ABSTRACT

In the world of computer science, data structure refers to the format that contains a collection of data values, their relationships, and the functions that can be applied to the data. Data structures arrange data so that it can be accessed and worked on with specific algorithms more effectively.

Syntax checker is a basic compiler designed to check certain syntaxes in a java code. It makes ones life easier by helping them debug their syntaxes if they aren't sure of the right syntax.

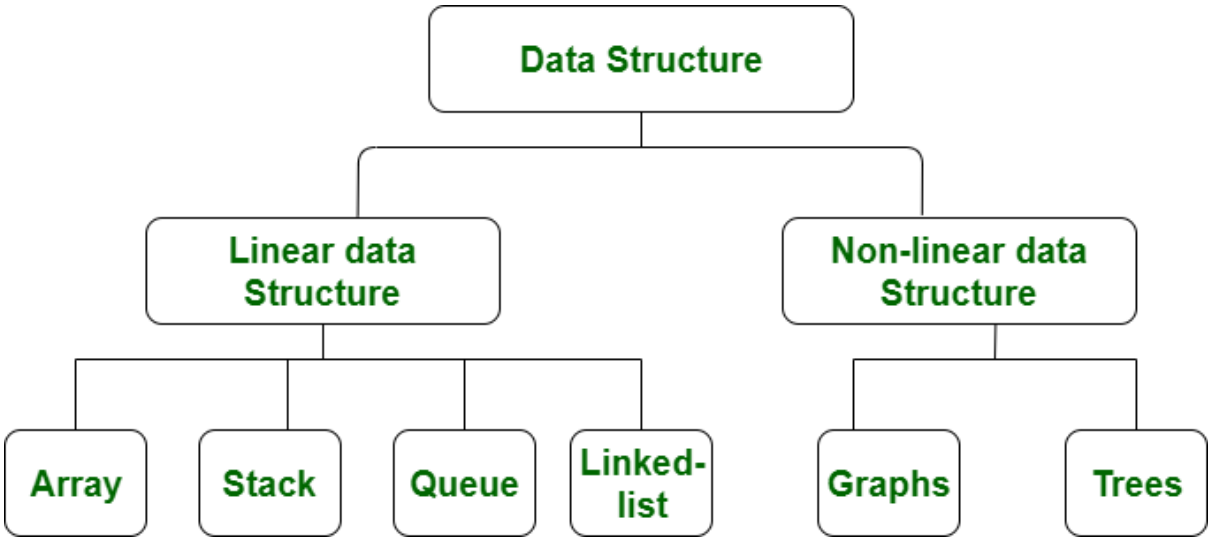
In our project, we plan to make it easier for the user to check their syntaxes by asking them to enter a syntax and debugging it for them and displaying an appropriate message. Parenthesis checking, for loop checking, while loop checking, do while loop checking, if statement checking, keyword checking, datatype checking and identifier checking have been done using various data structures such as arrays, linked list and stacks in Eclipse IDE. We have used the java.util.* package to include all the necessary packages required for this program.

In this program, one will be able to identify if they have made any mistakes or errors in a syntax of their code.

DATA STRUCTURES USED

A Data structure is a particular way of organizing data in a computer so that it can be used effectively. It is a collection of data types. It is a way or organizing the items in terms of memory, and also the way of accessing each item through some defined logic.

