# CECS 444 Final Project – TypeCheck

Source Code:

package typeCheck**;**

**import** java**.**io**.**FileNotFoundException**;**

**import** java**.**io**.**FileReader**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**Scanner**;**

/\*\*

\* Final version

\*

\* @author James Chavis

\* @author Giang Truong

\*

\*/

public class main **{**

/\*\*

\* Default constructor for main class

\* @param args - always void

\*/

public static void main**(**String**[]** args**)** **{**

Scanner in **=** **null;**

typecheck mycheck **=** **new** typecheck**();**

ArrayList**<**String**>** programString **=** **new** ArrayList**<**String**>();**

ArrayList**<**String**>** SemanticprogramString **=** **new** ArrayList**<**String**>();**

**try** **{**

in **=** **new** Scanner**(new** FileReader**(**"../CsimpleCalculator.txt"**));**

// pick up all string in program and fill the arraylist withit.

**while(**in**.**hasNextLine**()){**

String input **=** in**.**nextLine**();**

input **=** input**.**trim**();**

// add into arraylist

**if(**input**.**isEmpty**()** **||** **(**input**.**equals**(null))** **||** **(**input**.**equals**(**"EOF"**))){**

System**.**out**.**println**(**"ignore line"**);**

**}**

**else{**

programString**.**add**(**input**);**

**}**

**}**

// now run through the arraylist check for declaration.

// if declaration return true. we will remove the item from arraylist.

// else keep the item for semantic check in next loop.

**for(**int i **=** 0**;** i **<** programString**.**size**();** i**++){**

System**.**out**.**println**(**programString**.**get**(**i**));**

**if(!(**mycheck**.**Declaration**(**programString**.**get**(**i**)))){**

// we got a string that is not a declaration

// add it to the SemanticprogramString arraylist for semantic check.

SemanticprogramString**.**add**(**programString**.**get**(**i**));**

**}**

**}**

System**.**out**.**println**(**"END OF DECLARATION CHECK"**);**

// semantoc check loop

**for(**int j **=** 0**;** j **<** SemanticprogramString**.**size**();** j**++){**

System**.**out**.**println**(**SemanticprogramString**.**get**(**j**));**

mycheck**.**check**(**SemanticprogramString**.**get**(**j**));**

**}**

**if(!(**mycheck**.**braceCount**())){**

System**.**out**.**println**(**"error braces count"**);**

**}**

/\*\*

\* Call late checks

\*/

SemanticprogramString**.**forEach**(**inputLine **->{**

**if(!**typecheck**.**lateCheck**(**inputLine**)){**

System**.**out**.**println**(**inputLine **+** " \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Late check fail\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"**);**

**}**

**if(!**typecheck**.**lateSyntaxCheck**(**inputLine**)){**

System**.**out**.**println**(**inputLine **+** "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Late syntax check failed\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"**);**

**}**

**});**

**}** **catch** **(**FileNotFoundException e**)** **{**

// TODO Auto-generated catch block

e**.**printStackTrace**();**

**}**

in**.**close**();**

**}**

**}**

package typeCheck**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**Arrays**;**

**import** java**.**util**.**HashMap**;**

**import** java**.**util**.**List**;**

**import** java**.**util**.**Scanner**;**

**import** java**.**util**.**regex**.**Pattern**;**

/\*\*

\* Completes main testing logic

\* Final version

\* @author James Chavis

\* @author Giang Truong

\*

\*/

public class typecheck **{**

//private static String mystring = "";

private static int openbracecount **=** 0**;**

private static int closebracecount **=** 0**;**

private static int maincount **=** 0**;**

private static int functioncount **=** 0**;**

private static int returncount **=** 0**;**

private static PatternMatching mycheck **=** **new** PatternMatching**();**

private static HashMap**<**String**,** ArrayList**<**String**>>** mFunctionIndex **=** **new** HashMap**<**String**,** ArrayList**<**String**>>();**

// hashmap that store variable info

// key is variable name, value is the variable type

private static HashMap**<**String**,** String**>** mVariableIndex **=** **new** HashMap**<**String**,** String**>();**

// hashmap that store pointer info

// key is pointer name, value is the pointer type

private static HashMap**<**String**,** String**>** mPointerIndex **=** **new** HashMap**<**String**,** String**>();**

public typecheck**(){**

mPointerIndex**.**put**(**"null"**,** "char"**);**

mVariableIndex**.**put**(**"null"**,** "int"**);**

**}**

// remove all comma, semi colon

public static String RemoveAllCommaNSemicolon**(**String in**){**

in **=** in**.**replace**(**","**,** ""**);**

in **=** in**.**replace**(**";"**,** ""**);**

in **=** in**.**trim**();**

**return** in**;**

**}**

/\*\*

\* input = RemoveAllCommaNSemicolon(input);

keep track of declared variable and function

so when we get an expression of the form

assignment we can check to see if

the variable exist , and does the function

return type match the variable type.

\* @param input line of code to be detected

\*/

public static void check**(**String input**){**

//check number of open and close curly brace

**if(**input**.**contains**(**"{"**)){**

input **=** input**.**replace**(**"{"**,** ""**);**

input **=** input**.**trim**();**

**}**

**if(**input**.**contains**(**"}"**)){**

input **=** input**.**replace**(**"}"**,** ""**);**

input **=** input**.**trim**();**

**}**

// trim the string and remove everything un needed

// but keep the original input string to match the pattern

String ss **=** RemoveAllCommaNSemicolon**(**input**);**

// split the string into array of string

// this arr is global to all if else if statement but if i forget

// i can still use input and do the trimming inside the block.

String**[]** arr **=** ss**.**split**(**" "**);**

// check to make sure we only have 1 main

// and main pass the pattern int main()

**if(**input**.**contains**(**"main()"**)){**

maincheck**(**input**,** mycheck**);**

**}**

**else** **if(**mycheck**.**visitMainret**(**input**)){**

System**.**out**.**println**(**"return 0 for main"**);**

**}**

**else** **if(**input**.**contains**(**"printf"**)){**

String newin **=** input**.**replaceAll**(**"\""**,** ""**);**

**if(!(**mycheck**.**visitPrintf**(**newin**))){**

System**.**out**.**println**(**"printf error"**);**

**}**

**else{**

System**.**out**.**println**(**"printf passed"**);**

**}**

**}**

**else** **if(**input**.**contains**(**"scanf"**)){**

String newin **=** input**.**replaceAll**(**"\""**,** ""**);**

**if(!(**mycheck**.**visitScanf**(**newin**))){**

System**.**out**.**println**(**"scanf error"**);**

**}**

**else{**

System**.**out**.**println**(**"scanf passed"**);**

**}**

**}**

**else** **if(**input**.**contains**(**"include"**)){**

**if(!(**mycheck**.**visitStdio**(**input**))){**

System**.**out**.**println**(**"stdio error"**);**

**}**

**else{**

System**.**out**.**println**(**"stdio passed"**);**

**}**

**}**

// skipping if detect var/func/ptr/arr decl

**else** **if(**mycheck**.**visitVariableDeclaration**(**input**)){**

System**.**out**.**println**(**"skip var declaration"**);**

**}**

**else** **if(**mycheck**.**visitPointer**(**input**)){**

System**.**out**.**println**(**"skip ptr declaration"**);**

**}**

**else** **if(**mycheck**.**visitArray**(**input**)){**

System**.**out**.**println**(**"skip array declaration"**);**

**}**

**else** **if(**mycheck**.**visitfunction**(**input**)){**

System**.**out**.**println**(**"skip function header"**);**

**}**

// check function call

**else** **if(**mycheck**.**visitFunctionCall**(**input**)){**

boolean pass **=** **true;**

String s **=** input**;**

int j **=** 3**;**

s **=** RemoveAllCommaNSemicolon**(**s**);**

s **=** s**.**replaceAll**(**"\\("**,** ""**);**

s **=** s**.**replaceAll**(**"\\)"**,** ""**);**

String sarr**[]** **=** s**.**split**(**" "**);**

// 1st item in array is function name

// go into the function hashmap to check for it existence

**if(!**mFunctionIndex**.**containsKey**(**sarr**[**0**])){**

// no duplicate = no found = function is not declared before it is call

System**.**out**.**println**(**"error 5 function name not found in hashmap"**);**

pass **=** **false;**

**}**

// check for number or argument and param

// access the function hashmap get the value (arraylist) using

// the key (function name), then get the 3rd index from array list

// for the param number (count) compare it against size of function call arr-1 (because

// 1 item in the array is the function header)

**else** **if(!(**Integer**.**parseInt**(**mFunctionIndex**.**get**(**sarr**[**0**]).**get**(**2**).**trim**())** **==** **(**sarr**.**length**-**1**))){**

System**.**out**.**println**(**"error 6 miss match argument/param number"**);**

pass **=** **false;**

**}**

**else{**

// check argument type error , since argument is variable

// go to variable hashmap grab the value (the type)

// compare to the function hashmap param type.

**for(**int i **=** 0**;** i **<** sarr**.**length**-**1**;** i**++){**

// go to variable hashmap grab the value (the type) using sarr[i+1] the key

// compare to the function hashmap param type starting at arraylist 4th index increase by 2 each loop.

**if(!(**mVariableIndex**.**get**(**sarr**[**i**+**1**]).**equals**(**mFunctionIndex**.**get**(**sarr**[**0**]).**get**(**j**)))){**

System**.**out**.**println**(**"error 7 data type miss match argument/param"**);**

pass **=** **false;**

**break;**

**}**

j **=** j**+**2**;**

**}**

**}**

**if(**pass **==** **true){**

System**.**out**.**println**(**"function call passed"**);**

**}**

// reset pass for next run

pass **=** **true;**

**}**

**else** **if(**mycheck**.**visitReturn**(**input**)){**

// check for return type

// return varname;

// remove the semicolon

input **=** RemoveAllCommaNSemicolon**(**input**);**

// arr[] should only have 2 item arr[0] = return, arr[1] = varname

// pull out varname search variable hash table and get the varname data type

// match it vs the function return type (which function? look for functionid = returncount

// returncount start at zero everytime this condition pass it will increase by 1 before exit else if

// pick up the return type (value) from variable hashmap by supply in the key(variable name)

String varDatatype **=** mVariableIndex**.**get**(**arr**[**1**]);**

// compare return type and function return type

**if(!(**varDatatype**.**equals**(**functionReturnType**(**returncount**)))){**

System**.**out**.**println**(**"error code 8 function return type and return data type do not match"**);**

**}**

// update returncount for next return statement

returncount**++;**

**}**

// var = func()

**else** **if(**mycheck**.**visitVarAssignFunc**(**input**)){**

// varname = functioncall(argument1, argument2);

// if we get this statement

// first we replace paranthesis with space then remove ; and ,

input **=** input**.**replaceAll**(**"("**,** " "**);**

input **=** input**.**replaceAll**(**")"**,** " "**);**

// remove comma and semicolon

input **=** RemoveAllCommaNSemicolon**(**input**);**

input **=** input**.**trim**();**

// we get this

// varname = function call

// split them into an array using split and space delimiter

String sVAF**[]** **=** input**.**split**(**" "**);**

// pick up the variable data type by going to the variable hashmap

// supply the key sVAF[0] and get the value (data type) out

// compare it to the function return type (go to function hashmap

// supply the function name for key and get value array then subscript

// zero to get function return type. Compare them if they do not match

// output error 9 else they pass

**if(!(**mVariableIndex**.**get**(**sVAF**[**0**]).**equals**(**mFunctionIndex**.**get**(**sVAF**[**2**]).**get**(**0**)))){**

System**.**out**.**println**(**"error code 9 function return type and variable data type do not match"**);**

**}**

**}**

// fill in 10-16

// 17 address of

// can only be applied to integers, chars, and indexed strings (string[i])

**else** **if(**input**.**contains**(**"&"**)){**

// loop throught the array to get to the item after &

**for(**int i **=** 0**;** i **<** arr**.**length**;** i**++){**

**if(**arr**[**i**].**equals**(**"&"**)){**

// look for the one next to it and see if it is a int/char/string[i]

// go to the variable table and search for the value(data type)

**if(!(**mVariableIndex**.**get**(**arr**[**i**+**1**]).**equals**(**"int"**))** **&&**

**!(**mVariableIndex**.**get**(**arr**[**i**+**1**]).**equals**(**"char"**))** **&&** **!**isStringSub**(**arr**[**i**+**1**])){**

// not int / char or string[] so output error code 17

System**.**out**.**println**(**"error code 17 & of something that is not int/char/string[]"**);**

