PROJECT REPORT

CS-214 Data Structures and Algorithms

Submitted to:

Dr. Mohammad Imran

Submitted by:

190787 - Amal Abrar

190792 - Ali Taqi Wajid

COMPUTER PROGRAM SYNTAX CHECKER

Written in C++

==========

TheFlash2k#0407
Amal Abrar#3002

/TheFlash2k/SyntaxChecker/

@alitaqiwajid

190792@students.au.edu.pk
190787@students.au.edu.pk

Table of Contents

Abstract

A simple C++ program that takes user input or
verifies the syntax
of a program written in a file.

Introduction:

This project was assigned to us as a semester-end project. The main goal of this project was to test whether a user-defined file has correct syntax or not. In short, our goal was to create an interpreter, kind of, but not really. We were told to use a stack data structure for verification of parenthesis and keywords.

Work done by each group member:

Amal Abrar - 190787:

Complete Implementation of Class Node and Stack. Design for parenthesis matching algorithm and Flow Chart and implementation of UML class Diagram.

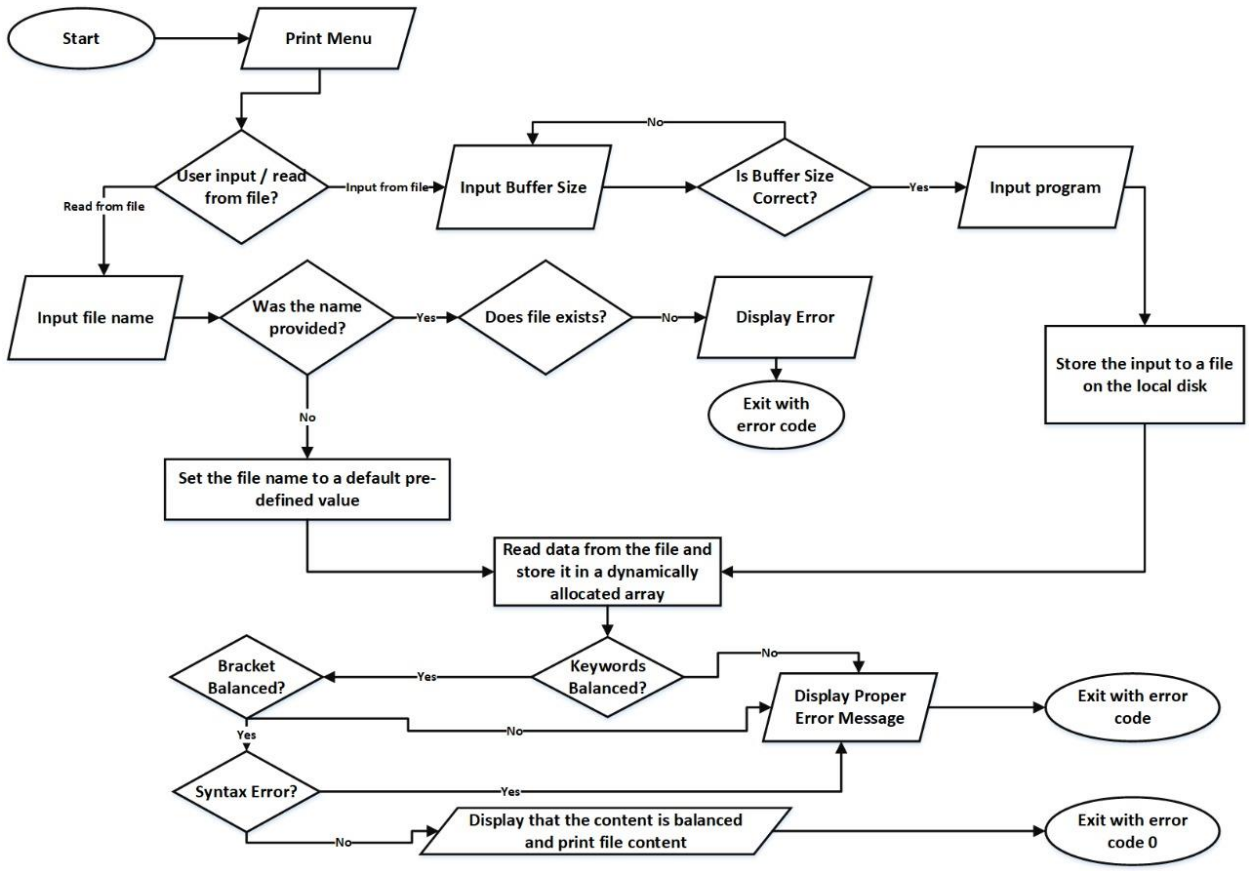
Ali Taqi Wajid - 190792:

Implementation of the complete algorithm. Separation of each class into their specific header and implementation files. Design, Implementation and development of main function and writing the report.