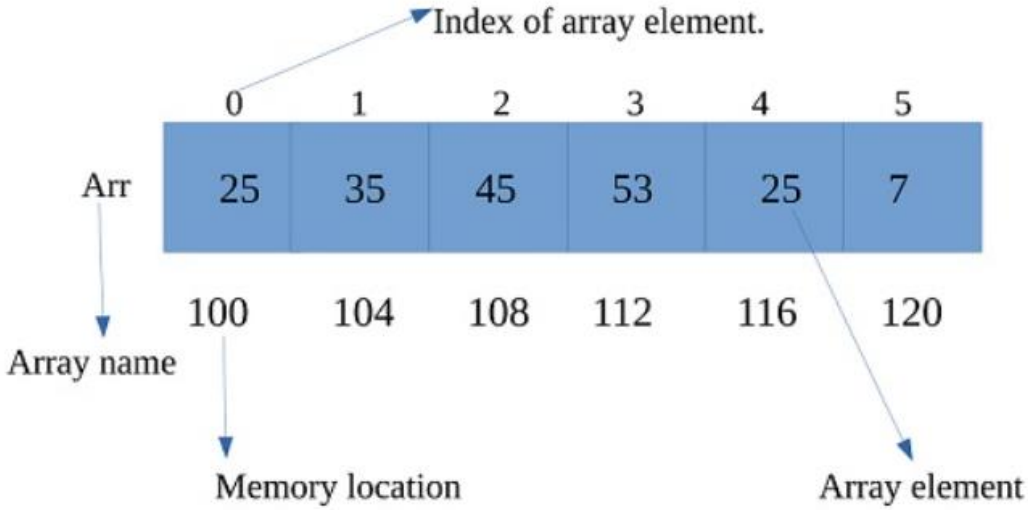
Data structures are of two types, Linear and Non-Linear. Under Linear data structures, we have Arrays, Linked Lists, Stacks and Queues and under Non-Linear data structures, we have Trees and Graphs.



In our program, we have used 3 Linear Data Structures, they are, Arrays, Stacks and Linked Lists.

Arrays

An array is a collection of homogeneous (same type) data items stored in contiguous memory locations. The idea is to store multiple items of the same type together. This makes it easier to calculate the position of each element by simply adding an offset to a base value, i.e., the memory location of the first element of any array.



In our code, we have used arrays to check the expression for the datatypes. We ask the user to enter the datatype for which they want to check if there are any errors or not, that input is stored in an array. We then have an array which consists of all the right datatypes. We compare the input array to the datatypes array using the .equals() command and check whether the expression entered by the user is right or not. We have also used arrays in our Stack, the elements of our stack is stored in an array called stackArr[].

Stacks

A stack is an abstract data type that holds an ordered, linear sequence of items. It follows a particular order in which the operations are performed.

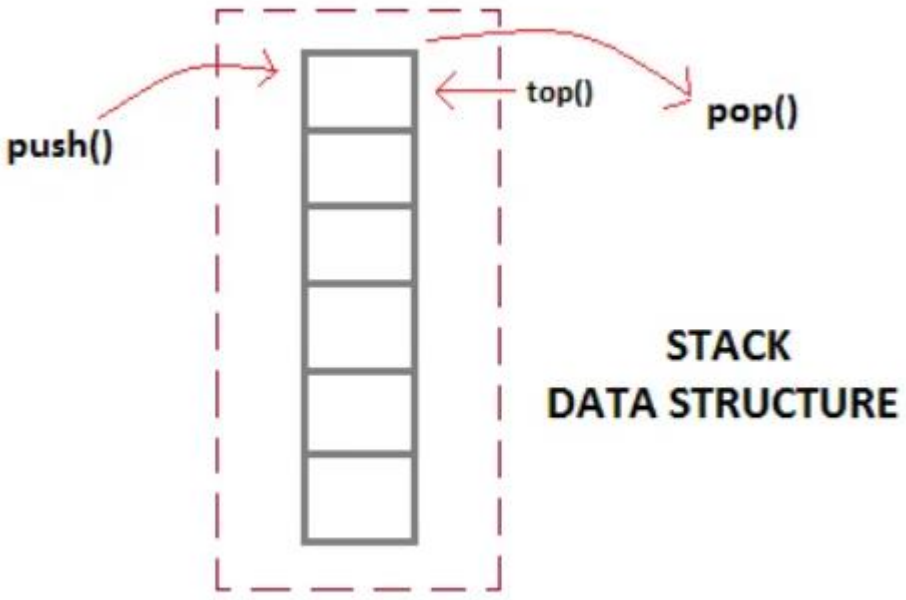
The main operations that are performed in the stack are:

Push: adds an item in the stack. If the stack is full, then it is said to be overflow condition.

