Andrew Le

Victor Balcazar

Monique Coutreras

CSPC 323 Final Project

Keywords

<prog> = A

<identifier> = B

<more-id-digit> = C

<dec-list> = D

<dec> = E

<type> = F

<stat-list> = G

<stat> = H

<print> = I

<assign> = J

<expr> = K

<term> = L

<factor> = M

<number> = N

<sign> = O

<more-digit> = P

<digit> = Q

<id> = R

BEGIN = t

INTEGER = u

PRINT = v

END. = w

Original Grammar

A-> PROGRAM B ; D t G w

B-> R C

C-> Q C | R C | LAMBDA

D-> F : E ;

E-> B , E | B

F-> u

G-> H | H G

H-> I | J

I-> v ( B ) ;

J-> B = K ;

K-> K + L | K - L | L

L-> L \* M | L / M | M

M-> B | N | ( K )

N-> O Q P

O-> + | - | LAMBDA

P-> Q P | LAMBDA

Q-> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

R-> p | q | r | s

BNF Grammar

A-> PROGRAM B ; D t G w

B-> R C

C-> Q C

C-> R C

C-> LAMBDA

D-> F : E ;

E-> B , E

E-> B

F-> u

G-> H

G-> H G

H-> I

H-> J

I-> v ( B ) ;

J-> B = K ;

K-> K + L

K-> K - L

K-> L

L-> L \* M

L-> L / M

L-> M

M-> B

M-> N

M-> ( K )

N-> O Q P

O-> +

O-> -

O-> LAMBDA

P-> Q P

P-> LAMBDA

Q-> 0

Q-> 1

Q-> 2

Q-> 3

Q-> 4

Q-> 5

Q-> 6

Q-> 7

Q-> 8

Q-> 9

R-> p

R-> q

R-> r

R-> s

LR-Parser

A -> PROGRAM B ; D t G w

B -> R C

C -> Q C

C -> R C

C -> LAMBDA

D -> F : E ;

E -> B S

S -> , E

S -> LAMBDA

F -> u

G -> H X

X -> G

X-> LAMBDA

H -> I

H -> J

I -> v ( B ) ;

J -> B = K ;

K -> L Y

Y -> + L Y

Y -> - L Y

Y -> LAMBDA

L -> M Z

Z -> \* M Z

Z -> / M Z

Z -> LAMBDA

M -> B

M -> N

M -> ( K )

N -> O Q P

O-> +

O-> -

O-> LAMBDA

P-> Q P

P-> LAMBDA

Q-> 0

Q-> 1

Q-> 2

Q-> 3

Q-> 4

Q-> 5

Q-> 6

Q-> 7

Q-> 8

Q-> 9

R-> p

R-> q

R-> r

R-> s

|  |  |  |
| --- | --- | --- |
| **States** | **FIRST** | **FOLLOW** |
| **A** | PROGRAM | $ |
| **B** | p q r s | ; , ) = \* / + - |
| **C** | 0 1 2 3 4 5 6 7 8 9 LAMBDA | t |
| **D** | u | t |
| **E** | p q r s | ; |
| **S** | , LAMBDA | ; |
| **F** | u | : |
| **G** | p q r s v | w |
| **X** | p q r s v LAMBDA | w |
| **H** | p q r s v | p q r s w v |
| **I** | v | p q r s w v |
| **J** | p q r s | p q r s w v |
| **K** | p q r s + - 0 1 2 3 4 5 6 7 8 9 ( | ; ) |
| **Y** | + - LAMBDA | ; ) |
| **L** | p q r s + - 0 1 2 3 4 5 6 7 8 9 ( | + - ; ) |
| **Z** | \* / LAMBDA | + - ; ) |
| **M** | p q r s + - 0 1 2 3 4 5 6 7 8 9 ( | \* / + - ; ) |
| **N** | + - 0 1 2 3 4 5 6 7 8 9 | \* / + - ; ) |
| **O** | + - LAMBDA | 0 1 2 3 4 5 6 7 8 9 |
| **P** | 0 1 2 3 4 5 6 7 8 9 LAMBDA | \* / + - ; ) |
| **Q** | 0 1 2 3 4 5 6 7 8 9 | 0 1 2 3 4 5 6 7 8 9 \* / + - ; ) = |
| **R** | p q r s | 0 1 2 3 4 5 6 7 8 9 ; , ) = + - \* / |

Compiler Code

/\*

Author: Andrew Le

Email: andrewle19@csu.fullerton.edu

4/17/17

Compiler that transforms a text file with code into a c++ executable code if it has no errors present