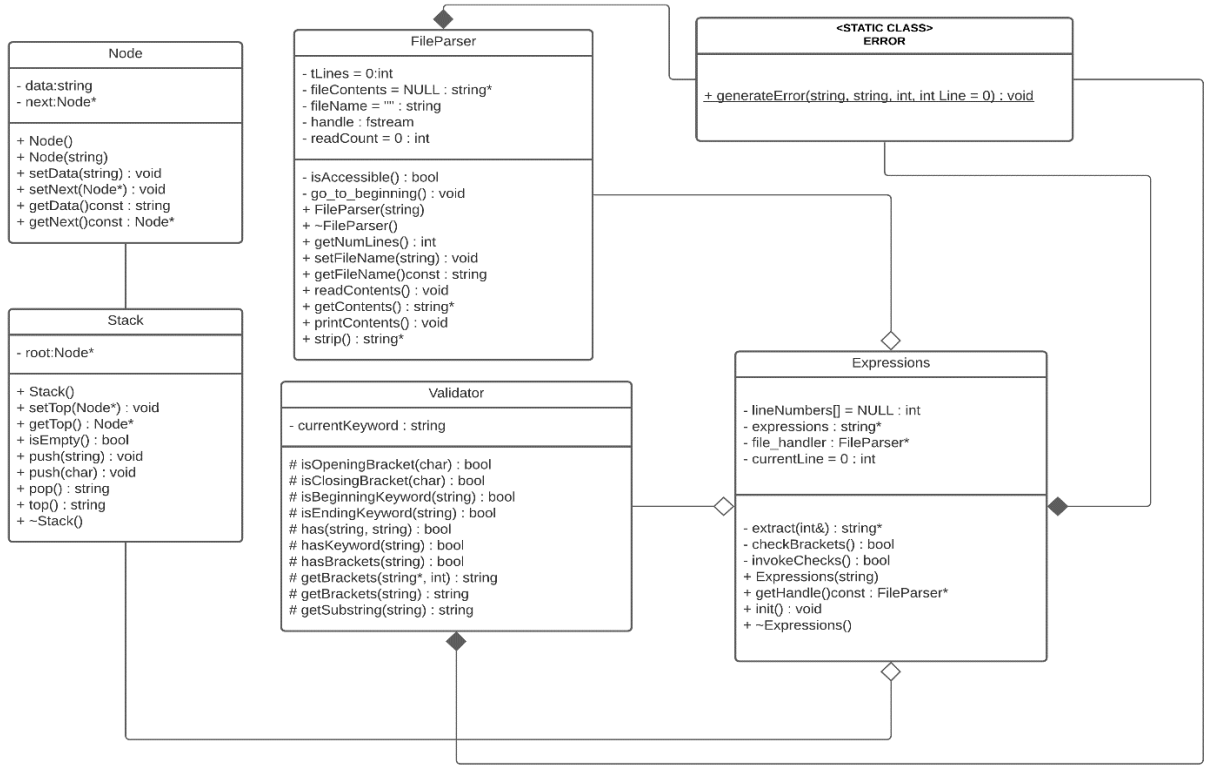
Data Structures OR Algorithm used:

We decided to use **"STACK"** and **"Arrays/Dynamic Arrays"** as the data structures in our programs. One of the most important functions of the stack is when we use it to match parenthesis. We decided to also use "**Hashmaps**" so that we'll have the line number as "key" and keyword as the "value". But, it was really complicating our program so we decided to ignore that and use 2 different arrays for the essentially doing the same task.

Flow Chart:



UML Class Diagram:



Uses of each Class:

- `class Node`:
  Is used to create the stack data structure.
- `class Stack:`
  Contains the methods and implementation of the "**Stack**" data structure.
- `class FileParser:`
  Contains all the methods that are necessary for file handling operation in our project.
- `class Validator:`
  Contains multiple methods for verification such as verifies if the passed argument is an opening/closing bracket OR beginning/ending keyword. Also, it contains methods to get substrings either out of a string array or just a string.
- `class Expressions:`
  This is the main class that contains the main algorithm implemented in it and is a child class of `class Validator.` It has 2 important methods, one to check if the brackets are balanced. Other to check if the keywords are balanced. It contains a few more methods.
- `class ERROR:`
  This class is a static class and cannot be instantiated. This class only contains a single static method which we will call whenever an error occurs.

class Node:

The class Node contains the following methods and attributes:

- Attributes:
  1) `std::string data; // Will contain the actual keywords/braces that need to be verified.`
  2) `Node* next; // Is a pointer to the next node in the stack.`
- Methods:
  1) `void setData(std::string);`
  2) `void setNext(Node*);`
  3) `Node* getNext()const;`
  4) `std::string getData()const;`

class Stack:

The class Stack is the complete implementation of Stack data structure with the push pop buttons and some minor features to enhance workflow and debugging. If we were to use the built-in stack, our program would work the same. We'd just have to change the declaration syntax. I was thinking about implementing a stack using templates but that was too much work and debugging would've gotten really hard. So, I decided to create a Node class having a single string type data. So, it has the following attributes and methods:

- Attributes:
  1) `Node* root; // An object of above mentioned class Node;`
- Methods:
  1) `void setTop(Node*); // Setter / Mutator`
  2) `Node* getTop(); // Getter / Accessor`
  3) `Node* getTop(); // Getter / Accessor`
  4) `bool isEmpty(); // Checks if the stack is empty and returns a boolean value`
  5) `void push(std::string); // Pushes a string onto the stack`
  6) `void push(char); // Typecasts a char into a string and pushes it onto the stack`
  7) `std::string pop(); // Removes the value at the top of the stack and also returns it.`
  8) `std::string top(); // Returns the value on top of the stack.`
  - There is another method called
    - `void print(); // Prints all the contents within the stack.`

    This is a debugging method which prints all the contents within the stack.

class FileParser:

This file contains methods to read file contents, store the file contents in a string array, get the number of lines in a file, returns a handle to a file, checks if the file is available etc. Proper error handling has also been done in this class. This class has a special method called "split()"which deletes any extra spaces before a keyword or after and trims it. It has the following attributes and methods:

- Attributes:
  1) `int tLines = 0; // Will contain the number of lines in the file.`
  2) `std::string* fileContents = NULL; // Pointer used to dynamically allocate an array later.`
  3) `std::string fileName = ""; // Contains the name of the file.`
  4) `std::fstream handle; // A handle to the file.`
  5) `int readCount = 0; // Keeps number of times the file has been read. This helps in getContents() method.`

- Methods:
  - Private Methods:

```
1) bool isAccessible();// Checks whether the file exists/ is accessible or not
2) void go_to_beginning(); // This method takes the pointer to the beginning of
   the file.as
```

  - Public Methods:

```
3) int getNumLines(); // Gets the number of lines in the file
4) void setFileName(std::string); // Sets the file name
5) std::string getFileName()const; // Returns the name of the file
6) void readContents(); // Reads the contents from the file
7) std::string* getContents(); // Returns a pointer to the contents
8) void printContents(); // Prints all the contents to the screen.
9) std::string* strip(); // Removes unnecessary whitespaces and returns a
   pointer
```

class Validator:

This class is used to validate multiple parameters. All the methods in the class are set to protected so that only the classes inherited from this class can use these methods but not any public class. This class contains the following methods and attributes:

- Attributes:
  This method doesn't contain any attribute. I originally added a `std::string currentWord;` But decided to remove it as I found a much better approach to tackle the issue we were facing.
- Methods:

```
1) bool isOpeningBracket(char); // Checks if the passed argument is an opening
   bracket.
2) bool isClosingBracket(char);// Checks if the passed argument is an closing
   bracket.
3) bool isBeginningKeyword(std::string); // Checks if the passed argument
   contains a beginning keyword.
4) bool isEndingKeyword(std::string); // Checks if the passed argument contains
   an ending keyword.
5) bool has(std::string, std::string); // Checks if the first argument contains
   a string equivalent to the second argument.
6) bool hasKeyword(std::string); // Checks if a string has any one of the pre-
   defined keywords
7) bool hasBrackets(std::string); // Checks if a string contains brackets.
8) std::string getBrackets(std::string*, int); // Takes two arguements, one is
   the string array and the other is the size of that array and returns all the
   braces as a single string.
9) std::string getBrackets(std::string); // Returns all the existing braces
   within a single string as a string.
10) std::string getSubstring(std::string str); // Returns a pre-defined keyword
    existing in the passed argument.
```

class Expressions:

This class is a child class of `class Validator`. This class contains all the most important algorithms such as checking if the brackets in the statements match or not. Then, checking if the keywords have proper endings or not. If not, proper error handling is also done using `class ERROR`. It contains only 2 public methods (besides the constructor and the destructor. One is used to begin the process of

checking. And the other returns a pointer of FileParser type to deal with methods within file such as printing the file(done in main) etc. It has the following methods and attributes:

- Attributes:
  1) `int lineNumbers[MAX] = { NULL }; // Will contain the line numbers at which certain data exists.`
  2) `std::string* expressions; // Reads all the data from the file and stores into this pointer.`
  3) `FileParser* file_handler; // An object of FilerParser that we will dynamically allocate with the fileName later on.`
  4) `int currentLine = 0; // An index which tells us the number upto which lineNumbers array has been filled.`
- Methods:
  - Private Methods:
  1) `bool checkBrackets(); // The main method that checks if all the brackets are balanced.`
  2) `bool invokeChecks(); // This is the method that will check if the keywords are balanced, if not, will throw proper errors.`
  - Public Methods:
  3) `FileParser* getHandle()const; // Returns a handle to the FileParser class.`
  4) `void init(); // The public method that can be used to initialize everything check.`

class ERROR:

This is a static and only contains a single static method that can be used to display error, the error type, error description, error code and if an error exists on a line. One of the key features of this class is that it cannot be instantiated.

- Method:

```
static void generateError(std::string error_type, std::string error_message,
int error_code, int line_number = 0);
```

END:

After completing this project, both of us are feeling confident going into the next semester and having new tools in our arsenal such as stacks and queues. This course has been one of the best courses related to programming so far in our university life.

EOF