Pop: removes an item from the stack. The items are popped in the reverse order in which they are pushed. If the stack is empty, then its said to be an underflow condition.

Peep: returns the top element of stack

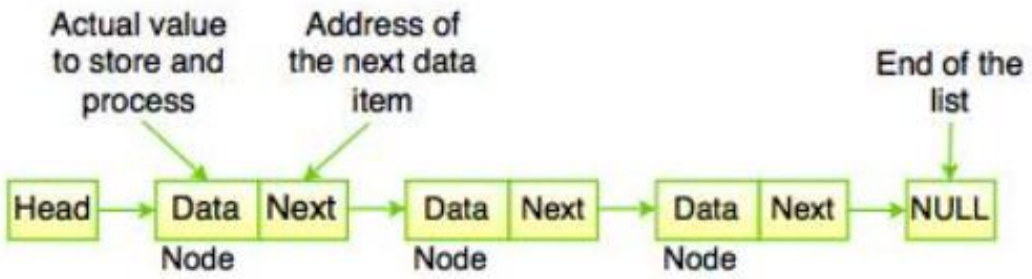
isEmpty(): returns true is stack is empty, else false.



In our program, we have used stack in the DelimiterMatching function to check whether the number of parentheses are balanced. We also use Stacks in the WhileLoopChecker to input the size of the stack which is basically the size of the inputted expression. The same is used in ForLoopChecker, DoWhileLoopChecker as well as IfStatementChecker.

Linked Lists

A linked list is a sequence of data structures, which are connected together via links. Linked List is a sequence of links which contains items. Each link contains a connection to another link. There are three main types- Singly Linked list, Doubly Linked List and Circular Linked List



In our program, we have used Singly Linked list to check for identifiers as well as to check for keywords.

We use stacks and store all the keywords and characters and using lists, we compare and identify if the input inputted by the user is an identifier or a keyword.