**}**

**else{**

System**.**out**.**println**(**"& passed"**);**

**}**

**}**

**}**

**}**

// check ^ error code 18 only be applied to integer pointers and char pointers

// ^ var where var is a pointer

**else** **if(**input**.**contains**(**"^"**)){**

// loop throught the array to get to the item after ^

**for(**int i **=** 0**;** i **<** arr**.**length**;** i**++){**

**if(**arr**[**i**].**equals**(**"^"**)){**

// look for the one next to it and see if it is a int/char pointer

// go to the pointer table and search for the value(data type)

**if(** **!(**mPointerIndex**.**get**(**arr**[**i**+**1**]).**equals**(**"int"**))** **&&**

**!(**mPointerIndex**.**get**(**arr**[**i**+**1**]).**equals**(**"char"**))** **){**

// not int / char pointer so output error code 18

System**.**out**.**println**(**"error code 18 ^ of something that is not int/char pointer"**);**

**}**

**else{**

System**.**out**.**println**(**"^ passed"**);**

**}**

**}**

**}**

**}**

**else** **if(**input**.**isEmpty**()** **||** **(**input**.**equals**(null))** **||** **(**input**.**equals**(**"EOF"**))){**

System**.**out**.**println**(**"ignore line"**);**

**}**

// if we get here the line failed all matching

**else{**

System**.**out**.**println**(**"failed to pass the typecheck"**);**

**}**

**}**

/\*\*

\* is String[] return true if we have a char type []

return false otherwise

\* @param var the line to be checked

\* @return true of false on fail or pass

\*/

public static boolean isStringSub **(**String var**)** **{**

PatternMatching c **=** **new** PatternMatching**();**

// match pattern String[]

**if(**c**.**visitArrSub**(**var**)){**

var **=** var**.**replace**(**"["**,** " "**);**

String**[]** v **=** var**.**split**(**" "**);**

// v[0] should be the variable name

// now check the array hashmap to see is it data type

// = to char if it is we got a string[] else return false

**if(**mVariableIndex**.**get**(**v**[**0**]).**equals**(**"char"**)){**

**return** **true;**

**}**

**}**

**return** **false;**

**}**

/\*\*

\* take in returncount which is the equivalence of functioncount

get all of the item in function hashmap compare the functioncount

to returncount, if functioncount = returncount get the return type of

that function.

\* @param which function to get

\* @return string of the return type

\*/

public static String functionReturnType**(**int returncount**){**

String**[]** funcnameList **=** getFunctionDictionary**();**

// convert int to string and trim it

String sreturncount **=** ""**+**returncount**;**

sreturncount **=** sreturncount**.**trim**();**

// loop the whole list of function name

**for(**int i **=** 0**;** i **<** funcnameList**.**length**;** i**++){**

// go to hashmap of function get the value using the function name key

// value is an array and the 2nd item in the array is the function

// unique id (function count) which match the returncount

**if(**mFunctionIndex**.**get**(**funcnameList**[**i**]).**get**(**1**).**equals**(**sreturncount**)){**

// we found the function for our return call

// now get the return type from the function and return it.

// function return type is 1st item in the arraylist

**return** mFunctionIndex**.**get**(**funcnameList**[**i**]).**get**(**0**);**

**}**

**}**

**return** **null;**

**}**

/\*\*

\* check the main function

\* @param input string to check for

\*/

public static void maincheck**(**String input**,** PatternMatching mycheck**){**

maincount**++;**

// case of more than 1 main() appear

**if(**maincount **>** 1**){**

System**.**out**.**println**(**"error code 1"**);**

**}**

// case of int main() fail the pattern matching

**else** **if(!**mycheck**.**visitmain**(**input**)){**

System**.**out**.**println**(**"error code 2"**);**

System**.**out**.**println**(**"main fail pattern matching not neccessary main with argument"**);**

**}**

**else{**

System**.**out**.**println**(**"main passed"**);**

**}**

**}**

/\*\*

\* counts for brace balancing

\* @return true if the counts match

\*/

public static boolean braceCount**(){**

System**.**out**.**println**(**"open brace count is : "**+** openbracecount**);**

System**.**out**.**println**(**"close brace count is : "**+** closebracecount**);**

**return** **(**openbracecount **==** closebracecount**);**

**}**

/\*\*

\* split the input using split function

add each item in array into the hashmap

the function name will be the key

value is an arraylist of return type, function unique ID

param number(count), 1st param return type, 1st param name,

2nd param return type, 2nd param name etc...

split the string into array of string using space between word

\* @param input The line being checked

\*/

public static void functionPopulate**(**String input**){**

input **=** RemoveAllCommaNSemicolon**(**input**);**

input **=** input**.**replaceAll**(**"\\("**,** ""**);**

input **=** input**.**replaceAll**(**"\\)"**,** ""**);**

String**[]** arr **=** input**.**split**(**"\\s+"**);**

ArrayList**<**String**>** value **=** **new** ArrayList**<**String**>();**

// add the return type first

value**.**add**(**arr**[**0**]);**

// add the unique id, this id is used

// to match it against return type hashmap

value**.**add**(**""**+**functioncount**);**

// update function count by +1

functioncount**++;**

// add number of parameter (param count of the function)

// the size of the arr[] - 2 (return type, name)

value**.**add**(**""**+((**arr**.**length **-** 2**)/**2**));**

// we have 1 or more param

int x **=** 2**;**

**if(!((**arr**.**length **-** 2**)/**2 **==** 0**)){**

// loop through the arr[] add all param

// return type and name into the value arraylist

**for** **(**int i **=** 2**;** i **<** arr**.**length**;** i **=** i**+**2**){**

// param return type

value**.**add**(**arr**[**i**]);**

// param name

value**.**add**(**arr**[**i**+**1**]);**

**}**

**}**

mFunctionIndex**.**put**(**arr**[**1**],** value**);**

**}**

/\*\*

\* builds array of functions to be checked, us containsKey() instead

\* @return String array of function names

\*/

public static String**[]** getFunctionDictionary**()** **{**

// TO-DO: fill an array of Strings with all the keys from the hashtable.

// Sort the array and return it.

List**<**String**>** keys **=** **new** ArrayList**<**String**>();**

**for** **(** String key **:** mFunctionIndex**.**keySet**()** **)** **{**

keys**.**add**(**key**);**

**}**

String **[]** mystrarr **=** **new** String**[**keys**.**size**()];**

**for(**int j **=** 0**;** j **<** mystrarr**.**length**;** j**++){**

mystrarr**[**j**]** **=** keys**.**get**(**j**);**

**}**

**return** mystrarr**;**

**}**

/\*\*

\* builds array of variable to be checked, us containsKey() instead

\* @return String array of variable names

\*/

public static String**[]** getVariableDictionary**()** **{**

// TO-DO: fill an array of Strings with all the keys from the hashtable.

// Sort the array and return it.

List**<**String**>** keys **=** **new** ArrayList**<**String**>();**

**for** **(** String key **:** mVariableIndex**.**keySet**()** **)** **{**

keys**.**add**(**key**);**

**}**

String **[]** mystrarr **=** **new** String**[**keys**.**size**()];**

**for(**int j **=** 0**;** j **<** mystrarr**.**length**;** j**++){**

mystrarr**[**j**]** **=** keys**.**get**(**j**);**

**}**

**return** mystrarr**;**

**}**

/\*\*

\* loop throught the list of key(function name) in mFunctionIndex

return true if found duplicate, false if there is no duplicate

\* @param functionName checks for key violations

\* @return whether there is a key collison

\*/

public static boolean isFunctionNameDuplicate**(**String functionName**)** **{**

// we have nothing in the hashmap mean no function was added at all

// no duplicate possible

**if(**mFunctionIndex**.**isEmpty**()** **==** **true){**

**return** **false;**

**}**

// we have at least 1 function added

**else{**

String**[]** funcnameList **=** getFunctionDictionary**();**

// loop the whole list of function name

**for(**int i **=** 0**;** i **<** funcnameList**.**length**;** i**++){**

// if found an equal (duplicated) return false

**if(**functionName**.**equals**(**funcnameList**[**i**])){**

**return** **true;**

**}**

**}**

**}**

**return** **false;**

**}**

/\*\*

\* loop throught the list of key(variable name) in mVariableIndex

return true if found duplicate, false if there is no duplicate

\*/

public static boolean isVariableNameDuplicate**(**String variableName**)** **{**

// we have nothing in the hashmap mean no variable was added at all

// no duplicate possible

**if(**mVariableIndex **==** **null** **||** mVariableIndex**.**isEmpty**()** **==** **true){**

**return** **false;**

**}**

// we have at least 1 variable added

**else{**

String**[]** variableList **=** getVariableDictionary**();**

// loop the whole list of variable name

**for(**int i **=** 0**;** i **<** variableList**.**length**;** i**++){**

// if found an equal (duplicated) return false

**if(**variableName**.**equals**(**variableList**[**i**])){**

**return** **true;**

**}**

**}**

**}**

**return** **false;**

**}**

/\*\*

\* this function check for the declaration of

\* 1) a variable

\* 2) a pointer

\* 3) array

\* 4) function header

\* INPUT: a string (a single line from the input program)

\* return true if the input pass the general form of declaration checking

\* return false if it does not.

\*/

public static boolean Declaration **(**String input**)** **{**

String ss **=** RemoveAllCommaNSemicolon**(**input**);**

// split the string into array of string

// this arr is global to all if else if statement but if i forget

// i can still use input and do the trimming inside the block.

String**[]** arr **=** ss**.**split**(**" "**);**

//check number of open and close curly brace

**if(**input**.**contains**(**"{"**)){**

openbracecount**++;**

input **=** input**.**replace**(**"{"**,** ""**);**

input **=** input**.**trim**();**

**}**

**if(**input**.**contains**(**"}"**)){**

closebracecount**++;**

input **=** input**.**replace**(**"}"**,** ""**);**

input **=** input**.**trim**();**

**}**

// check common form of var decl

**if(**PatternMatching**.**visitCommonVar**(**input**)** **&&** PatternMatching**.**visitPrimitive**(**arr**[**0**])){**

// check specific form variable declaration pattern

**if(**PatternMatching**.**visitVariableDeclaration**(**input**)){**

// check for variable duplicate

**if(**mVariableIndex**.**containsKey**(**arr**[**1**])){**

// we found duplicate error 3

System**.**out**.**println**(**"error 4 variable name duplicate"**);**

**}**

**else{**

// get the variable name and data type

// save them in the variable hashmap

// arr[1] var name, arr[0] var type

mVariableIndex**.**put**(**arr**[**1**],** arr**[**0**]);**

System**.**out**.**println**(**"variable declaration passed"**);**

**}**

**}**

**else{**

System**.**out**.**println**(**"error variable declaration "**);**

**}**

// return true if the statement pass the general form even if it fail the detail form.

**return** **true;**

**}**

// check common form of ptr

// NOTE here i assume we only have int,char,double,float ptr that why i do not

// check for long and short data type

**else** **if(**PatternMatching**.**visitCommonPtr**(**input**)** **&&** PatternMatching**.**visitPrimitive**(**arr**[**0**])){**

// specific form of pointer declaration

**if(**PatternMatching**.**visitPointer**(**input**)){**

// if it match the pointer declaration then add it into the pointer hashmap

// along with the datatype, key for name, value for data type

mPointerIndex**.**put**(**arr**[**2**],** arr**[**0**]);**

System**.**out**.**println**(**"ptr declaration passed"**);**

**}**

**else{**

System**.**out**.**println**(**"error ptr declaration "**);**

**}**

// return true if the statement pass the general form even if it fail the detail form.

**return** **true;**

**}**

// check common form of array decl

**else** **if(**PatternMatching**.**visitCommonArr**(**input**)** **&&** PatternMatching**.**visitPrimitive**(**arr**[**0**])){**

// specific form of array declaration

**if(**PatternMatching**.**visitArray**(**input**)){**

// add to the variable hash map key = array name value = array data type

// example int n [ 10 ] ; arr[0] = data type, arr[1] = var name

mVariableIndex**.**put**(**arr**[**1**],** arr**[**0**]+**"Arr"**);**

System**.**out**.**println**(**"array declaration passed"**);**

**}**

**else{**

System**.**out**.**println**(**"error Array declaration "**);**

**}**

// return true if the statement pass the general form even if it fail the detail form.