\*/

#include <iostream>

#include <string>

#include "identify.h"

#include "predictiveParsingTable.h"

#include <fstream>

using namespace std;

int main()

{

predictiontable compiler; // intialize the predictive parsing table

ifstream data; // handles opening and reading of the file data.txt

ofstream output; // handles oppening and writing of the file newdata.txt

string statement; // takes in lines from the txt file

identify parser; // parses through txt file to take out undesirables

data.open("finalv1.txt"); // opens the code file

// checks if file successfully opened

if (data.is\_open())

{

// takes in lines as long as the file is not at eof

while (data.eof() != true)

{

getline(data, statement); // gets the line

// checks for comment lines/extra white space removes all of them saves data to array in class

parser.check(statement);

}

// opens the finalv2.txt file or writes to it depending on if it exists

output.open("finalv2.txt");

parser.printfile(output); // prints all the new data to file

cout << "Removing Comment Lines and Unessecary Whitespace" << endl;

output.close(); // close file

if(parser.checkReserved() && parser.checkColon())

{

string statement; // combine all the parsed words in one statement

// inserts parsed words into one statement

// if any of the words are the reserved words replace them with special placeholder

for (int i = 0; i < parser.getSize(); i++)

{

if(parser.getWord(i) == "PROGRAM")

{

statement += 'z';

}

else if(parser.getWord(i) == "INTEGER")

{

statement += 'u';

}

else if(parser.getWord(i) == "PRINT")

{

statement += 'v';

}

else if(parser.getWord(i) == "END.")

{

statement += 'w';

}

else if(parser.getWord(i) == "BEGIN")

{

statement += 't';

}

// not a reserved word so we just add

else

{

statement += parser.getWord(i);

}

}

// checks if code is accepted then proceeds ask user which language user wants to compile code in

if(compiler.check(statement))

{

int choice; // user choice one what to comple

bool quit = false; // quits when user picks choice

cout << "Accepted Code" << endl;

while(quit != true)

{

// menu of langagues

cout << "1. Compile to C++" << endl;

cout << "2. Compile to Python" << endl;

cout << "Enter choice(1-2):";

cin >> choice;

if(choice == 1) // if user wants c++

{

cout << "Compiling to C++ file..." << endl;

output.open("Final.cpp");

parser.writeCplus(output);

quit = true; // quits the loop

output.close();

}

else if(choice == 2) // if user wants python

{

cout << "Compiling to Python file..." << endl;

output.open("Final.py");

parser.writePython(output);

quit = true; // quits out of loop

output.close();

}

else // handles invalid input

{

cout << "Invalid Input " << endl;

}

}

}

// does not write to cpp if rejected

else

{

cout << "Rejected Code" << endl;

}

}

}

else

{

cout << "Could not open finalv1.txt" << endl;

}

data.close(); // close the file

system("pause");

return 0;

}

// Author: Andrew Le

// Identifier class

// Takes out extra white space/comment lines/blank lines then writes the new code in txt file

#ifndef IDENTIFY\_h

#define IDENTIFY\_h

#include <fstream>

using namespace std;

class identify

{

private:

string words[200]; // can store a program that is 200 lines

string special[9] = {"=","\*","-",";","(",")","+",",",":"}; // list of special characters

int size; // stores the size of the array

string reservedwords[5] = {"PROGRAM","BEGIN","END.","INTEGER","PRINT"}; // reserved words

public:

// default constructor

identify()

{

size = 0;

}

// get the size of the array

// INPUT: NONE

// OUTPUT: size of the words array

int getSize()

{

return size;

}

// gets word at specific index

// INPUT: index of word

// OUTPUT: return word[i]

string getWord(int index)

{

return words[index];

}

// Checks if the string is a reserved word

// INPUT: w- string we are checking

// OUTPUT: returns the length of the word using it as an index

int isReserved(string w)

{

string word;

for (int i = 0; i < w.length(); i++)

{

word += w[i];

for(int j = 0; j < 5; j++)

{

if(word == reservedwords[j])

{

words[size] = word;

size++; //increment the size

return (int)word.length();

}

}

}

return 0;

}

// checks if the string is a special char

// Input: string w

// Output: false/true

bool isSpecial(char w)

{

bool isSpecial = false;

for (int i = 0; i < 9; i++)

{

if (special[i].find(w) != special[i].npos) // checks to see if s is in special char list by comparing if it is found at a position that is not the nth position