CODE

```
package Project;
import java.util.*;
public class SyntaxChecker
{
        public static void main(String args[])
   {
       String expression;
       String expre;
       String[] a = new String[100];
       Scanner sc = new Scanner(System.in);
       int choice;
       do
       {
          System.out.println("Enter your choice: ");
          System.out.println("1. brackets checking");
          System.out.println("2. while loop checking");
          System.out.println("3. for loop checking");
          System.out.println("4. do while loop checking");
          System.out.println("5. if statement checking");
          System.out.println("6. keyword checker");
          System.out.println("7. datatype checker");
          System.out.println("8. identifier checker");
          System.out.println("9. exit");
          choice = sc.nextInt();

          switch(choice)
          {
               case 1: System.out.println("Enter your expression to check
brackets: ");
                       expression = sc.next();
                       boolean result =
SyntaxChecker.isDelimiterMatching(expression);
                       System.out.println(expression +" == "+result);
                       System.out.println();
                       break;
```

```
            case 2: System.out.println("Enter your while expression: ");
                expre = sc.next();
                SyntaxChecker.whileLoopChecker(expre);
                        System.out.println();
                           break;

            case 3: System.out.println("Enter your for expression: ");
                expre = sc.next();
                SyntaxChecker.forLoopChecker(expre);
                        System.out.println();
                           break;

            case 4: System.out.println("Enter your do-while expression: ");
                            expre = sc.next();
                            SyntaxChecker.DoWhileLoopChecker(expre);
                           System.out.println();
                           break;

            case 5: System.out.println("Enter your if expression: ");
                                expre = sc.next();

        SyntaxChecker.IfStatementChecker(expre);
                                System.out.println();
                                break;

            case 6: System.out.println("Enter your keyword: ");
                    String v = sc.next();
                    SLIST obj = new SLIST();
                    obj.keywordsCharacters();
                    obj.KeywordChecker(v);
                                    System.out.println();
                                    break;

            case 7: System.out.println("Enter your datatype: ");
                    String z = sc.next();
```

```
                a[0] = z;
                SyntaxChecker.DataTypeChecker(a);
                System.out.println();
                                break;

            case 8: System.out.println("Enter your identifier: ");
                                String Y = sc.next();
                                SLIST obj1 = new SLIST();
                                obj1.keywordsCharacters();

                                obj1.IdentifierChecker(Y);
                                System.out.println();
                                        break;

                default: System.out.println("Invalid Input!");
        }
    }
    while(choice!=9);
    sc.close();
}
public static boolean isDelimiterMatching(String inputExpr)
{
    int stackSize = inputExpr.length();
    Stack theStack = new Stack(stackSize);
    for (int j = 0; j < inputExpr.length(); j++)
    {
        char ch = inputExpr.charAt(j);
        switch (ch)
        {
            case '{':
            case '[':
            case '(':
                    theStack.push(ch);
                    break;
            case '}':
            case ']':
            case ')':
```

```
                    if (!theStack.isStackEmpty())
                    {
                      char stackContent = theStack.pop();
                      if ((ch == '}' && stackContent != '{') || (ch == ']' &&
stackContent != '[') || (ch == ')' && stackContent != '('))
                      {
                         System.out.println("\nMismatch found");
                         return false;
                      }
                    }
                    else
                    {
                      System.out.println("\nMismatch found ");
                      return false;
                    }
                    break;
               default:  break;
            }
        }
        if (!theStack.isStackEmpty())
        {
           System.out.println("\nError: missing right delimiter");
           System.out.println();
           return false;