**return** **true;**

**}**

// common func header

**else** **if(**PatternMatching**.**visitcommonfuncheader**(**input**)** **&&** PatternMatching**.**visitReturnType**(**arr**[**0**])){**

// specific func header check

// check function pattern

**if(**PatternMatching**.**visitfunction**(**input**)){**

// check to see if the function name (procedure ID)

// already appear in the hashmap

// if it does error 3 no duplicate procedure ID allowed

// get key from mFunctionIndex and compare to arr[1](the func name)

**if(**mFunctionIndex**.**containsKey**(**arr**[**1**])){**

// we found duplicate error 3

System**.**out**.**println**(**"error 3 function duplicate name found"**);**

**}**

// no duplicated name found

// populate the function hashmap with the function information

**else{**

functionPopulate**(**input**);**

System**.**out**.**println**(**"function header declaration passed"**);**

**}**

**}**

**else** **{**

System**.**err**.**println**(**"failed to pass the typecheck"**);**

**}**

// return true if the statement pass the general form even if it fail the detail form.

**return** **true;**

**}**

// default return false if it do not pass any of the 4 general declaration form

// it is not declaration.

System**.**out**.**println**(**"not declaration "**);**

**return** **false;**

**}**

/\*\*

\* This will run serveral semantic checks (9 - 14)

\* @param input the line of text to check

\* @return whether synatax passes

\*/

public static boolean lateCheck**(**String input**){**

boolean retVal **=** **true;**

**if** **(**PatternMatching**.**visitIfStatOper**(**input**))** **{**

System**.**out**.**println**(**"\*\*\*" **+** input **+** "\*\*\* found if statement with operator"**);**

int left **=** input**.**length**()** **-** input**.**replace**(**"("**,** ""**).**length**();**

int right **=** input**.**length**()** **-** input**.**replace**(**")"**,** ""**).**length**();**

**if** **(**left **!=** right**)** **{**

System**.**err**.**println**(**"error code 10: invalid if statement (unequal parenthesis count)"**);**// error

//printStack(Thread.currentThread().getStackTrace()); // message

//System.exit(10);// quit

retVal **=** **false;**

**}**

/\*\*

\* this line is found to contain an if statement that contains an

\* operator we must check the left and right to side of the operator

\* to see id both are function or both are variable

\*/

input **=** input**.**replaceAll**(**"\\(\\)"**,** "(words)"**);**// this is a hack to work around empty function calls.

input **=** input**.**replaceAll**(**"else "**,** ""**);**

// String ifStatementOps =

// ("(\\w+)(\\w+\\(\\w\*?\\))?\\s?(=|>|>=|<|<=|!=)\\s?(\\w+)(\\w+\\(\\w\*?\\))?");

// Pattern ifStatOpCheck = Pattern.compile(ifStatementOps);

// Matcher matcher = ifStatOpCheck.matcher(input);

String**[]** groups **=** input**.**split**(**"[^\\w']+"**);**// split on space or non-words

// for tacking names and whether they are found

String arg1 **=** **null;**

String arg2 **=** **null;**

boolean found1 **=** **false;**

boolean found2 **=** **false;**

**if(**charCheck**(**input**)){**//check to see if there is a char operation

groups **=** input**.**split**(**"((?<=[^\\w']+)|(?=[^\\w']+))"**);**// split on space or non-words

ArrayList**<**String**>** tokens **=** **new** ArrayList**<**String**>(**Arrays**.**asList**(**groups**));**

tokens**.**removeAll**(**Arrays**.**asList**(**" "**,** "'"**,** "else"**,** "("**,** ")"**));**//clear any spaces or ' groups

//

// tokens.forEach(token -> {

// System.out.println(token);

// });

//System.out.println(charCheck(input));

int index **=** **-**1**;**//go through, clean up the tokens and combine

**for(**int i **=** 0**;** i **<** tokens**.**size**();** i**++){**

String newStr **=** tokens**.**get**(**i**)** **+** tokens**.**get**(**i**+**1**);**

**if((**tokens**.**get**(**i**)** **+** tokens**.**get**(**i**+**1**)).**matches**(**"(==|!=)"**)){**

tokens**.**set**(**i**,** newStr**);**

tokens**.**remove**(**i**+**1**);**

index **=** i**;**

**break;**

**}**

**}**

//System.out.println("-------------------------------------------------------------------------------");

//System.out.println(index);

//printList(tokens);

**if(**tokens**.**get**(**index**-**1**).**matches**(**"'\\w'"**)){**

System**.**out**.**println**(**"left hand side has the char"**);**

String rArg **=** tokens**.**get**(**index **+** 1**);**

**if(**rArg**.**equalsIgnoreCase**(**"null"**)){**//allow null for char or \*char

//return true

**}** **else** **{**//if not null

**if(!**mVariableIndex**.**containsKey**(**rArg**)** **&&** **!**mFunctionIndex**.**containsKey**(**rArg**)){**

System**.**err**.**println**(**"error code 10: invalid if statement (variable name not found for char "

**+** "comparison))"**);**

//printStack(Thread.currentThread().getStackTrace());

//System.exit(10);

//return false;

**}**

**}**

**}** **else** **{**

//System.out.println("right hand side has the char");

String lArg **=** tokens**.**get**(**index **-** 1**);**

**if(**lArg**.**equalsIgnoreCase**(**"null"**)){**//allow null for char or \*char

//return true

**}** **else** **{**//if not null

**if(!**mVariableIndex**.**containsKey**(**lArg**)** **&&** **!**mFunctionIndex**.**containsKey**(**lArg**)){**

System**.**err**.**println**(**"error code 10: invalid if statement (variable name not found for char "

**+** "comparison))"**);**

//printStack(Thread.currentThread().getStackTrace());

//System.exit(10);

//return false;

**}**

**}**

**}**

// for checking variable names

**}** **else** **if** **(**groups**.**length **==** 3**)** **{**// if groups size is 3, then the args are

// variables.

arg1 **=** groups**[**1**];**// 1st variable

arg2 **=** groups**[**2**];**// 2nd variable

// check for variable names

found1 **=** mVariableIndex**.**containsKey**(**arg1**);**

found2 **=** mVariableIndex**.**containsKey**(**arg2**);**

// if one or more of the variables are not found

**if** **(!**found1 **||** **!**found2**)** **{**

System**.**err**.**println**(**"error code 10: invalid if statement (variable names not found))"**);**

//printStack(Thread.currentThread().getStackTrace());

//System.exit(10);

**}**

**if(!**mVariableIndex**.**get**(**arg1**).**equals**(**mVariableIndex**.**get**(**arg2**))){**

System**.**err**.**println**(**"error code 10: invalid if statement (variables are of different types))"**);**

//printStack(Thread.currentThread().getStackTrace());

//System.exit(10);

**}**

**}** **else** **{**// this is for checking the function names, same steps above

arg1 **=** groups**[**1**];**

arg2 **=** groups**[**3**];**

String**[]** funNames **=** getFunctionDictionary**();**

found1 **=** mFunctionIndex**.**containsKey**(**arg1**);**

found2 **=** mFunctionIndex**.**containsKey**(**arg2**);**

**if** **(!**found1 **||** **!**found2**)** **{**

System**.**err**.**println**(**"error code 10: invalid if statement (function names not found))"**);**

//printStack(Thread.currentThread().getStackTrace());

//System.exit(10);

**}**

String return1 **=** mFunctionIndex**.**get**(**arg1**).**get**(**0**);**

String return2 **=** mFunctionIndex**.**get**(**arg1**).**get**(**0**);**

**if(!**return1**.**equals**(**return2**)){**

System**.**err**.**println**(**"error code 10: invalid if statement (function returns do not match))"**);**

//printStack(Thread.currentThread().getStackTrace());

//System.exit(10);

**}**

**}**

**}** **else** **if** **(**PatternMatching**.**visitIfStatFunc**(**input**))** **{**

/\*\*

\* for now, checking for a proper function will be ignored

\*/

int left **=** input**.**length**()** **-** input**.**replace**(**"("**,** ""**).**length**();**

int right **=** input**.**length**()** **-** input**.**replace**(**")"**,** ""**).**length**();**

**if** **(**left **!=** right**)** **{**

System**.**err**.**println**(**"error code 10: invalid if statement (unequal parenthesis count)"**);**// error

//printStack(Thread.currentThread().getStackTrace()); // message

//System.exit(10);// quit

retVal **=** **false;**

**}**

String**[]** groups **=** input**.**split**(**"[^\\w']+"**);**// split on space or

// non-words

String arg1 **=** groups**[**1**];**

String**[]** funcNames **=** getFunctionDictionary**();**// load function names

boolean match **=** **false;**

**for** **(**String s **:** funcNames**)** **{**// check to see if function exits

**if** **(**s**.**equals**(**arg1**))** **{**

match **=** **true;**

**break;**// stop when found

**}**

**}**

**if** **(!**match**)** **{**// if we didn't find the function name, error out

System**.**err**.**println**(**"error code 10: invalid if statement (function name not found))"**);**

//printStack(Thread.currentThread().getStackTrace());

//System.exit(10);

retVal **=** **false;**

**}**

**if** **(**mFunctionIndex**.**get**(**arg1**)** **!=** **null** **&&** **!**mFunctionIndex**.**get**(**arg1**).**get**(**0**).**equals**(**"bool"**))** **{**// check that the return type is correct

System**.**err**.**println**(**"error code 10: invalid if statement (function must return bool type))"**);**

//printStack(Thread.currentThread().getStackTrace());

//System.exit(10);

retVal **=** **false;**

**}**

**}** **else** **if** **(**PatternMatching**.**visitIfStatVar**(**input**))** **{**

int left **=** input**.**length**()** **-** input**.**replace**(**"("**,** ""**).**length**();**

int right **=** input**.**length**()** **-** input**.**replace**(**")"**,** ""**).**length**();**

**if** **(**left **!=** right**)** **{**

System**.**err**.**println**(**"error code 10: invalid if statement (unequal parenthesis count)"**);**// error

//printStack(Thread.currentThread().getStackTrace()); // message

//System.exit(10);// quit

retVal **=** **false;**

**}**

String**[]** groups **=** input**.**split**(**"[^\\w']+"**);**// split on space or

// non-words

String arg1 **=** groups**[**1**];**

String**[]** varNames **=** getVariableDictionary**();**

boolean match **=** **false;**

**for** **(**String s **:** varNames**)** **{**

**if** **(**s**.**equals**(**arg1**))** **{**

match **=** **true;**

**break;**

**}**

**}**

**if** **(!**match**)** **{**

System**.**err**.**println**(**"error code 10: invalid if statement (variable name not found))"**);**

//printStack(Thread.currentThread().getStackTrace());

//System.exit(10);

retVal **=** **false;**

**}**

**if** **(**mVariableIndex**.**get**(**arg1**)** **!=** "bool"**)** **{**

System**.**err**.**println**(**"error code 10: invalid if statement (function must return bool type))"**);**

//printStack(Thread.currentThread().getStackTrace());

//System.exit(10);

retVal **=** **false;**

**}**

**}** **else** **if** **(**PatternMatching**.**visitWhileStatVar**(**input**))** **{**

int left **=** input**.**length**()** **-** input**.**replace**(**"("**,** ""**).**length**();**

int right **=** input**.**length**()** **-** input**.**replace**(**")"**,** ""**).**length**();**

**if** **(**left **!=** right**)** **{**

System**.**err**.**println**(**"error code 10: invalid while statement (unequal parenthesis count)"**);**// error

//printStack(Thread.currentThread().getStackTrace()); // message

//System.exit(10);// quit

retVal **=** **false;**

**}**

String**[]** groups **=** input**.**split**(**"[^\\w']+"**);**// split on space or

// non-words

String arg1 **=** groups**[**1**];**

String**[]** varNames **=** getVariableDictionary**();**

boolean match **=** **false;**

**for** **(**String s **:** varNames**)** **{**

**if** **(**s**.**equals**(**arg1**))** **{**

match **=** **true;**

**break;**

**}**

**}**

**if** **(!**match**)** **{**

System**.**err**.**println**(**"error code 10: invalid while statement (variable name not found))"**);**

//printStack(Thread.currentThread().getStackTrace());

//System.exit(10);

retVal **=** **false;**

**}**

**if** **(**mVariableIndex**.**get**(**arg1**)** **!=** "bool"**)** **{**

System**.**err**.**println**(**"error code 10: invalid while statement (variable must be of type bool))"**);**

//printStack(Thread.currentThread().getStackTrace());

//System.exit(10);

retVal **=** **false;**

**}**

**}** **else** **if** **(**PatternMatching**.**visitWhileStatFunc**(**input**))** **{**

int left **=** input**.**length**()** **-** input**.**replace**(**"("**,** ""**).**length**();**

int right **=** input**.**length**()** **-** input**.**replace**(**")"**,** ""**).**length**();**

**if** **(**left **!=** right**)** **{**

System**.**err**.**println**(**"error code 10: invalid while statement (unequal parenthesis count)"**);**// error

//printStack(Thread.currentThread().getStackTrace()); // message

//System.exit(10);// quit

retVal **=** **false;**

**}**

input **=** input**.**replaceAll**(**"\\(\\)"**,** "(words)"**);**// this is a hack to

// work around empty

// function calls.