{

isSpecial = true;

}

}

return isSpecial;

}

// Checks string to see if there are comment lines and extra uneeded white space and removes both stores data into string array

// Input: string w - line from the file

// Output: None

void check(string w)

{

int comment = 0; // stores the amt of // backslash lines if 2 then it is a comment line we will ignores

int i = isReserved(w); // checks if the string contains reserved word returns the legnth of reserved word and uses the length of that word as the starting point of i

while (i < w.length())

{

// if there is a comment found it will ignore rest of the line

if (comment == 2)

{

break;

}

else

{

if (w[i] == '/') // if a backspace key is a char

{

comment++; // increment comment

i++;

}

else if (isspace(w[i])) // if white space then increment

{

i++;

}

else if (isSpecial(w[i])) // if char is a special char

{

if(words[size].empty()) // if the current word is empty add on the special character

{

words[size] = w[i];

size++;// increment the size

}

else // else the current word has items so add special char to next word and increment 2

{

words[size+1] = w[i];

size += 2;

}

i++;

}

else // then the char is a letter or a digit so just add to existing words array

{

words[size] = words[size] + w[i];

i++;

}

}

}

}

// checks if a colon is missing after the word INTEGER

// INPUT: NONE

// OUTPUT: msg on if it is missing and bool true or false

bool checkColon()

{

bool hasColon = true;

// loops through all words

for(int i = 0; i < 200; i ++)

{

// searchs for the INTEGER KEY Word

if(words[i] == "INTEGER")

{

// check if next word is : if not then return false

if (words[i+1] != ":")

{

hasColon = false;

}

}

}

// error message

if(hasColon != true)

{

cout << ": is Missing" << endl;

}

return hasColon;

}

// check reserved checks all the words making sure none of the required reserved words are missing

// INPUT: None

// OUTPUT: msg on which are missing and bool true or false

bool checkReserved()

{

// stores result of test for BEGIN/INT/PROGRAM/END.

bool hasInt = false, hasBegin = false, hasProg = false, hasEnd = false;

for(int i = 0; i < 200; i++)

{

if(words[i] == "INTEGER")

{

hasInt = true;

}

if(words[i] == "BEGIN")

{

hasBegin = true;

}

if(words[i] == "PROGRAM")

{

hasProg = true;

}

if(words[i] == "END.")

{

hasEnd = true;

}

}

// if everything passes the test

if(hasInt == true && hasBegin == true && hasProg == true && hasEnd == true)

{

return true;

}

// error messeges for when each of the reserved words are missing

if(hasInt == false)

{

cout << "INTEGER is expected(integer missing or spelled wrong)" << endl;

}

if(hasBegin == false)

{

cout << "BEGIN is expected(begin missing or spelled wrong)" << endl;

}

if(hasProg == false)

{

cout << "PROGRAM is expected(program missing or spelled wrong)" << endl;

}

if(hasEnd == false)

{

cout << "END. is expected(end. missing or spelled wrong)" << endl;

}

return false;

}

// prints out all the data properly formated into file

// Input: output ofstream to write to file

// Ouput: None

void printfile(ofstream &out)

{

int i = 0; // iterator

while (i < size) // loop ends when i reachs the size

{

// if the word index is ; or BEGIN then we go to new line

if (words[i] == ";" || words[i] == "BEGIN")

{

out << words[i] << endl;

}

else

{

out << words[i] << " ";

}

i++; // increment

}

}

// Writes to parsed text to executable c++ file

// INPUT: output stream

// OUTPUT: return executable c++ file

void writeCplus(ofstream &out)

{

int i = 0; //iterator

bool end = false; // bool for when END. is found

// loop ends once END. is found

while (end != true)

{

// if the word is Program conver to c++ headers

if (words[i] == "PROGRAM")

{

out << "#include <iostream>" << endl;

out << "using namespace std;" << endl;

out << "int main()" << endl;

out << "{" << endl;

i += 2;

}

// if word is integer conver to c++ style

else if(words[i] == "INTEGER")

{

out << "int ";

i++; // skip the colon

}

// if the word is begin skip to next index

else if(words[i] == "BEGIN")

{

}

// if semi colon end line

else if(words[i] == ";")

{

out << ";" << endl;

}

// convert print to c++ then place word that is 2 indexs away in middle

else if(words[i] == "PRINT")

{

out << "cout << " << words[i+2] << " << endl;";

i+=3;//increment by 3

}

// end the loop if END. is signaled

else if(words[i] == "END.")