        }
        return true;
    }
    public static boolean whileLoopChecker(String input)
    {
         int flag=0;
         int stackSize = input.length();
         if( (input.length()<6) ||(input.charAt(0) != 'w') || (input.charAt(1)
!= 'h') ||
             (input.charAt(2)  !=  'i')    ||  (input.charAt(3)  !=  'l')    ||
(input.charAt(4) != 'e'))
         {
                System.out.println("\nError in 'while' keyword usage");
```

```
            return false;
        }
        else
        {
            if((input.charAt(5)!='(') && (input.charAt(stackSize- 1 ) != ')'))
            {
             System.out.println("\nSyntax error, Brackets are missing");
               flag++;
            }
            else if(input.charAt(5)!='(')
            {
             System.out.println("\nOpening parenthesis absent after 'while' keyword");
               flag++;
            }
            else if(input.charAt(stackSize-1) != ')')
            {
               System.out.println("\nClosing parenthesis absent at the end of while");
               flag++;
            }
            else if((input.charAt(7)) != '>' && (input.charAt(7))!='<' &&
                  ((input.charAt(7))!='=' || (input.charAt(8))!='=') &&
                  ((input.charAt(7))!='!' || (input.charAt(8))!='='))
            {
               System.out.println("\nError in the usage of symbol");
               flag++;
            }
            if(flag == 0)
            {
               System.out.println("\nNo error!");
            }
         }
        return true;
    }
```

```
public static boolean forLoopChecker(String input)
{
   int bracket1 = 0, bracket2 = 0, flag = 0;
   int stackSize = input.length();
   if( (input.length()<3)       || (input.charAt(0) != 'f') ||
      (input.charAt(1) != 'o') || (input.charAt(2) != 'r'))
   {
    System.out.println("\nError in 'for' keyword usage");
    return false;
   }
   else
   {
    for(int i=0;i<stackSize;i++)
    {
          char ch = input.charAt(i);
          if(ch == '(')
          {
                bracket1 ++;
          }
          else if(ch == ')')
          {
                bracket2 ++;
          }
          else if(ch == ';')
          {
          }
          else if(ch == ' ')
          {
                continue;
          }
          else
          {
                continue;
          }
    }
      if((input.charAt(3)!='(') && (input.charAt(stackSize- 1 ) != ')'))
```

```
        {
                System.out.println("\nSyntax error, Brackets are missing");
                flag++;
        }
        else if(input.charAt(3)!='(')
        {
                System.out.println("\nOpening parenthesis absent after 'for'
keyword");
                flag++;
        }
        else if(input.charAt(stackSize-1 ) != ')')
        {
                System.out.println("\nClosing parenthesis absent at the end
of for");
                flag++;
        }
        else if((input.charAt(stackSize-5))!=';' || (input.charAt(stackSize-
9))!=';' && (input.charAt(stackSize-10))!=';')
           {
              System.out.println("\nSemicolon Error");
              flag++;
           }
        else if(bracket1 != 1 || bracket2 != 1 || bracket1 != bracket2)
        {
                System.out.println("\nParentheses Count Error");
                flag++;
        }
        if(flag == 0)
        {
                System.out.println("\nNo error!");
        }
        return true;
       }
    }
    public static boolean DoWhileLoopChecker(String input)
    {
        int flag = 0;
```

```
        int stackSize = input.length();

        if((input.length()<3) || (input.charAt(0) != 'd') || (input.charAt(1)
!= 'o'))
        {
                System.out.println("\nError in 'do' keyword usage");
                return false;
        }
        else
        {
               if((input.charAt(2)!='{') && (input.charAt(stackSize- 13) !=
'}') && (input.charAt(stackSize- 12 ) != '}'))
               {
                System.out.println("\nSyntax error, Brackets are missing");
                  flag++;
               }
               else if(input.charAt(stackSize-1) != ';')
               {
                System.out.println("\nMissing semicolon after while");
                flag++;
               }
               else if(input.charAt(2) != '{')
               {
                  System.out.println("\nOpening parenthesis absent after
'do' keyword");
                  flag++;
               }
               else if(input.charAt(stackSize-13)!='}' &&
input.charAt(stackSize-12)!='}')
               {
                System.out.println("\nClosing parenthesis absent after 'do'
statement");
                  flag++;
               }
               else if(input.charAt(stackSize- 2 ) != ')')
               {