String**[]** groups **=** input**.**split**(**"[^\\w']+"**);**// split on space or

// non-words

String arg1 **=** groups**[**1**];**

String**[]** funcNames **=** getFunctionDictionary**();**

boolean match **=** **false;**

**for** **(**String s **:** funcNames**)** **{**

**if** **(**s**.**equals**(**arg1**))** **{**

match **=** **true;**

**break;**

**}**

**}**

**if** **(!**match**)** **{**

System**.**err**.**println**(**"error code 10: invalid while statement (function name not found))"**);**

//printStack(Thread.currentThread().getStackTrace());

//System.exit(10);

retVal **=** **false;**

**}**

**if** **(**mFunctionIndex**.**get**(**arg1**)** **!=** **null** **&&** **!**mFunctionIndex**.**get**(**arg1**).**get**(**0**).**equals**(**"bool"**))** **{**

System**.**err**.**println**(**"error code 10: invalid while statement (function must return bool type))"**);**

//printStack(Thread.currentThread().getStackTrace());

//System.exit(10);

retVal **=** **false;**

**}**

**}** **else** **if** **(**PatternMatching**.**visitWhileStatOper**(**input**))** **{**

System**.**out**.**println**(**"\*\*\*" **+** input **+** "\*\*\* found while statement with operator"**);**

int left **=** input**.**length**()** **-** input**.**replace**(**"("**,** ""**).**length**();**

int right **=** input**.**length**()** **-** input**.**replace**(**")"**,** ""**).**length**();**

**if** **(**left **!=** right**)** **{**

System**.**err**.**println**(**"error code 10: invalid while statement (unequal parenthesis count)"**);**// error

//printStack(Thread.currentThread().getStackTrace()); // message

//System.exit(10);// quit

retVal **=** **false;**

**}**

/\*\*

\* this line is found to contain a while statement that contains an

\* operator we must check the left and right to side of the operator

\* to see id both are function or both are variable

\*/

input **=** input**.**replaceAll**(**"\\(\\)"**,** "(words)"**);**// this is a hack to

// work around empty

// function calls.

String**[]** groups **=** input**.**split**(**"[^\\w']+"**);**// split on space or

// non-words

// for tacking names and whether they are found

String arg1 **=** **null;**

String arg2 **=** **null;**

boolean found1 **=** **false;**

boolean found2 **=** **false;**

**if(**charCheck**(**input**)){**

input **=** input**.**replaceAll**(**"\\((\\w+)\\)"**,** ""**);**

groups **=** input**.**split**(**"[^\\w']+"**);**// split on space or non-words

//printArr(groups);

System**.**out**.**println**(**input**);**

String arg **=** ""**;**

**if(**groups**[**1**].**matches**(**"'\\S'"**)){**

arg **=** groups**[**2**];**

System**.**out**.**println**(**"left side is the char"**);**

**}** **else** **{**

arg **=** groups**[**1**];**

System**.**out**.**println**(**"right side is the char"**);**

**}**

**if(!**arg**.**equalsIgnoreCase**(**"null"**)){**//pass if comparing to null

**if(!**mFunctionIndex**.**containsKey**(**arg**)** **&&** **!**mVariableIndex**.**containsKey**(**arg**)){**

//not a valid variable or function

System**.**err**.**println**(**"error code 11: invalid while statement (function / variables not found)"**);**// error

//printStack(Thread.currentThread().getStackTrace()); // message

//System.exit(10);// quit

**}**

//check arg values

**if(**mFunctionIndex**.**containsKey**(**arg**)){**

**if(**mFunctionIndex**.**get**(**arg1**)** **!=** **null** **&&** **!**mFunctionIndex**.**get**(**arg**).**get**(**0**).**equals**(**"char"**)){**

System**.**err**.**println**(**"error code 11: invalid while statement (invalid char \\ func comparison)"**);**// error

//printStack(Thread.currentThread().getStackTrace()); // message

//System.exit(10);// quit

**}**

**}** **else** **{**

**if(!**mVariableIndex**.**get**(**arg**).**equals**(**"char"**)){**

System**.**err**.**println**(**"error code 11: invalid while statement (invalid char \\ var comparison)"**);**// error

//printStack(Thread.currentThread().getStackTrace()); // message

//System.exit(10);// quit

**}**

**}**

**}**

// for checking variable names

**}** **else** **if** **(**groups**.**length **==** 3**)** **{**// if groups size is 3, then the args are

// variables.

arg1 **=** groups**[**1**];**// 1st variable

arg2 **=** groups**[**2**];**// 2nd variable

found1 **=** mVariableIndex**.**containsKey**(**arg1**);**

found2 **=** mVariableIndex**.**containsKey**(**arg2**);**

// if one or more of the variables are not found

**if** **(!**found1 **||** **!**found2**)** **{**

System**.**err**.**println**(**"error code 11: invalid while statement (variable names not found))"**);**

//printStack(Thread.currentThread().getStackTrace());

//System.exit(10);

retVal **=** **false;**

**}**

String return1 **=** mVariableIndex**.**get**(**arg1**);**

String return2 **=** mVariableIndex**.**get**(**arg2**);**

**if(!**return1**.**equals**(**return2**)){**

System**.**err**.**println**(**"error code 11: invalid while statement (variable type mismatch))"**);**

//printStack(Thread.currentThread().getStackTrace());

//System.exit(10);

retVal **=** **false;**

**}**

**}** **else** **{**// this is for checking the function names, same steps above

arg1 **=** groups**[**1**];**

arg2 **=** groups**[**3**];**

found1 **=** mFunctionIndex**.**containsKey**(**arg1**);**

found2 **=** mFunctionIndex**.**containsKey**(**arg2**);**

**if** **(!**found1 **||** **!**found2**)** **{**

System**.**err**.**println**(**"error code 10: invalid while statement (function names not found))"**);**

//printStack(Thread.currentThread().getStackTrace());

//System.exit(10);

retVal **=** **false;**

**}**

String return1 **=** mFunctionIndex**.**get**(**arg1**).**get**(**0**);**

String return2 **=** mFunctionIndex**.**get**(**arg2**).**get**(**0**);**

**if(!**return1**.**equals**(**return2**)){**

System**.**err**.**println**(**"error code 11: invalid while statement (function return type mismatch))"**);**

//printStack(Thread.currentThread().getStackTrace());

//System.exit(10);

retVal **=** **false;**

**}**

**}**

**}** **return** retVal**;**

**}**

/\*\*

\* Runs the late syntax checks for 14-16

\* @param input the line of code to check

\* @return true or false for passing syntax

\*/

static public boolean lateSyntaxCheck**(**String input**){**

boolean retVal **=** **true;**

**if(**PatternMatching**.**visitCharArray**(**input**))** **{**//check the String[1] type call

retVal **=** checkArr**(**input**);**

**}**

**if(**PatternMatching**.**visitAssignmentCheck**(**input**))** **{**//check for assignement call

retVal **=** checkAssignCall**(**input**);**

**}**

**if(**PatternMatching**.**visitMathCheck**(**input**)){**//check for a math operation call

retVal **=** mathOpCheck**(**input**);**

**}**

**if(**PatternMatching**.**visitLogicalCheck**(**input**)){**//check for the logical operation call

retVal **=** logicalOpCheck**(**input**);**

**}**

**if(**PatternMatching**.**visitCompCheck**(**input**)){**//check syntax of calls

retVal **=** compOpCheck**(**input**);**

**}**

**if(**PatternMatching**.**visitEqualCheck**(**input**))** **{**//check for call of == or !=

retVal **=** equalityCheck**(**input**);**

**}**

**if(**PatternMatching**.**visitAbsValCheck**(**input**)){**//checks the absolute value call

retVal **=** absValCheck**(**input**);**

**}**

**if(**PatternMatching**.**visitNegCheck**(**input**)){**//checks the syntax of negations

retVal **=** negCheck**(**input**);**

**}**

**return** retVal**;**

**}**

/\*\*

\* Performs syntax check on a negation statement

\* @param in the string that holds the negation statment

\* @return if the call is made correctly

\*/

public static boolean negCheck**(**String in**){**

boolean retVal **=** **false;**

String input **=** in**.**replaceAll**(**"\\((\\S\*)\\)"**,** ""**);**

System**.**out**.**println**(**"Negation found: " **+** in**);**

String **[]**groups **=** input**.**split**(**"((?<=[^\\w']+)|(?=[^\\w']+))"**);**// split on space or non-words

//printArr(groups);

ArrayList**<**String**>** tokens **=** **new** ArrayList**<**String**>(**Arrays**.**asList**(**groups**));**//load into arraylist

tokens**.**removeAll**(**Arrays**.**asList**(**" "**));**//remove spaces

//printList(tokens);

int index **=** **-**1**;**//find the operator in token list

**for(**int i **=** 0**;** i **<** tokens**.**size**();** i**++){**//find and combine && or ||

String op **=** tokens**.**get**(**i**);**

**if(**op**.**equals**(**"!"**)){**

index **=** i **+** 1**;**

**break;**

**}**

**}**

String arg **=** tokens**.**get**(**index**);**

/\*\*

\* lets check THE arg

\*/

**if(!**mVariableIndex**.**containsKey**(**arg**)** **&&** **!**mFunctionIndex**.**containsKey**(**arg**)){**

System**.**err**.**println**(**"error code 15: could not find variable / function name: " **+** arg**);**

printStack**(**Thread**.**currentThread**().**getStackTrace**());**

//System.exit(15);

retVal **=** **false;**

**}**

**if(**mVariableIndex**.**containsKey**(**arg**)){**//if a variable

**if(!**mVariableIndex**.**get**(**arg**).**equals**(**"bool"**)){**

System**.**err**.**println**(**"error code 15: variable is not an bool: " **+** arg**);**

printStack**(**Thread**.**currentThread**().**getStackTrace**());**

//System.exit(15);

retVal **=** **false;**

**}**

**}** **else** **{**//if a function

**if(**mFunctionIndex**.**get**(**arg**)** **!=** **null){**

**if(!**mFunctionIndex**.**get**(**arg**).**get**(**0**).**equals**(**"bool"**)){**

System**.**err**.**println**(**"error code 15: function return is not an bool: " **+** arg**);**

printStack**(**Thread**.**currentThread**().**getStackTrace**());**

//System.exit(15);

retVal **=** **false;**

**}**

**}**

**}**

**return** retVal**;**

**}**

/\*\*

\* Checks the absolute value decleration.

\* @param in the absolute value call to check

\* @return whether the syntax is good.

\*/

public static boolean absValCheck**(**String in**){**

boolean retVal **=** **true;**

String input **=** in**.**replaceAll**(**"\\((\\S\*)\\)"**,** ""**);**

System**.**out**.**println**(**"found absolute value decleration: " **+** in**);**

String **[]**groups **=** input**.**split**(**"((?<=[^\\w']+)|(?=[^\\w']+))"**);**// split on space or non-words

//printArr(groups);

ArrayList**<**String**>** tokens **=** **new** ArrayList**<**String**>(**Arrays**.**asList**(**groups**));**//load into arraylist

tokens**.**removeAll**(**Arrays**.**asList**(**" "**));**//remove spaces

//printList(tokens);

int index **=** **-**1**;**//find the operator in token list

**for(**int i **=** 0**;** i **<** tokens**.**size**();** i**++){**//find and combine && or ||

String op **=** tokens**.**get**(**i**);**

**if(**op**.**equals**(**"|"**)){**

index **=** i **+** 1**;**

**break;**

**}**

**}**

String arg **=** tokens**.**get**(**index**);**

/\*\*

\* lets check THE arg

\*/

**if(!**mVariableIndex**.**containsKey**(**arg**)** **&&** **!**mFunctionIndex**.**containsKey**(**arg**)** **&&**

**!**mPointerIndex**.**containsKey**(**arg**)){**

System**.**err**.**println**(**"error code 15: could not find variable / function name: " **+** arg**);**

printStack**(**Thread**.**currentThread**().**getStackTrace**());**

//System.exit(15);

retVal **=** **false;**

**}**

**if(**mVariableIndex**.**containsKey**(**arg**)){**//if a variable

**if(!**mVariableIndex**.**get**(**arg**).**equals**(**"int"**)){**

System**.**err**.**println**(**"error code 15: variable is not an int: " **+** arg**);**

printStack**(**Thread**.**currentThread**().**getStackTrace**());**

//System.exit(15);

retVal **=** **false;**

**}**

**}** **else** **if(**mPointerIndex**.**containsKey**(**arg**)){**//if int pointer

**if(!**mPointerIndex**.**get**(**arg**).**equals**(**"int"**)){**

System**.**err**.**println**(**"error code 15: variable is not an int pointer: " **+** arg**);**

printStack**(**Thread**.**currentThread**().**getStackTrace**());**

//System.exit(15);

retVal **=** **false;**

**}**

**}** **else** **{**//if a function

**if(**mFunctionIndex**.**get**(**arg**)** **!=** **null){**

**if(!**mFunctionIndex**.**get**(**arg**).**get**(**0**).**equals**(**"int"**)){**

System**.**err**.**println**(**"error code 15: function return is not an int: " **+** arg**);**

printStack**(**Thread**.**currentThread**().**getStackTrace**());**

//System.exit(15);

retVal **=** **false;**

**}**

**}**

**}**

**return** retVal**;**

**}**

/\*\*

\* Check the eqaulity statement for the same types

\* @param in The string to check

\* @return whether the type check passes

\*/

public static boolean equalityCheck**(**String in**){**

boolean retVal **=** **true;**

String input **=** in**.**replaceAll**(**"\\((\\S\*)\\)"**,** ""**);**

System**.**out**.**println**(**"equality statement found: " **+** in**);**

String **[]**groups **=** input**.**split**(**"((?<=[^\\w']+)|(?=[^\\w']+))"**);**// split on space or non-words

