14; myMap["ST"] = 15;
    myMap["W"] = 16; myMap["WB"] = 17; myMap["WC"] = 18; myMap["WD"] = 19;
    myMap["WE"] = 20; myMap["WF"] = 21; myMap["SR"] = 22; myMap["A"] = 23;
    myMap["AB"] = 24; myMap["AC"] = 25; myMap["E"] = 26;  myMap["EB"] = 27;
    myMap["T"] = 28; myMap["TB"] = 29; myMap["F"] = 30; myMap["N"] = 31;
    myMap["NB"] = 32; myMap["S"] = 33; myMap["D"] = 34; myMap["L"] = 35;

    return ppTable[myMap[stackVal]][myMap[readVal]];
}

//Identifier helper
bool iHelp(string *stackVal, string *read, vector<string> *iStack) {
    string chartVal;
    iStack->push_back(*stackVal);
    bool tempValid= true;
    string tempInput = (*read);
    string tempRead;
    while(!tempInput.empty()) {
        *stackVal = iStack->back();
        iStack->pop_back();
        if(tempRead == "") {
            tempRead = tempInput[0];
            tempInput.erase(tempInput.begin());
        }
        if (tempRead == *stackVal) {
            //if my stack is empty was here
            tempRead = "";
        } else {
            chartVal = parse(*stackVal, tempRead);
            if (chartVal == "|") {chartVal = "blank";}
            if (chartVal == "blank") {
                    tempValid = false;
                    break;
            } else if (chartVal == "z") {
                tempValid = false;
                std::cout << "Expected ; before " << (*read) << "\n";
                break;
            } else if (chartVal == "v") {
                tempValid = false;
                std::cout << "Expected var before " << (*read) << "\n";
                break;
            } else if (chartVal == "g") {
                std::cout << "Expected begin before " << (*read) << "\n";
                tempValid = false;
                break;
            } else if (chartVal == "e") {
                std::cout << "Expected end.\n";
                tempValid = false;
                break;
            } else if (chartVal == "i") {
                std::cout << "Expected title before " << (*read) << "\n";
                tempValid = false;
                break;
            } else if (chartVal == "y") {
                std::cout << "Expected , after “value=”\n";
                tempValid = false;
                break;
            }else if( chartVal == "λ")
                continue;
            else {
                istringstream iss(chartVal);
                vector<string> tokens;
                string token;
                while(iss >> token) {
                    tokens.push_back(token);
                    }
                reverse(tokens.begin(), tokens.end());
                for (auto x : tokens) {
                    iStack->push_back(x);
                }
            }
        }
    }
    if(tempValid) {
        (*read) = tempRead;
        istringstream iss(chartVal);
        string token,temp;
        while(iss >> token) {
            temp = token;
            iStack->pop_back();
        }
    }
    return tempValid;
}

//Puts all the words separated by a space from "finalf23" into a string
void createStack(vector<string> *myStack) {
    ifstream inputFile("finalf23.txt");
    if (!inputFile.is_open()) {
        cerr << "Error opening file!" << std::endl;
        return;
    }
    string line, word;
    while (getline(inputFile, line)) {
        stringstream ss(line);
        while (ss >> word) {
            if (!word.empty() && word[word.size() - 1] == ';' && word != ";")
{
                word.pop_back();
                if (!word.empty()) {
                    myStack->push_back(word);
                    myStack->push_back(";");
                }
            }else if (!word.empty() && word[word.size() - 1] == ',' && word !=
",") {
                word.pop_back();
                if (!word.empty()) {
                    myStack->push_back(word);
                    myStack->push_back(",");
                }
            } else if (!word.empty() && word[0] == '(' && word != "(") {
                word.pop_back();
                if (!word.empty()) {
                    myStack->push_back("(");
                    myStack->push_back(word.substr(1));
                }
            } else if (!word.empty() && word[word.size() - 1] == ')' && word
!= ")") {
                word.pop_back();
                if (!word.empty()) {
                    myStack->push_back(word);
                    myStack->push_back(")");
                }
            } else {
                myStack->push_back(word);
            }
        }
    }
    inputFile.close();
}

bool checkGrammar(vector<string> *program) {
    vector<string> *input = new vector<string>(*program);

    //Test the Grammar

    //Step 3: Setup the pre-loop declarations
    vector<string> myStack;
    string read;
    string stackVal;
    string chartVal;
    bool valid;