```

```
                System.out.println("\nClosing parenthesis absent at end of while condition");
                flag++;
            }
            else if(input.charAt(stackSize- 7)!='(' && input.charAt(stackSize- 6)!='(')
            {
             System.out.println("\nOpening parenthesis absent after 'while' keyword");
                flag++;
            }
            else if(((input.charAt(stackSize-4) != '>') && (input.charAt(stackSize-5) != '>'))
                                        &&
                    ((input.charAt(stackSize-4) != '<') && (input.charAt(stackSize-5) != '<'))
                                        &&
                    ((input.charAt(stackSize-5) != '!') && (input.charAt(stackSize-6) != '!'))
                                        &&
                    ((input.charAt(stackSize-4) != '=') && (input.charAt(stackSize-5) != '='))
                                        &&
                    ((input.charAt(stackSize-5) != '=') && (input.charAt(stackSize-6) != '='))
                                        &&
                    ((input.charAt(stackSize-4) != '=') && (input.charAt(stackSize-5) != '=')))
            {
                System.out.println("\nError in the usage of symbol");
                flag++;
            }
            if( (input.length()<6) || (input.charAt(stackSize-12) != 'w') && (input.charAt(stackSize-11) != 'w') ||
                (input.charAt(stackSize-11) != 'h') && (input.charAt(stackSize-10) != 'h') ||
```

```
                (input.charAt(stackSize-10) != 'i') &&
(input.charAt(stackSize-9) != 'i') ||
                (input.charAt(stackSize-9) != 'l') &&
(input.charAt(stackSize-8) != 'l') ||
                (input.charAt(stackSize-8) != 'e') &&
(input.charAt(stackSize-7) != 'e'))
                    {
                     System.out.println("\nError in 'while' keyword
usage");
                     return false;
                    }
            if(flag == 0)
            {
               System.out.println("\nNo error!");
            }
         }
        return true;
    }
    public static boolean IfStatementChecker(String input)
    {
        int flag = 0;
        int stackSize = input.length();
        if( (input.length()<3) ||(input.charAt(0) != 'i') || (input.charAt(1)
!= 'f') )
        {
              System.out.println("\nError in 'if' keyword usage");
              return false;
        }
        else
        {
              if((input.charAt(2)!='(') && (input.charAt(stackSize- 1 ) !=
')'))
              {
               System.out.println("\nSyntax error, Brackets are missing");
                 flag++;
              }
              else if(input.charAt(stackSize- 1 ) != ')')
```

```
            {
               System.out.println("\nClosing parenthesis absent at the
end of statement");
               flag++;
            }
            else if(input.charAt(2)!='(')
            {
             System.out.println("\nOpening parenthesis absent after 'if'
keyword");
               flag++;
            }
            else if(((input.charAt(stackSize-4) != '>') ||
(input.charAt(stackSize-3) != '>'))
                                                   &&
                  ((input.charAt(stackSize-4) != '<') ||
(input.charAt(stackSize-3) != '<'))
                                                   &&
                  ((input.charAt(stackSize-5) != '!') ||
(input.charAt(stackSize-4) != '!'))
                                                   &&
                  ((input.charAt(stackSize-4) != '=') ||
(input.charAt(stackSize-3) != '='))
                                                   &&
                  ((input.charAt(stackSize-5) != '=') ||
(input.charAt(stackSize-4) != '='))
                                                   &&
                  ((input.charAt(stackSize-4) != '=') ||
(input.charAt(stackSize-3) != '=')))
            {
               System.out.println("\nError in the usage of symbol");
               flag++;
            }
            if(flag == 0)
            {
               System.out.println("\nNo error!");
            }
         }
```

```
        return true;
    }
    public static boolean DataTypeChecker(String[] a)
    {
        String[] ss= {"int", "char", "boolean", "float", "double", "short",
"long", "byte"};
        if(ss[0].equals(a[0]) || ss[1].equals(a[0]) || ss[2].equals(a[0]) ||
ss[3].equals(a[0]) ||
          ss[4].equals(a[0]) || ss[5].equals(a[0]) || ss[6].equals(a[0]) ||
ss[7].equals(a[0]) )
        {
              System.out.println("\n'"+a[0]+ "' is a datatype!");
        }
        else
        {
              System.out.println("\n'"+a[0]+ "' is not a datatype");
        }
        return true;
    }
}
class Stack
{
    int stackSize;
    char[] stackArr;
    int top;
    public Stack(int size)