//printArr(groups);

ArrayList**<**String**>** tokens **=** **new** ArrayList**<**String**>(**Arrays**.**asList**(**groups**));**//load into arraylist

tokens**.**removeAll**(**Arrays**.**asList**(**" "**));**//remove spaces

//printList(tokens);

int index **=** **-**1**;**//find the operator in token list

**for(**int i **=** 0**;** i **<** tokens**.**size**();** i**++){**//find and combine && or ||

String op **=** tokens**.**get**(**i**);**

**if(**op**.**matches**(**"(!|=)"**)){**

index **=** i**;**

tokens**.**set**(**i**,** op **+** tokens**.**get**(**i **+** 1**));**

tokens**.**remove**(**i**+**1**);**

**}**

**}**

String leftArg **=** tokens**.**get**(**index **-** 1**);**

String rightArg **=** tokens**.**get**(**index **+** 1**);**

String return1 **=** ""**;**

String return2 **=** ""**;**

/\*\*

\* lets check the left arg

\*/

**if(!**mVariableIndex**.**containsKey**(**leftArg**)** **&&** **!**mFunctionIndex**.**containsKey**(**leftArg**)){**

System**.**err**.**println**(**"error code 15: could not find variable / function name: " **+** leftArg**);**

printStack**(**Thread**.**currentThread**().**getStackTrace**());**

//System.exit(15);

retVal **=** **false;**

**}**

**if(**mVariableIndex**.**containsKey**(**leftArg**)){**

return1 **=** mVariableIndex**.**get**(**leftArg**);**

**}** **else** **{**

**if(**mFunctionIndex**.**get**(**rightArg**)** **!=** **null){**

return1 **=** mFunctionIndex**.**get**(**rightArg**).**get**(**0**);**

**}**

**}**

/\*\*

\* lefts check the right arg

\*/

**if(!**mVariableIndex**.**containsKey**(**rightArg**)** **&&** **!**mFunctionIndex**.**containsKey**(**rightArg**)){**

System**.**err**.**println**(**"error code 15: could not find variable / function name: " **+** rightArg**);**

printStack**(**Thread**.**currentThread**().**getStackTrace**());**

//System.exit(15);

retVal **=** **false;**

**}**

**if(**mVariableIndex**.**containsKey**(**rightArg**)){**

return2 **=** mVariableIndex**.**get**(**rightArg**);**

**}** **else** **{**

**if(**mFunctionIndex**.**get**(**rightArg**)** **!=** **null){**

return2 **=** mFunctionIndex**.**get**(**rightArg**).**get**(**0**);**

**}**

**}**

**if(**return1 **!=** return2**){**

System**.**err**.**println**(**"error code 15: " **+** leftArg **+** " not same type as " **+** rightArg**);**

printStack**(**Thread**.**currentThread**().**getStackTrace**());**

//System.exit(15);

retVal **=** **false;**

**}**

**return** retVal**;**

**}**

/\*\*

\* checks the string for comparrison between int types

\* @param in the string to check

\* @return whether the comparision is syntaticly valid

\*/

public static boolean compOpCheck**(**String in**){**

boolean retVal **=** **true;**

String input **=** in**.**replaceAll**(**"\\((\\S\*)\\)"**,** ""**);**

System**.**out**.**println**(**"found comparison operation: " **+** in**);**

String **[]**groups **=** input**.**split**(**"((?<=[^\\w']+)|(?=[^\\w']+))"**);**// split on space or non-words

**if(**groups**[**0**].**equals**(**"#"**))**//skip comments and imports

**return** **true;**

//printArr(groups);

ArrayList**<**String**>** tokens **=** **new** ArrayList**<**String**>(**Arrays**.**asList**(**groups**));**//load into arraylist

tokens**.**removeAll**(**Arrays**.**asList**(**" "**));**//remove spaces

//printList(tokens);

int index **=** **-**1**;**//find the operator in token list

**for(**int i **=** 0**;** i **<** tokens**.**size**();** i**++){**//find and combine && or ||

String op **=** tokens**.**get**(**i**);**

**if(**op**.**matches**(**"(<|<=|>|>=)"**)){**

index **=** i**;**

**if(**tokens**.**get**(**i **+** 1**).**equals**(**"="**)){**

tokens**.**set**(**i**,** **(**op **+** tokens**.**get**(**i**+**1**)));**

tokens**.**remove**(**i**+**1**);**

**}**

**break;**

**}**

**}**

String leftArg **=** tokens**.**get**(**index **-** 1**);**

String rightArg **=** tokens**.**get**(**index **+** 1**);**

/\*\*

\* lets check the left arg

\*/

**if(!**mVariableIndex**.**containsKey**(**leftArg**)** **&&** **!**mFunctionIndex**.**containsKey**(**leftArg**)){**

System**.**err**.**println**(**"error code 15: could not find variable / function name: " **+** leftArg**);**

printStack**(**Thread**.**currentThread**().**getStackTrace**());**

retVal **=** **false;**

//System.exit(15);

**}**

**if(**mVariableIndex**.**containsKey**(**leftArg**)){**

**if(!**mVariableIndex**.**get**(**leftArg**).**equals**(**"int"**)){**

System**.**err**.**println**(**"error code 15: variable type isn't int: " **+** leftArg**);**

printStack**(**Thread**.**currentThread**().**getStackTrace**());**

retVal **=** **false;**

//System.exit(15);

**}**

**}** **else** **{**

**if(**mFunctionIndex**.**get**(**leftArg**)** **!=** **null** **&&!**mFunctionIndex**.**get**(**leftArg**).**get**(**0**).**equals**(**"int"**)){**

System**.**err**.**println**(**"error code 15: function return type isn't int: " **+** leftArg**);**

printStack**(**Thread**.**currentThread**().**getStackTrace**());**

retVal **=** **false;**

//System.exit(15);

**}**

**}**

/\*\*

\* lefts check the right arg

\*/

**if(!**mVariableIndex**.**containsKey**(**rightArg**)** **&&** **!**mFunctionIndex**.**containsKey**(**rightArg**)){**

System**.**err**.**println**(**"error code 15: could not find variable / function name: " **+** rightArg**);**

printStack**(**Thread**.**currentThread**().**getStackTrace**());**

retVal **=** **false;**

//System.exit(15);

**}**

**if(**mVariableIndex**.**containsKey**(**rightArg**)){**

**if(!**mVariableIndex**.**get**(**rightArg**).**equals**(**"int"**)){**

System**.**err**.**println**(**"error code 15: variable type isn't int: " **+** rightArg**);**

printStack**(**Thread**.**currentThread**().**getStackTrace**());**

retVal **=** **false;**

//System.exit(15);

**}**

**}** **else** **{**

**if(**mFunctionIndex**.**get**(**rightArg**)** **!=** **null** **&&** **!**mFunctionIndex**.**get**(**rightArg**).**get**(**0**).**equals**(**"int"**)){**

System**.**err**.**println**(**"error code 15: function return type isn't int: " **+** rightArg**);**

printStack**(**Thread**.**currentThread**().**getStackTrace**());**

retVal **=** **false;**

//System.exit(15);

**}**

**}**

**return** retVal**;**

**}**

/\*\*

\* Checks if the logical operator call passes<br>

\*

\* @param in

\* @return

\*/

public static boolean logicalOpCheck**(**String in**){**

boolean retVal **=** **true;**

String input **=** in**.**replaceAll**(**"\\((\\S\*)\\)"**,** ""**);**//removes function params

System**.**out**.**println**(**"comparison operator found: " **+** input**);**

String **[]**groups **=** input**.**split**(**"((?<=[^\\w']+)|(?=[^\\w']+))"**);**// split on space or non-words

//printArr(groups);

ArrayList**<**String**>** tokens **=** **new** ArrayList**<**String**>(**Arrays**.**asList**(**groups**));**//load into arraylist

tokens**.**removeAll**(**Arrays**.**asList**(**" "**));**//remove spaces

//printList(tokens);

int index **=** **-**1**;**//find the operator in token list

**for(**int i **=** 0**;** i **<** tokens**.**size**();** i**++){**//find and combine && or ||

String op **=** tokens**.**get**(**i**);**

**if(**op**.**matches**(**"(\\&|\\|)"**)){**

index **=** i**;**

tokens**.**set**(**i**,** **(**op **+** tokens**.**get**(**i**+**1**)));**

tokens**.**remove**(**i**+**1**);**

**break;**

**}**

**}**

String leftArg **=** tokens**.**get**(**index **-** 1**);**

String rightArg **=** tokens**.**get**(**index **+** 1**);**

/\*\*

\* lets check the left arg

\*/

**if(!**mVariableIndex**.**containsKey**(**leftArg**)** **&&** **!**mFunctionIndex**.**containsKey**(**leftArg**)){**

System**.**err**.**println**(**"error code 15: could not find variable / function name: " **+** leftArg**);**

printStack**(**Thread**.**currentThread**().**getStackTrace**());**

//System.exit(15);

retVal **=** **false;**

**}**

**if(**mVariableIndex**.**containsKey**(**leftArg**)){**

**if(!**mVariableIndex**.**get**(**leftArg**).**equals**(**"bool"**)){**

System**.**err**.**println**(**"error code 15: variable type isn't bool: " **+** leftArg**);**

printStack**(**Thread**.**currentThread**().**getStackTrace**());**

//System.exit(15);

retVal **=** **false;**

**}**

**}** **else** **{**

**if(**mFunctionIndex**.**get**(**leftArg**)** **!=** **null** **&&!**mFunctionIndex**.**get**(**leftArg**).**get**(**0**).**equals**(**"bool"**)){**

System**.**err**.**println**(**"error code 15: function return type isn't bool: " **+** leftArg**);**

printStack**(**Thread**.**currentThread**().**getStackTrace**());**

//System.exit(15);

retVal **=** **false;**

**}**

**}**

/\*\*

\* lefts check the right arg

\*/

**if(!**mVariableIndex**.**containsKey**(**rightArg**)** **&&** **!**mFunctionIndex**.**containsKey**(**rightArg**)){**

System**.**err**.**println**(**"error code 15: could not find variable / function name: " **+** rightArg**);**

printStack**(**Thread**.**currentThread**().**getStackTrace**());**

//System.exit(15);

retVal **=** **false;**

**}**

**if(**mVariableIndex**.**containsKey**(**rightArg**)){**

**if(!**mVariableIndex**.**get**(**rightArg**).**equals**(**"bool"**)){**

System**.**err**.**println**(**"error code 15: variable type isn't bool: " **+** rightArg**);**

printStack**(**Thread**.**currentThread**().**getStackTrace**());**

//System.exit(15);

retVal **=** **false;**

**}**

**}** **else** **{**

**if(**mFunctionIndex**.**get**(**rightArg**)** **!=** **null** **&&** **!**mFunctionIndex**.**get**(**rightArg**).**get**(**0**).**equals**(**"bool"**)){**