{

out << "return 0;" << endl;

out << "}";

end = true;

}

// if it is a special char then add spaces in between

else if(isSpecial(words[i][0]))

{

// if previous word was special character dont add space before

if(isSpecial(words[i-1][0]))

{

out << words[i] << " ";

}

// add space before since it was a regular expression

else

{

out << " " << words[i] << " ";

}

}

// regular expression

else

{

out << words[i];

}

i++; // increment

}

}

// Writes the parsed txt into an executable python file

// INPUT: output stream to a python file

// OUTPUT: a executable python file

void writePython(ofstream &out)

{

int i = 0; //iterator

bool end = false; // bool for when END. is found

bool begin = false; // bool to signal the begining

// loop ends once END. is found

while (end != true)

{

// if the word index BEGIN then we start to write the code

if (words[i] == "BEGIN")

{

begin = true;

i++;

}

// write everything after begin in python

if(begin)

{

if(words[i] == ";")

{

out << '\n';

}

// convert print to python

else if(words[i] == "PRINT")

{

out << "print";

}

// end the loop if END. is signaled

else if(words[i] == "END.")

{

end = true;

}

// if it is a special char then add spaces in between

else if(isSpecial(words[i][0]))

{

// if previous word was special character dont add space before

if(isSpecial(words[i-1][0]))

{

out << words[i] << " ";

}

// add space before since it was a regular expression

else

{

out << " " << words[i] << " ";

}

}

// regular expression

else

{

out << words[i];

}

}

i++; // increment

}

}

};

#endif /\* indentify\_h \*/

/\*

AUTHOR: Andrew Le

Email: andrewle19@csu.fullerton.edu

4/26/17

Predictive parsing table class

accepts/ rejects statement depending on the input matching the language

also handles returning specefic errors

\*/

#ifndef predicitveParsingTable\_h

#define predicitveParsingTable\_h

#include <string>

#include <stack>

using namespace std;

class predictiontable

{

private:

const int charSize = 30;

const int stateSize = 22;

stack<char> s; // char stack

// array of language

char specialcharacters[30] = {'z','t','u','v','w','p','q','r','s','+','-','\*','/','=',

'(',')',',',';',':','0','1','2','3','4','5','6','7','8','9','$'};

// array of states

char states[22] = {'A','B','C','D','E','S','F','G','X','H','I','J','K','Y','L','Z','M',

'N','O','P','Q','R'};

// 2d array of the predictive table

string table[22][30] = {"zB;DtGw","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e", "e","e","m","e","e","RC","RC","RC","RC","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e", "e","e","e","e","e","RC","RC","RC","RC","l","l","l","l","l","e","l","l","l","e","QC","QC","QC","QC","QC","QC","QC","QC","QC","QC","e", "e","e","F:E;","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e", "e","m","e","e","e","BS","BS","BS","BS","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e", "e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e",",E","l","e","e","e","e","e","e","e","e","e","e","e","e", "e","e","u","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e",

"e","e","e","HX","e","HX","HX","HX","HX","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e", "e","e","e","G","l","G","G","G","G","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e", "e","e","e","I","e","J","J","J","J","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e", "e","e","e","v(B);","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e", "e","e","e","e","e","B=K;","B=K;","B=K;","B=K;","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e", "e","e","e","m","e","LY","LY","LY","LY","LY","LY","e","e","e","LY","e","e","e","e","LY","LY","LY","LY","LY","LY","LY","LY","LY","LY","e",

"e","e","e","e","e","e","e","e","e","+LY","-LY","e","e","e","e","l","e","l","e","e","e","e","e","e","e","e","e","e","e","e", "e","e","e","e","e","MZ","MZ","MZ","MZ","MZ","MZ","e","e","e","MZ","e","e","e","e","MZ","MZ","MZ","MZ","MZ","MZ","MZ","MZ","MZ","MZ","e", "e","e","e","m","e","e","e","e","e","l","l","\*MZ","/MZ","e","e","l","e","l","e","e","e","e","e","e","e","e","e","e","e","e",

"e","e","e","e","e","B","B","B","B","N","N","e","e","e","(K)","e","e","e","e","N","N","N","N","N","N","N","N","N","N","e",

"e","e","e","e","e","e","e","e","e","OQP","OQP","e","e","e","e","e","e","e","e","OQP","OQP","OQP","OQP","OQP","OQP","OQP","OQP","OQP","OQP","e",