    {
       stackSize = size;
       stackArr = new char[stackSize];
       top = -1;
    }
    public void push(char entry)
    {
       stackArr[++top] = entry;
    }
    public char pop()
    {
```

```
        char entry = stackArr[top--];
        return entry;
    }
    public char peek()
    {
        return stackArr[top];
    }
    public boolean isStackEmpty()
    {
        return (top == -1);
    }
    public boolean isStackFull()
    {
        return (top == stackSize - 1);
    }
}
class Node
{
    String data;
    Node next;
    Node()
    {
        data = "";
        next = null;
    }
    Node(String element)
    {
        data =element;
        next = null;
    }
    Node(String element, Node n)
    {
        data = element;
        next = n;
    }
}
class SLIST
```

```
{
   Node head;
   SLIST()
   {
      head = null;
   }
   SLIST(String s)
   {
      head = new Node(s);
   }
   void insertfront(String s)
   {
      Node temp = new Node(s);
      temp.next = head;
      head = temp;
   }
   public boolean IdentifierChecker(String input)
   {
        Node temp = head;
       if((input.charAt(0)>='A' && input.charAt(0)<='Z') || (input.charAt(0)>='a' && input.charAt(0)<='z') || (input.charAt(0) == '$') || (input.charAt(0) == '_'))
       {
          while(temp!=null)
          {
             if(input.compareTo(temp.data)==0)
             {
               System.out.println("\nKeyword '"+input+"' is used, therefore it isn't a valid identifier");
                 return false;
             }
             temp=temp.next;
          }
          temp = head;
          while(temp!=null)
          {
             for(int i=1; i<input.length(); i++)
```

```
            {

        if((Character.toString(input.charAt(i))).compareTo(temp.data)==0||(int)input.charAt(i)==34||(int)input.charAt(i)==92)
                        {
                                System.out.println("\nSpecial character other than '_' and '$' is used, it is not an identifier");
                           return false;
                        }
                }
                temp=temp.next;
            }
        }
        else
        {
         System.out.println("\n'"+input+"', the starting violates the rules of an identifier");
            return false;
        }
        System.out.println("\n'"+input+"' is a valid identifier!");
        return true;
    }
    public boolean KeywordChecker(String a)
    {
        Node temp = head;
        while(temp != null)
        {
                if(a.compareTo(temp.data)==0)
          {
                System.out.println("\n'"+a+ "' is a keyword!");
                return false;
        }
                temp=temp.next;
        }

        System.out.println("\n'"+a+ "' is not a keyword");
```

```
    return true;
}
void keywordsCharacters()
{
   insertfront("abstract");
   insertfront("int");
   insertfront("char");
   insertfront("continue");
   insertfront("for");
   insertfront("new");
   insertfront("switch");
   insertfront("assert");
   insertfront("default");
   insertfront("goto");
   insertfront("package");
   insertfront("synchronized");
   insertfront("boolean");
   insertfront("do");
   insertfront("if");
   insertfront("private");
   insertfront("this");
   insertfront("break");
   insertfront("double");
   insertfront("implements");
   insertfront("protected");
   insertfront("throw");
   insertfront("byte");
   insertfront("else");
   insertfront("import");
   insertfront("public");
   insertfront("throws");
   insertfront("case");
   insertfront("enum");
   insertfront("instanceof");
   insertfront("return");
   insertfront("transient");
   insertfront("catch");
```

```
insertfront("extends");
insertfront("int");
insertfront("short");
insertfront("try");
insertfront("char");
insertfront("final");
insertfront("interface");
insertfront("static");
insertfront("void");
insertfront("class");
insertfront("finally");
insertfront("long");
insertfront("strictfp");
insertfront("volatile");
insertfront("const");
insertfront("float");
insertfront("native");
insertfront("super");
insertfront("while");
insertfront("*");
insertfront(" ");
insertfront("@");
insertfront("%");
insertfront("!");
insertfront("#");
insertfront("^");
insertfront("&");
insertfront("(");
insertfront(")");
insertfront("-");
insertfront("~");
insertfront("`");
insertfront("+");
insertfront("/");
insertfront("{");
insertfront("}");
insertfront("[");
```

```
        insertfront("]");
        insertfront("");
        insertfront("'");
        insertfront(";");
        insertfront(":");
        insertfront(",");
        insertfront(".");
        insertfront(">");
        insertfront("<");
        insertfront("?");
        insertfront("|");
        insertfront("=");
    }
}
```