System**.**err**.**println**(**"error code 15: function return type isn't int: " **+** rightArg**);**

printStack**(**Thread**.**currentThread**().**getStackTrace**());**

//System.exit(15);

retVal **=** **false;**

**}**

**}**

**return** retVal**;**

**}**

/\*\*

\* runs check to make sure that any math call has arguements as int<br>

\* For arithmetic operations (+,-,\*,/), both operands must be integer<br>

\* @param in the string that contains the operation

\* @return whether the check works

\*/

public static boolean mathOpCheck**(**String in**){**

System**.**out**.**println**(**"found math operation: " **+** in**);**

boolean retVal **=** **true;**

String input **=** in**.**replaceAll**(**"\\((\\S\*)\\)"**,** ""**);**

String **[]**groups **=** input**.**split**(**"((?<=[^\\w']+)|(?=[^\\w']+))"**);**// split on space or non-words

//printArr(groups);

ArrayList**<**String**>** tokens **=** **new** ArrayList**<**String**>(**Arrays**.**asList**(**groups**));**

tokens**.**removeAll**(**Arrays**.**asList**(**" "**));**

//printList(tokens);

int index **=** **-**1**;**

**for(**int i **=** 0**;** i **<** tokens**.**size**();** i**++){**

**if(**tokens**.**get**(**i**).**matches**(**"(\\+|\\-|\\\*|\\/)"**)){**

index **=** i**;**

**break;**

**}**

**}**

String oper **=** tokens**.**get**(**index**);**

String leftArg **=** tokens**.**get**(**index **-** 1**);**

String rightArg **=** tokens**.**get**(**index **+** 1**);**

/\*\*

\* lets check the left arg

\*/

**if(!**leftArg**.**matches**(**"\\d+"**)){**//skip if the left side is digits

**if(!**mVariableIndex**.**containsKey**(**leftArg**)** **&&** **!**mFunctionIndex**.**containsKey**(**leftArg**)){**

System**.**err**.**println**(**"error code 15: could not find variable / function name: " **+** leftArg**);**

//printStack(Thread.currentThread().getStackTrace());

//System.exit(15);

**}**

**if(**mVariableIndex**.**containsKey**(**leftArg**)){**

**if(!**mVariableIndex**.**get**(**leftArg**).**equals**(**"int"**)){**

System**.**err**.**println**(**"error code 15: variable type isn't int: " **+** leftArg**);**

//printStack(Thread.currentThread().getStackTrace());

//System.exit(15);

**}**

**}** **else** **if(**mPointerIndex**.**containsKey**(**leftArg**))** **{**

**if(!**mPointerIndex**.**get**(**leftArg**).**equals**(**"int"**)){**

System**.**err**.**println**(**"error code 15: variable type isn't int pointer: " **+** leftArg**);**

//printStack(Thread.currentThread().getStackTrace());

//System.exit(15);

**}** **else** **if(**oper**.**equals**(**"/"**)** **||** oper**.**equals**(**"\*"**)){**

System**.**err**.**println**(**"error code 16: pointer type can only do + or -: " **+** leftArg**);**

**}**

**}** **else** **{**

**if(**mFunctionIndex**.**get**(**leftArg**)** **!=** **null** **&&!**mFunctionIndex**.**get**(**leftArg**).**get**(**0**).**equals**(**"int"**)){**

System**.**err**.**println**(**"error code 15: function return type isn't int: " **+** leftArg**);**

//printStack(Thread.currentThread().getStackTrace());

//System.exit(15);

**}**

**}**

**}**

/\*\*

\* lefts check the right arg

\*/

**if(!**rightArg**.**matches**(**"\\d+"**)){**//skip if the left side is digits

**if(!**mVariableIndex**.**containsKey**(**rightArg**)** **&&** **!**mFunctionIndex**.**containsKey**(**rightArg**)){**

System**.**err**.**println**(**"error code 15: could not find variable / function name: " **+** rightArg**);**

//printStack(Thread.currentThread().getStackTrace());

//System.exit(15);

**}**

**if(**mVariableIndex**.**containsKey**(**rightArg**)){**

**if(!**mVariableIndex**.**get**(**rightArg**).**equals**(**"int"**)){**

System**.**err**.**println**(**"error code 15: variable type isn't int: " **+** rightArg**);**

//printStack(Thread.currentThread().getStackTrace());

//System.exit(15);

**}**

**}** **else** **{**

**if(**mFunctionIndex**.**get**(**rightArg**)** **!=** **null** **&&** **!**mFunctionIndex**.**get**(**rightArg**).**get**(**0**).**equals**(**"int"**)){**

System**.**err**.**println**(**"error code 15: function return type isn't int: " **+** rightArg**);**

//printStack(Thread.currentThread().getStackTrace());

//System.exit(15);

**}**

**}**

**}**

**return** retVal**;**

**}**

/\*\*

\* Check if the given string contains a char decleration ie 't'

\* @param s the string to check

\* @return boolean of whether there is a char found

\*/

public static boolean charCheck**(**String s**){**

String regex **=** "(\\S)\*\\'{1}\\S{1}\\'{1}(\\S)\*"**;**

Pattern cCheck **=** Pattern**.**compile**(**regex**);**

**return** cCheck**.**matcher**(**s**).**find**();**

**}**

/\*\*

\* This function checks if the given array call for two different

\* requirements<br>

\* 1 - array index is numeric(12)<br>

\* 2 - the variable being indexed is a String(13)<br>

\*

\* @param input

\* - Line of text that contains the array index call

\* @return always returns true, if a condition is violated, the program

\* exits

\*/

public static boolean checkArr**(**String input**)** **{**

// split on non-word chars, use lookahead to keep delimiters

String**[]** groups **=** input**.**split**(**"((?<=[^\\w']+)|(?=[^\\w']+))"**);**

ArrayList**<**String**>** tokens **=** **new** ArrayList**<**String**>(**Arrays**.**asList**(**groups**));**

**while** **(**tokens**.**remove**(**" "**)** **==** **true)** **{**

**}**

**;**// remove all space chars as tokens

int left **=** 0**;** // index of the left brace

boolean stop **=** **false;**

**while** **(!**stop **&&** left **<** tokens**.**size**())** **{**// iterate through loop looking

// for [

**if** **(**tokens**.**get**(**left**).**equals**(**"["**))** **{**// if found

**if** **(!**tokens**.**get**(**left **+** 2**).**equals**(**"]"**))** **{**// check left + 2 is ]

System**.**err**.**println**(**"error code 12: invalid array index call"**);**// error

//printStack(Thread.currentThread().getStackTrace()); // message

//System.exit(12);// quit

**return** **false;**

**}**

stop **=** **true;**// found [

**}** **else** **{**

left**++;**// keep looking

**}**

**}**

// check if we found [, exit if not

**if** **(**left **==** tokens**.**size**())** **{**

System**.**err**.**println**(**"error code 12: invalid array index call"**);**// error

//printStack(Thread.currentThread().getStackTrace()); // message

//System.exit(12);// quit

**return** **false;**

**}**

// check if the token between [ and ] is numeric

**if** **(!**tokens**.**get**(**left **+** 1**).**matches**(**"\\d+"**))** **{**

System**.**err**.**println**(**"error code 12: invalid array index call (index is non-numeric)"**);**// error

//printStack(Thread.currentThread().getStackTrace()); // message

//System.exit(12);// quit

**return** **false;**

**}**

String**[]** varNames **=** getVariableDictionary**();**// get variable names

boolean found **=** **false;**// this is for variable checking

String arg **=** tokens**.**get**(**left **-** 1**);**// variable name will immediately

// preceed [

// search variables for arg

**for** **(**String s **:** varNames**)** **{**

**if** **(**arg**.**equals**(**s**))**

found **=** **true;**

**}**

**if** **(!**found**)** **{**// if variable is not found, exit

System**.**err**.**println**(**"error code 12: invalid array index call " **+** arg **+** " not found"**);**// error

//printStack(Thread.currentThread().getStackTrace()); // message

//System.exit(13);// quit

**return** **false;**

**}**

**if** **(!**mVariableIndex**.**get**(**arg**).**equals**(**"String"**))** **{**// variable is found,

// but not a String

System**.**err**.**println**(**"error code 13: invalid array index call " **+** arg **+** " not of type String"**);**// error

//printStack(Thread.currentThread().getStackTrace()); // message

//System.exit(13);// quit

**return** **false;**

**}**

**return** **true;**// if the program is running by this point, the array call

// worked.

**}**

/\*\*

\* Takes assign call and checks that the left and right side return the correct types

\* @param input

\*/

public static boolean checkAssignCall**(**String input**)** **{**

ArrayList**<**String**>** tokens **=** **new** ArrayList**<>(**Arrays**.**asList**(**input**.**split**(**"((?<=[^\\w']+)|(?=[^\\w']+))"**)));**

**while** **(**tokens**.**contains**(**" "**))** **{**

tokens**.**remove**(**" "**);**

**}**

int index **=** tokens**.**indexOf**(**"="**);**

String leftArg **=** **null;**

String leftType **=** **null;**

String rightArg **=** **null;**

String rightType **=** **null;**

leftArg **=** tokens**.**get**(**index **-**1**);**

rightArg **=** tokens**.**get**(**index**+**1**);**

**if(**rightArg**.**equals**(**"NULL"**)** **&&** **(**mVariableIndex**.**containsKey**(**leftArg**))){**

**if(!(**mVariableIndex**.**get**(**leftArg**).**equals**(**"char"**)** **||**

mPointerIndex**.**get**(**leftArg**).**equals**(**"int"**))){**

System**.**err**.**println**(**"error code 14: Invalid function: " **+** rightArg**);**// error

//printStack(Thread.currentThread().getStackTrace()); // message

//System.exit(14);// quit

**return** **false;**

**}**

**}**

**if(**tokens**.**contains**(**"+"**)** **||** tokens**.**contains**(**"-"**)){**

**}**

**if** **(**tokens**.**contains**(**"("**))** **{**// if this is a function call

rightArg **=** tokens**.**get**(**index **+** 1**);**

**if** **(!**mFunctionIndex**.**containsKey**(**rightArg**))** **{**// check the function

// exists

System**.**err**.**println**(**"error code 14: Invalid function: " **+** rightArg**);**// error

//printStack(Thread.currentThread().getStackTrace()); // message

//System.exit(14);// quit

**return** **false;**

**}**

rightType **=** mFunctionIndex**.**get**(**rightArg**).**get**(**0**);**

**}** **else** **{**// not a function, check the variable

rightArg **=** tokens**.**get**(**index **+** 1**);**

**if** **(!**mVariableIndex**.**containsKey**(**rightArg**))** **{**// check variable exists

System**.**err**.**println**(**"error code 14: Invalid variable: " **+** rightArg**);**// error

//printStack(Thread.currentThread().getStackTrace()); // message

//System.exit(14);// quit

**return** **false;**

**}**

rightType **=** mVariableIndex**.**get**(**rightArg**);**

**}**

leftArg **=** tokens**.**get**(**index **-** 1**);**

**if** **(!**mVariableIndex**.**containsKey**(**leftArg**))** **{**// check left hand variable

// exists

System**.**err**.**println**(**"error code 14: invalid variable: " **+** leftArg**);**// error

//printStack(Thread.currentThread().getStackTrace()); // message

//System.exit(14);// quit

**return** **false;**

**}** **else** **{**

leftType **=** mVariableIndex**.**get**(**leftArg**);**

**}**

**if** **(!**leftType**.**equals**(**rightType**))** **{**

System**.**err**.**println**(**

"error code 14: Left side does not match the right side: " **+** leftType **+** " != " **+** rightType**);**// error

//printStack(Thread.currentThread().getStackTrace()); // message

//System.exit(14);// quit

**return** **false;**

**}**

**return** **true;** //if this function has not retruned false yet, the statement is good

**}**

/\*\*

\*Error prints the stack leading up to the problem

\* @param s the StackTraceElement leading to the problem

\*/

public static void printStack**(**StackTraceElement**[]** s**){**

**for(**int i **=** 0**;** i **<** s**.**length**;** i**++){**

System**.**err**.**println**(**s**[**i**]);**

**}**

**}**

**}**

package typeCheck**;**

**import** java**.**util**.**regex**.**Pattern**;**

/\*\*

\* finds regex matches

\* Final version

\* @author James Chavis

\* @author Giang Truong

\*

\*/

public class PatternMatching **{**

// a single char lower or upper case

private static final String c **=** "[a-zA-z]"**;**

// a number between 0-9

private static final String n **=** "[0-9]"**;**

// operator +/-/\*/ /

private static final Pattern patternOperator **=** Pattern**.**compile**(**"(^)+($)|(^)-($)|(^)\*($)|(^)/($)"**);**

// true false boolean

private static final Pattern patternBoolean **=** Pattern**.**compile**(**"true|false"**);**

// left parathesis (

private static final Pattern patternLParen **=** Pattern**.**compile**(**"(^)[(]($)"**);**

// right parathesis )

private static final Pattern patternRParen **=** Pattern**.**compile**(**"(^)[)]($)"**);**