"e","e","e","e","e","e","e","e","e","+","-","e","e","e","e","e","e","e","e","l","l","l","l","l","l","l","l","l","l","e",

"e","e","e","e","e","e","m","m","e","l","l","l","l","e","e","l","e","l","e","QP","QP","QP","QP","QP","QP","QP","QP","QP","QP","e",

"e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","0","1","2","3","4","5","6","7","8","9","e",

"e","e","e","e","e","p","q","r","s","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e","e"};

public:

// default constructor

predictiontable()

{

s.push('A');

}

// pushs to stack depending which state and key is wanted

// INPUT: int stateindex- index of State, keyindex- index of key

// OUTUPUT: none

void advanceState(int stateindex,int keyindex)

{

s.pop(); // pops the top of the stack

// loop to push states into the stack starts at end of string state because we want to

// push from then end first

if (table[stateindex][keyindex] == "e")

{

cout << "error" << endl;

cout << stateindex << " " << keyindex << endl;

}

for(int i = (int)table[stateindex][keyindex].length()-1; i >= 0 ;i--)

{

s.push(table[stateindex][keyindex][i]); // push the table index to stack

// cout << "push: " << s.top() << endl;

}

}

// Checks whether the word is accepted by the Predictive parsing Table

// Uses many helper functions to accomplish this

// INPUT: string word- the target Word

// OUTPUT: Returns True or False depending on accepted or rejected

bool check(string word)

{

bool accepted = false; // if the word is accepted or rejected

string match; // string match to check if the input string matches

int i = 0; // the index of the key we want to search

// until the stack is empty

while(s.empty() == false)

{

// change any alphabetical character to a lower to match the parsing table

if(isalpha(word[i]))

{

word[i] = tolower(word[i]);

}

if(checkspecial(word[i])) // check if search key is valid

{

// cout << "read: " << word[i] << endl;

// cout << "pop: " << s.top() << endl;

if(checkspecial(s.top()) && s.top() == word[i]) // check if top of stack is special character and is equal to word

{

match += s.top(); // append top to match

// cout << "match: " << s.top() << endl;

s.pop(); // pop the stack3

i++;

}

else if(checkspecial(s.top())) // checks if the top of the stack is a special char if so then it is missing because no match

{

cout << s.top() << " is Missing" << endl;

return false;

}

else if(checkState(s.top())) // checks if top of stack is a state

{

advanceState(stateindex(s.top()),keyindex(word[i])); // advances the state

}

else if(s.top() == 'l') // if the state is lamda then pop the stack

{

s.pop();

}

else if(s.top() == 'e') // if illegal expression error code is at the top of stack

{

cout << "Illegal Expression" << endl;

return false;

}

else if(s.top() == 'm') // if missing semicolon is at the top of the stack

{

cout << "; is Missing" << endl;

return false;

}

else // if none of above is true reject the input

{

return false;

}

}

else // if the search key is not valid

{

cout << "Illegal Expression" << endl;

return false;

}

}

// final check if the word is in the language

if(match == word)

{

accepted = true;

}

return accepted;

}

// check if the key is in the special character array or language

// INPUT: char key- key we want to search for

// OUPUT: true or false

bool checkspecial(char key)

{

bool special = false;

for(int i = 0; i < charSize; i++)

{

// checks if key is a special char

if(specialcharacters[i] == key)

{

special = true;

}

}

return special;

}

// Find which state index we are at

// INPUT: char state- the letter of the state

// OUTPUT: returns index of the state

bool checkState(char state)

{

for(int i = 0;i < stateSize;i++)

{

if(states[i] == state) // checks if state is actually a state

{

return true; // returns true

}

}

return false; // returns false

}

// find the index of key(use after we know key is already in index

// INPUT: char key- the key we want to search for

// OUPUT: the index of the key

int keyindex(char key)

{

// if the key is an aphabetical char then change it to lower case to match the table

if(isalpha(key))

{

key = tolower(key);

}

for(int i = 0;i < charSize;i++)

{

if(specialcharacters[i] == key) // searchs for location of key in list

{

return i; // returns the index

}

}

return -1; // returns -1 if not found

}

// Find which state index we are at

// INPUT: char state- the letter of the state

// OUTPUT: returns index of the state

int stateindex(char state)

{

for(int i = 0;i < stateSize;i++)

{

if(states[i] == state) // searches for location of key in list

{

return i; // returns the index

}

}

return -1; // returns -1 if not found

}

};

#endif /\* predicitveParsingTable\_h \*/