SAMPLE OUTPUT

Parenthesis checking

```
Enter your choice:
1. brackets checking
2. while loop checking
3. for loop checking
4. do while loop checking
5. if statement checking
6. keyword checker
7. datatype checker
8. identifier checker
9. exit
1
Enter your expression to check brackets:
())

Mismatch found
()) == false
```

```
Enter your choice:
1. brackets checking
2. while loop checking
3. for loop checking
4. do while loop checking
5. if statement checking
6. keyword checker
7. datatype checker
8. identifier checker
9. exit
1
Enter your expression to check brackets:
()()
()() == true
```

While loop checking

```
Enter your choice:
1. brackets checking
2. while loop checking
3. for loop checking
4. do while loop checking
5. if statement checking
6. keyword checker
7. datatype checker
8. identifier checker
9. exit
2
Enter your while expression:
while(x!=6)

No error!
```

```
Enter your choice:
1. brackets checking
2. while loop checking
3. for loop checking
4. do while loop checking
5. if statement checking
6. keyword checker
7. datatype checker
8. identifier checker
9. exit
2
Enter your while expression:
whilex>5)

Opening parenthesis absent after 'while' keyword
```

For loop checking

```
Enter your choice:
1. brackets checking
2. while loop checking
3. for loop checking
4. do while loop checking
5. if statement checking
6. keyword checker
7. datatype checker
8. identifier checker
9. exit
3
Enter your for expression:
for(i=0i>10;i++)

Semicolon Error
```

```
Enter your choice:
1. brackets checking
2. while loop checking
3. for loop checking
4. do while loop checking
5. if statement checking
6. keyword checker
7. datatype checker
8. identifier checker
9. exit
3
Enter your for expression:
for(i=0;i>10;i++)

No error!
```

Do while loop

```
Enter your choice:
1. brackets checking
2. while loop checking
3. for loop checking
4. do while loop checking
5. if statement checking
6. keyword checker
7. datatype checker
8. identifier checker
9. exit
4
Enter your do-while expression:
do{}while(x>5)

Missing semicolon after while

Error in 'while' keyword usage
```

```
Enter your choice:
1. brackets checking
2. while loop checking
3. for loop checking
4. do while loop checking
5. if statement checking
6. keyword checker
7. datatype checker
8. identifier checker
9. exit
4
Enter your do-while expression:
do{}while(x>5);

No error!
```

If statement checking

```
Enter your choice:
1. brackets checking
2. while loop checking
3. for loop checking
4. do while loop checking
5. if statement checking
6. keyword checker
7. datatype checker
8. identifier checker
9. exit
5
Enter your if expression:
uf(x==3)

Error in 'if' keyword usage
```

```
Enter your choice:
1. brackets checking
2. while loop checking
3. for loop checking
4. do while loop checking
5. if statement checking
6. keyword checker
7. datatype checker
8. identifier checker
9. exit
5
Enter your if expression:
if(x==3)

No error!
```

Keyword checking

```
Enter your choice:
1. brackets checking
2. while loop checking
3. for loop checking
4. do while loop checking
5. if statement checking
6. keyword checker
7. datatype checker
8. identifier checker
9. exit
6
Enter your keyword:
hello

'hello' is not a keyword
```

```
Enter your choice:
1. brackets checking
2. while loop checking
3. for loop checking
4. do while loop checking
5. if statement checking
6. keyword checker
7. datatype checker
8. identifier checker
9. exit
6
Enter your keyword:
void

'void' is a keyword!
```

Data type checking

```
Enter your choice:
1. brackets checking
2. while loop checking
3. for loop checking
4. do while loop checking
5. if statement checking
6. keyword checker
7. datatype checker
8. identifier checker
9. exit
7
Enter your datatype:
class

'class' is not a datatype
```

```
Enter your choice:
1. brackets checking
2. while loop checking
3. for loop checking
4. do while loop checking
5. if statement checking
6. keyword checker
7. datatype checker
8. identifier checker
9. exit
7
Enter your datatype:
boolean

'boolean' is a datatype!
```

Identifier Checking

```
Enter your choice:
1. brackets checking
2. while loop checking
3. for loop checking
4. do while loop checking
5. if statement checking
6. keyword checker
7. datatype checker
8. identifier checker
9. exit
8
Enter your identifier:
abstract

Keyword 'abstract' is used, therefore it isn't a valid identifier
```

```
Enter your choice:
1. brackets checking
2. while loop checking
3. for loop checking
4. do while loop checking
5. if statement checking
6. keyword checker
7. datatype checker
8. identifier checker
9. exit
8
Enter your identifier:
pen

'pen' is a valid identifier!
```

Exit

```
Enter your choice:
1. brackets checking
2. while loop checking
3. for loop checking
4. do while loop checking
5. if statement checking
6. keyword checker
7. datatype checker
8. identifier checker
9. exit
9
Invalid Input!
```