// comparasion

private static final Pattern patternComparasion **=** Pattern**.**compile**(**"<|>|<=|>=|=="**);**

//assignment

private static final Pattern patternAssignment **=** Pattern**.**compile**(**"="**);**

// and or not

private static final Pattern patternChoice **=** Pattern**.**compile**(**"&&|or|!"**);**

// conditional

private static final Pattern patternCondition **=** Pattern**.**compile**(**"if|else if|while|do|for"**);**

private static final Pattern patternElse**=** Pattern**.**compile**(**"else"**);**

private static final Pattern patternBreak**=** Pattern**.**compile**(**"break"**);**

private static final Pattern patternCont**=** Pattern**.**compile**(**"continue"**);**

// end of file

private static final Pattern patternEOF**=** Pattern**.**compile**(**"EOF"**);**

//private static final String patternPrimitive = ("int|double|char|float|long|short|bool");

private static final Pattern primitive **=** Pattern**.**compile**(**"(^)int($)|(^)double($)|(^)char($)|(^)float($)|(^)long($)|(^)short($)|(^)bool($)"**);**

private static final String patternPrimitiveArray **=** **(**"(^)int[]($)|(^)double[]($)|(^)char[]($)|(^)float[]($)|(^)long[]($)|(^)short[]($)"**);**

private static final Pattern Returntype **=** Pattern**.**compile**(**"(^)int($)|(^)double($)|(^)char($)|(^)float($)|(^)long($)|(^)short($)|(^)bool($)|(^)void($)"**);**

private static final Pattern main **=** Pattern**.**compile**(**"int main()"**);**

private static final Pattern mainret **=** Pattern**.**compile**(**"return 0"**);**

private static final String patternfuncname **=** c **+** "[a-zA-z0-9]{1,14}"**;**

private static final String patternvariablename **=** "^[a-zA-z][a-zA-z0-9]{1,14}$"**;**

private static final Pattern variable **=** Pattern**.**compile**(**patternvariablename**);**

private static final String patternparameter **=** "[int|double|char|float|long|short|bool]\\s[a-zA-z][a-zA-z0-9]{1,14}$"**;**

private static final Pattern parameter **=** Pattern**.**compile**(**patternparameter**);**

private static final String patternfuncreturntype **=** **(**"[int|double|char|float|long|short|bool|void]"**);**

//void add (double a, double b)

private static final Pattern funcheader **=** Pattern**.**compile**(**"[int|double|char|float|long|short|bool|void]\\s[a-zA-z0-9]{1,14}\\s\\((.\*?)\\)"**);**

private static final Pattern commonfuncheader **=** Pattern**.**compile**(**"(\\w+)\\s(\\w+)\\s\\((.\*?)\\)"**);**

private static final Pattern funcCall **=** Pattern**.**compile**(**"[a-zA-z0-9]{1,14}\\s\\((.\*?)\\);"**);**

private static final Pattern patternReturn **=** Pattern**.**compile**(**"(^)return\\s[a-zA-z][a-zA-z0-9]{1,14}[;]$"**);**

// NOTE we won't have " int a = 1+2 " or "int a = a+b" a+# or #+b or a+b+c or etc...

// WE WILL ONLY HAVE THE BASE CASE int a; or int a = #;

private static final Pattern varNumberVer1 **=** Pattern**.**compile**(**"[int|double|float|long|short]\\s[a-zA-z][a-zA-z0-9]{0,14}[;]$"**);**

private static final Pattern varnumberVer2 **=** Pattern**.**compile**(**"[int|double|float|long|short]\\s[a-zA-z][a-zA-z0-9]{0,14}\\s"**+** "=\\s[0-9]{1,14};$"**);**

private static final Pattern varCharVer1 **=** Pattern**.**compile**(**"char\\s[a-zA-z][a-zA-z0-9]{0,14};$"**);**

private static final Pattern varCharVer2 **=** Pattern**.**compile**(**"char\\s[a-zA-z][a-zA-z0-9]{0,14}\\s"**+** "=\\s[']"**+**"[a-zA-z]"**+**"['][;]$"**);**

private static final Pattern varBoolVer1 **=** Pattern**.**compile**(**"bool\\s[a-zA-z][a-zA-z0-9]{0,14};$"**);**

private static final Pattern varBoolVer2 **=** Pattern**.**compile**(**"bool\\s[a-zA-z][a-zA-z0-9]{0,14}\\s"**+** "=\\s(true|false)"**+**"[;]$"**);**

/\*

int \* ip; pointer to an integer

double \* dp; pointer to a double

float \* fp; pointer to a float

char \* ch pointer to a character

\*/

private static final Pattern patternpointer **=** Pattern**.**compile**(**"[int|double|float|char]\\s[\*]\\s[a-zA-z][a-zA-z0-9]{0,14}"**+**"[;]$"**);**

// array declaration: int n [ 10 ] ;

// array maximum size is 99 minimum is 0

private static final Pattern patternArray**=** Pattern**.**compile**(**"[int|double|char]\\s[a-zA-z][a-zA-z0-9]{0,14}\\s"**+**"\\[\\s" **+** "[1-9][0-9]?\\s"**+** "\\][;]$"**);**

// array subscript string[i]

private static final Pattern patternArrSub **=** Pattern**.**compile**(**"[a-zA-z][a-zA-z0-9]{0,14}"**+**"\\s\\[" **+** "[0-9]{1,3}"**+** "\\]$"**);**

private static final Pattern patternPrintf **=** Pattern**.**compile**(**"printf\\((.\*?)\\)[;]$"**);**

private static final Pattern patternScanf **=** Pattern**.**compile**(**"scanf\\((.\*?)\\)[;]$"**);**

private static final Pattern patternStdio **=** Pattern**.**compile**(**"#\\sinclude\\s<stdio.h>"**);**

// common form of var/arr/ptr declaration

private static final Pattern CommonVar **=** Pattern**.**compile**(**"^?(\\w+)\\s^?(\\w+);$"**);**

private static final Pattern CommonVarWithInitialization **=** Pattern**.**compile**(**"^?(\\w+)\\s^?(\\w+)\\s=\\s(.\*?);$"**);**

private static final Pattern CommonPtr **=** Pattern**.**compile**(**"^?(\\w+)\\s?[\*]\\s?(\\w+);$"**);**

private static final Pattern CommonArr **=** Pattern**.**compile**(**"^?(\\w+)\\s(\\w+)\\s?\\[(.\*?)];$"**);**

//---------------------------------------

//---------------------------------If statment checks--------------------------------------------------------

private static final String ifStatementVar **=** "^?(if)\\((\\S+)\\)[{]?$?"**;**

private static final String ifStatementFunc **=** "^?(if)\\({1}\\S+\\(\\S\*?\\)\\){1}\\{?$?"**;**

private static final String ifStatementOps **=** "^?(if)\\s?\\(\\s?(\\S+)\\s?(\\((\\S+)\*\\))?\\s?(==|>|>=|<|<=|!=)\\s?(\\S+)\\s?(\\((\\S+)\*\\))?\\s?\\)\\s?[{]?$?"**;**

private static final Pattern ifStatVarCheck **=** Pattern**.**compile**(**ifStatementVar**);**

private static final Pattern ifStatFuncCheck **=** Pattern**.**compile**(**ifStatementFunc**);**

private static final Pattern ifStatOpCheck **=** Pattern**.**compile**(**ifStatementOps**);**

//--------------------------------- end If statment checks--------------------------------------------------------

//---------------------------------while statmentment checks--------------------------------------------------

private static final String whileStatementVar **=** "^?(while)\\((\\S+)\\)[{]?$?"**;**

private static final String whileStatementFunc **=** "^?(while)\\({1}\\S+\\(\\S\*?\\)\\){1}\\}?$?"**;**

private static final String whileStatementOps **=** "(while)\\({1}(\\S+\\(\\S\*?\\))?\\s?(==|>|>=|<|<=|!=)\\s?(\\S+)\\((\\S+\\(\\S\*?\\))?\\){1}\\{?$?"**;**

private static final Pattern whileStatVarCheck **=** Pattern**.**compile**(**whileStatementVar**);**

private static final Pattern whileStatFuncCheck **=** Pattern**.**compile**(**whileStatementFunc**);**

private static final Pattern whileStatOpCheck **=** Pattern**.**compile**(**whileStatementOps**);**

//----------------------------------end while statement setup----------------------------------------------------

//--------------------------------String = char array index------------------------------------------------------

private static final String arrIndex **=** "^?(\\S+)\\[{1}\\d+\\]{1}"**;**

private static final Pattern arrIndexCall **=** Pattern**.**compile**(**arrIndex**);**

//--------------------------------String = char array index------------------------------------------------------

//---------------------------------#14 assignment check -----------------------------------------------------------

private static final String assignStatement **=** "^((int|double|char|float|long|short|bool|void)\\s+)?(\\S+)\\s\*={1}\\s\*(\\S+)\\(?\\)?\\s\*;\\s\*$"**;**

private static final Pattern assignCheck **=** Pattern**.**compile**(**assignStatement**);**

//----------------------------------------------------- -----------------------------------------------------------

//----------------------------------#15 various type checks--------------------------------------------------------

private static final String mathCheck **=** "^?(\\w+)\\s?(\\+|\\-|\\\*|\\/){1}\\s?(\\w+)"**;**

private static final String logicalCheck **=** "[^#]\*(\\w)+(\\((\\S\*)\\))?\\s\*(&&|\\|\\|){1}\\s?(\\w)+(\\((\\S\*)\\))?\\s\*"**;**

private static final String compCheck **=** "[^#]\*(\\w)+(\\((\\S\*)\\))?\\s\*(<|<=|>|>=){1}\\s?(\\w)+(\\((\\S\*)\\))?\\s\*"**;**

private static final String equalCheck **=** "(\\w)+(\\((\\S\*)\\))?\\s\*(==|!=){1}\\s?(\\w)+(\\((\\S\*)\\))?\\s\*"**;**

private static final String absValCheck **=** "(\\w\*)\*\\|[^\\s](\\S+)\\|(\\w\*)\*"**;**

private static final String negationCheck **=** "\\!(\\w+)"**;**

private static final Pattern negTypeCheck **=** Pattern**.**compile**(**negationCheck**);**

private static final Pattern absValTypeCheck **=** Pattern**.**compile**(**absValCheck**);**

private static final Pattern equalTypeCheck **=** Pattern**.**compile**(**equalCheck**);**

private static final Pattern compTypeCheck **=** Pattern**.**compile**(**compCheck**);**

private static final Pattern logicalTypeCheck **=** Pattern**.**compile**(**logicalCheck**);**

private static final Pattern mathTypeCheck **=** Pattern**.**compile**(**mathCheck**);**

//-------------------------------------------#15 end --------------------------------------------------

public static boolean visitCommonVar **(**String target**)** **{**

**return** **(**CommonVar**.**matcher**(**target**).**find**()** **||** CommonVarWithInitialization**.**matcher**(**target**).**find**());**

**}**

public static boolean visitCommonPtr **(**String target**)** **{**

**return** CommonPtr**.**matcher**(**target**).**find**();**

**}**

public static boolean visitCommonArr **(**String target**)** **{**

**return** CommonArr**.**matcher**(**target**).**find**();**

**}**

public static boolean visitcommonfuncheader **(**String target**)** **{**

**return** commonfuncheader**.**matcher**(**target**).**find**();**

**}**

public static boolean visitMainret **(**String target**)** **{**

**return** mainret**.**matcher**(**target**).**find**();**

**}**

public static boolean visitPrintf **(**String target**)** **{**

**return** patternPrintf**.**matcher**(**target**).**find**();**

**}**

public static boolean visitScanf **(**String target**)** **{**

**return** patternScanf**.**matcher**(**target**).**find**();**

**}**

public static boolean visitStdio **(**String target**)** **{**

**return** patternStdio**.**matcher**(**target**).**find**();**

**}**

public static boolean visitPointer **(**String target**)** **{**

**return** patternpointer**.**matcher**(**target**).**find**();**

**}**

public static boolean visitArray **(**String target**)** **{**

**return** patternArray**.**matcher**(**target**).**find**();**

**}**

public static boolean visitArrSub **(**String target**)** **{**

**return** patternArrSub**.**matcher**(**target**).**find**();**

**}**

//private static final Pattern variableDeclation = Pattern.compile("[var1|var2]"));

public static boolean visitVariableDeclaration **(**String target**)** **{**

**return** **(**varNumberVer1**.**matcher**(**target**).**find**()** **||** varnumberVer2**.**matcher**(**target**).**find**()**