    //Create the program
    std::cout << "Testing input" << "\n";
    //Begin the Stack
    myStack.push_back("end.");
    myStack.push_back("P");
    //While loop to test word
    while(!myStack.empty()) {
        stackVal = myStack.back();
        myStack.pop_back();
        if (read == "") {
            read = (*input)[0];
            input->erase(input->begin());
        }
        if (read == stackVal) {
            if (myStack.empty()) {
                valid = true;
                break;
            }
            read = "";
        } else {
            //Handle Identifiers
            if(read[0] == 'a'||(read[0] == 'b' && read != "begin")|| read[0]
== 'c'||read[0] == 'd'||
                (read[0] == 'w' && read != "write")||read[0] == 'f') {
                if(!(iHelp(&stackVal, &read, &myStack))) {
                        valid = false;
                        break;
                }
            }
            chartVal = parse(stackVal, read);
            if (chartVal == "|") {chartVal = "blank";}
            if (chartVal == "z") {
                std::cout << "Expected ; before " << read << "\n";
                valid = false;
                break;
            } else if (chartVal == "p") {
                std::cout << "Expected begin before " << read << "\n";
                valid = false;
                break;
            } else if (chartVal == "v") {
                valid = false;
                std::cout << "Expected var before " << read << "\n";
                break;
            } else if (chartVal == "e") {
                std::cout << "Expected end.\n";
                valid = false;
                break;
            } else if (chartVal == "i") {
                std::cout << "Expected title before " << read << "\n";
                valid = false;
                break;
            } else if (chartVal == "y") {
                std::cout << "Expected , after “value=”\n";
                valid = false;
                break;
            } else if (chartVal == "blank") {
                valid = false;
                break;
            }
            else if( chartVal == "λ")
                continue;
            else {
                istringstream iss(chartVal);
                vector<string> tokens;
                string token;
                while(iss >> token) {
                    tokens.push_back(token);
                }
                reverse(tokens.begin(), tokens.end());
                for (auto x : tokens) {
                    myStack.push_back(x);
                }
            }
        }
    }
    //Check results of string
    if (valid) {
        std::cout << "The input is accepted.\n";
    }
    else {
        std::cout << "\nThe input is rejected.\n";
    }
    return valid;

}
```

```
void compileMe(vector<string> *program) {
//Reserved words
    vector<string> reserved {"program", "vars", "begin", "integer", "end."};

    //Part I:   Program Title
    program->pop_back(); //Pop Program
    string title = "";
    title = program->back();
    cout << " ===== " << title << " ===== \n";
    program->pop_back(); //Pop the Title
    program->pop_back(); //Pop ;



    //Part II:  Variable Declarations
    map<string,int> vars;
    program->pop_back(); //Pop var

    while(program->back() != ":") {
        if (find(reserved.begin(), reserved.end(), program->back()) !=
reserved.end()) {
            std::cout << "Reserved word '" << program->back()<< "' cannot be a
variable name. Cannot Compile...\n";
            exit(1);
        }
        vars.insert({program->back(), nun});
        program->pop_back();
        if (program->back() == ",") {
            program->pop_back(); //Pop ,
        }
    }

    program->pop_back(); //Pop :
    program->pop_back(); //Pop Type Integer
    program->pop_back(); //Pop ;



    //Part III: Program Begin
    program->pop_back(); //Pop Begin

    vector<string> expression;
    string varName;

    while(program->back() != "end") {

        //Write
        if (program->back() == "write") {

            program->pop_back(); program->pop_back(); //Pop write and (

            if(program->back() == "“value=”") {
                program->pop_back(); program->pop_back(); //Pop “value=”,
                if (vars.find(program->back()) == vars.end()) {
                    std::cout << "Variable Not Found\n";
                }else if (vars[program->back()] == nun) {
                    std::cout << "Null Value\n";
                } else {
                    std::cout << "value = " << vars[program->back()] << "\n";
                }
                program->pop_back(); //Pop variable

            } else {
                if (vars.find(program->back()) == vars.end()) {
                    std::cout << "Variable Not Found\n";
                }else if (vars[program->back()] == nun) {
                    std::cout << "Null Value\n";
                } else {
                    std::cout << vars[program->back()] << "\n";
                }
                program->pop_back(); //Pop element
            }

            program->pop_back(); program->pop_back(); //Pop ) and ;
        }

        //Variable
        else if (vars.find(program->back()) != vars.end()) {

            varName = program->back();
            program->pop_back(); program->pop_back(); //Pop variable and =

            while((program->back() != ";")) {
                if (isalpha(program->back()[0])) {
                    if (vars.find(program->back()) == vars.end()) {
                        std::cout << "Error:Undeclared variable in 
evaluation\n";
                        exit(1);
                    }else if (vars[program->back()] == nun) {
                        std::cout << "Error:Evaluation with a Null 
variable\n";;
                    } else {

expression.push_back(to_string(vars[program->back()]));
                    }
                } else {
                    expression.push_back(program->back());
                }
                program->pop_back(); //Pop the current
            }

            vars[varName] = evaluate(expression);
            expression.clear();
            program->pop_back(); //Pop ;
        }

        //Not Found
        else if (program->back() == "end." || program->back() == "end") {
            break;
        } else{
            std::cout << "Undeclared/Unassigned Variable Exception...\n";
            exit(1);
        }
    }

    //Part IV: Program End
    program->pop_back(); //Pop end.
    if (!program->empty()) {
        std::cout << "Something went wrong";
    }
}
```

Compiler output

```
Output of Compiler

Testing input
The input is accepted.
Now Compiling...

 ===== f2023 =====
5
value = 42
```