**||** varCharVer1**.**matcher**(**target**).**find**()** **||** varCharVer2**.**matcher**(**target**).**find**()**

**||** varBoolVer1**.**matcher**(**target**).**find**()** **||** varBoolVer2**.**matcher**(**target**).**find**());**

**}**

public static boolean visitReturn **(**String target**)** **{**

**return** patternReturn**.**matcher**(**target**).**find**();**

**}**

public static boolean visitVarAssignFunc **(**String target**)** **{**

**if(!(**target**.**contains**(**"="**))){**

**return** **false;**

**}**

**else{**

String**[]** arr **=** target**.**split**(**"="**);**

// check left side the variable matching

**if(!(**visitVariable**(**arr**[**0**].**trim**()))){**

**return** **false;**

**}**

// check right side the function call matching

**if(!(**visitFunctionCall**(**arr**[**1**].**trim**()))){**

**return** **false;**

**}**

**}**

**return** **true;**

**}**

public static boolean visitFunctionCall **(**String target**)** **{**

**return** funcCall**.**matcher**(**target**).**find**();**

**}**

public static boolean visitVariable **(**String target**)** **{**

**return** variable**.**matcher**(**target**).**find**();**

**}**

public static boolean visitParam **(**String target**)** **{**

**return** parameter**.**matcher**(**target**).**find**();**

**}**

public static boolean visitPrimitive **(**String target**)** **{**

**return** primitive**.**matcher**(**target**).**find**();**

**}**

public static boolean visitReturnType **(**String target**)** **{**

**return** Returntype**.**matcher**(**target**).**find**();**

**}**

public static boolean visitmain **(**String target**)** **{**

**return** main**.**matcher**(**target**).**find**();**

**}**

public static boolean visitfunction **(**String target**)** **{**

/\*

\* 3 cases only

\* int func1 ( int a, int b )

\* int func1 ( int a )

\* int func1 ( )

\* any will work go in if block and check for the parameter

\*/

**if(**funcheader**.**matcher**(**target**).**find**()** **==** **false){**

**return** **false;**

**}**

**if(**funcheader**.**matcher**(**target**).**find**()){**

// check param by first substring them out

target **=** target**.**replaceAll**(**","**,** ""**);**

String paramlist **=** target**.**substring**(**target**.**indexOf**(**"("**)+**1**,**target**.**indexOf**(**")"**));**

paramlist **=** paramlist**.**trim**();**

// case int func1 ( )

**if(**paramlist**.**equals**(**""**)** **||** paramlist**.**isEmpty**()){**

**return** **true;**

**}**

// case at least 1 param

**else{**

// split the param using space

String**[]** paramArr **=** paramlist**.**split**(**" "**);**

**if((**paramArr**.**length **%** 2**)** **==** 1**){**

//param arr size must be even 2 4 6 8 10

**return** **false;**

**}**

// check all data type

**for(**int i **=** 0**;** i **<** paramArr**.**length**;** i **=** i**+**2**){**

**if(!(**visitPrimitive**(**paramArr**[**i**]))){**

// not primitive type

**return** **false;**

**}**

**}**

// check the variable naming limit

**for(**int j **=** 1**;** j **<** paramArr**.**length**;** j **=** j**+**2**){**

**if(!(**visitVariable**(**paramArr**[**j**]))){**

// variable name isn't passing

**return** **false;**

**}**

**}**

**}**

**}**

**return** **true;**

**}**

// remove all comma, semi colon

public String RemoveAllCommaNSemicolon**(**String token**){**

token **=** token**.**replace**(**","**,** ""**);**

token **=** token**.**replace**(**";"**,** ""**);**

**return** token**;**

**}**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*start if statements\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

public static boolean visitIfStatOper**(**String target**){**

**return** ifStatOpCheck**.**matcher**(**target**).**find**();**

**}**

public static boolean visitIfStatFunc**(**String target**){**

**return** ifStatFuncCheck**.**matcher**(**target**).**find**();**

**}**

public static boolean visitIfStatVar**(**String target**){**

**return** ifStatVarCheck**.**matcher**(**target**).**find**();**

**}**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*end if statements\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*start while loops\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

public static boolean visitWhileStatOper**(**String target**){**

**return** whileStatOpCheck**.**matcher**(**target**).**find**();**

**}**

public static boolean visitWhileStatFunc**(**String target**){**

**return** whileStatFuncCheck**.**matcher**(**target**).**find**();**

**}**

public static boolean visitWhileStatVar**(**String target**){**

**return** whileStatVarCheck**.**matcher**(**target**).**find**();**

**}**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*end while loops\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//check for char array index is an int

public static boolean visitCharArray**(**String target**){**

**return** arrIndexCall**.**matcher**(**target**).**find**();**

**}**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*14 - assignment check \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

public static boolean visitAssignmentCheck**(**String target**){**

**return** assignCheck**.**matcher**(**target**).**find**();**

**}**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*15 - start various checks \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

public static boolean visitMathCheck**(**String target**){**

**return** mathTypeCheck**.**matcher**(**target**).**find**();**

**}**

public static boolean visitLogicalCheck**(**String target**){**

**return** logicalTypeCheck**.**matcher**(**target**).**find**();**

**}**

public static boolean visitCompCheck**(**String target**){**

**return** compTypeCheck**.**matcher**(**target**).**find**();**

**}**

public static boolean visitEqualCheck**(**String target**){**

**return** equalTypeCheck**.**matcher**(**target**).**find**();**

**}**

public static boolean visitAbsValCheck**(**String target**){**

**return** absValTypeCheck**.**matcher**(**target**).**find**();**

**}**

public static boolean visitNegCheck**(**String target**){**

**return** negTypeCheck**.**matcher**(**target**).**find**();**

**}**

**}**

Input:

# include <stdio.h>

void add **(**int firstNumber**,** int secondNumber**){**

printf**(**"O Result e: %d"**,** firstNumber **+** secondNumber**);**

**}**

void sub **(**int firstNumber**,** int secondNumber**){**

printf**(**"O Result e: %d"**,** firstNumber **-** secondNumber**);**

**}**

void mul **(**int firstNumber**,** int secondNumber**){**

printf**(**"O Result e: %d"**,** firstNumber **\*** secondNumber**);**

**}**

void div **(**int firstNumber**,** int secondNumber**){**

printf**(**"O Result e: %d"**,** firstNumber **/** secondNumber**);**

**}**

int main**()** **{**

char operator**;**

char mychar **=** 'a'**;**

int firstNumber**;**

int secondNumber**;**

int thirdNumber **=** 7**;**

int fourNumber **=** 9**;**

bool check**;**

bool my **=** true**;**

int **\*** myintptr**;**

int myarr **[** 5 **];**

char num5 **=** '5'**;**

thirdNumber **=** firstNumber **+** num5**;**

abs **(**firstNumber**);**

firstNumber **=** **^**thirdNumber**;**

**while(**firstNumber**){};**

**if(**firstNumber**){};**

check **=** firstNumber **&&** secondNumber**;**

printf**(**"Enter an operator (+, -, \*,): "**);**

scanf**(**"%c"**,** **&**operator**);**

printf**(**"Enter two operands: "**);**

scanf**(**"%lf %lf"**,&**firstNumber**,** **&**secondNumber**);**

**if** **(** operator **==** '+' **)** **{**

add **(**firstNumber**,** secondNumber**);**

**}**

**else** **if** **(** operator **==** '-' **)** **{**

sub **(**firstNumber**,** secondNumber**);**

**}**

**else** **if** **(** operator **==** '\*' **)** **{**

mul **(**firstNumber**,** secondNumber**);**

**}**

**else** **if** **(** operator **==** '/' **)** **{**

div **(**firstNumber**,** secondNumber**);**

**}**

**return** 0**;**

**}**

Output:

ignore line

ignore line

ignore line

ignore line

ignore line

ignore line

ignore line

ignore line

ignore line

ignore line

ignore line

ignore line

# include <stdio.h>

not declaration

void add (int firstNumber, int secondNumber){

function header declaration passed

printf("O Result e: %d", firstNumber + secondNumber);

not declaration

}

not declaration

void sub (int firstNumber, int secondNumber){

function header declaration passed

printf("O Result e: %d", firstNumber - secondNumber);

not declaration

}

not declaration

void mul (int firstNumber, int secondNumber){

function header declaration passed

printf("O Result e: %d", firstNumber \* secondNumber);

not declaration

}

not declaration

void div (int firstNumber, int secondNumber){

function header declaration passed

printf("O Result e: %d", firstNumber / secondNumber);

not declaration

}

not declaration

int main() {

not declaration

char operator;

variable declaration passed

char mychar = 'a';

variable declaration passed

int firstNumber;

variable declaration passed

int secondNumber;

variable declaration passed

int thirdNumber = 7;

variable declaration passed

int fourNumber = 9;

variable declaration passed

bool check;

variable declaration passed

bool my = true;

variable declaration passed

int \* myintptr;

ptr declaration passed

int myarr [ 5 ];

array declaration passed

char num5 = '5';

variable declaration passed

thirdNumber = firstNumber + num5;

not declaration

abs (firstNumber);

not declaration

firstNumber = ^thirdNumber;

not declaration

while(firstNumber){};

not declaration

if(firstNumber){};

not declaration

check = firstNumber && secondNumber;

not declaration

printf("Enter an operator (+, -, \*,): ");

not declaration

scanf("%c", &operator);

not declaration

printf("Enter two operands: ");

not declaration

scanf("%lf %lf",&firstNumber, &secondNumber);

not declaration

if ( operator == '+' ) {

not declaration

add (firstNumber, secondNumber);

not declaration

}

not declaration

else if ( operator == '-' ) {

not declaration

sub (firstNumber, secondNumber);

not declaration

}

not declaration

else if ( operator == '\*' ) {

not declaration

mul (firstNumber, secondNumber);

not declaration

}

not declaration

else if ( operator == '/' ) {

not declaration

div (firstNumber, secondNumber);

not declaration

}

not declaration

return 0;

not declaration

}

not declaration

END OF DECLARATION CHECK

# include <stdio.h>

stdio passed

printf("O Result e: %d", firstNumber + secondNumber);

printf passed

}

ignore line

printf("O Result e: %d", firstNumber - secondNumber);

printf passed

}

ignore line

printf("O Result e: %d", firstNumber \* secondNumber);

printf passed

}

ignore line

printf("O Result e: %d", firstNumber / secondNumber);

printf passed

}

ignore line

int main() {

main passed

thirdNumber = firstNumber + num5;

failed to pass the typecheck

abs (firstNumber);

error 5 function name not found in hashmap

firstNumber = ^thirdNumber;

error code 18: Deref variable not found

while(firstNumber){};

error 5 function name not found in hashmap

if(firstNumber){};

error 5 function name not found in hashmap

check = firstNumber && secondNumber;

printf("Enter an operator (+, -, \*,): ");

printf passed

scanf("%c", &operator);

scanf passed

printf("Enter two operands: ");

printf passed

scanf("%lf %lf",&firstNumber, &secondNumber);

scanf passed

if ( operator == '+' ) {

failed to pass the typecheck

add (firstNumber, secondNumber);

function call passed

}

ignore line

else if ( operator == '-' ) {

failed to pass the typecheck

sub (firstNumber, secondNumber);

function call passed

}

ignore line

else if ( operator == '\*' ) {

failed to pass the typecheck

mul (firstNumber, secondNumber);

function call passed

}

ignore line

else if ( operator == '/' ) {

failed to pass the typecheck

div (firstNumber, secondNumber);

function call passed

}

ignore line

return 0;

return 0 for main

}

ignore line

open brace count is : 11

close brace count is : 11

found comparison operation: # include <stdio.h>

found math operation: printf("O Result e: %d", firstNumber + secondNumber);

found math operation: printf("O Result e: %d", firstNumber - secondNumber);

found math operation: printf("O Result e: %d", firstNumber \* secondNumber);

found math operation: printf("O Result e: %d", firstNumber / secondNumber);

found math operation: thirdNumber = firstNumber + num5;

error code 15: variable type isn't int: num5

error code 10: invalid while statement (variable must be of type bool))

while(firstNumber){}; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Late check fail\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

error code 10: invalid if statement (function must return bool type))

if(firstNumber){}; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Late check fail\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

comparison operator found: check = firstNumber && secondNumber;

error code 15: variable type isn't bool: firstNumber

error code 15: variable type isn't bool: secondNumber

check = firstNumber && secondNumber;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Late syntax check failed\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*if ( operator == '+' ) {\*\*\* found if statement with operator

\*\*\*else if ( operator == '-' ) {\*\*\* found if statement with operator

\*\*\*else if ( operator == '\*' ) {\*\*\* found if statement with operator

\*\*\*else if ( operator == '/' ) {\*\*\* found if statement with operator