SYNTAX CHECKER

A MINI PROJECT REPORT

18CSC304J - Compiler Design

Submitted by

Harshitha G [RA2011030010020]

Devshree Moghe [RA2011030010049]

Under the guidance of

Dr. B Yamini

Assistant Professor, Department of Networking and Communication

In Partial Fulfillment of the Requirements for the Degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE ENGINEERING
with specialization in CYBER SECURITY

DEPARTMENT OF NETWORKING AND COMMUNICATIONS

COLLEGE OF ENGINEERING AND TECHNOLOGY SRM

INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR- 603 203

MAY 2023

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that this mini project report titled “Syntax Checker” is the bonafide work done by Harshitha G [RA2011030010020] and Devshree Moghe [RA2011030010049] who carried out the mini project work and Laboratory exercises under my supervision for **18CSC304J - Compiler Design**. Certified further, that to the best of my knowledge the work reported herein does not form part of any other work.

Dr. B Yamini
ASSISTANT PROFESSOR
Department of Networking and Communications

Dr. Annapurani Panaiyappan.K
HEAD OF THE DEPARTMENT
Department of Networking and Communications

Signature of the Internal Examiner-I

Signature of the Internal Examiner-II

TABLE OF CONTENTS

ABSTRACT

I. A syntax checker, also known as a parser, is a key component in the process of compiling a programming language. It is responsible for analyzing the structure of the source code and determining if it conforms to the syntax rules of the language.

II. A syntax checker works by breaking down the input source code into a series of tokens, such as keywords, identifiers, operators, and punctuation. It then uses a set of rules defined by the language's grammar to analyze the relationships between the tokens and determine if the code is syntactically correct.

III. If the syntax checker identifies an error in the code, it should provide an appropriate error message indicating the location of the error and the nature of the problem.

IV. To implement a syntax checker, you would typically use a parser generator tool, such as PLY (Python Lex-Yacc) or ANTLR. These tools allow you to define the syntax of the programming language using a set of rules and generate a parser that can be used to check the syntax of input source code..

LIST OF ABREVIATION

AST	Abstract Syntax tree
CFG	Control flow graph
I/O	Input/Output
PEG	Parsing expression grammar
AST-visitor	Abstract syntax tree visitor
IR	Intermediate representation
LR(k)	Left-to-right k-token lookahead

INTRODUCTION

A syntax checker is a critical component in the process of compiling a programming language. It is used to check the input source code for errors and ensure that it conforms to the syntax rules of the language. Here are some reasons why a syntax checker is important in compiler design:

1. Error Detection: A syntax checker helps to identify errors in the input source code, such as missing or incorrect punctuation, improper use of keywords and operators, and other syntax-related issues. By detecting these errors early on, the syntax checker can save developers time by preventing them from having to manually search through their code for errors.

2. Improved Code Quality: By enforcing the syntax rules of the language, a syntax checker helps to ensure that the input source code is of high quality and adheres to industry best practices. This can help to prevent errors and bugs in the compiled code and ultimately lead to a better product.

3. Consistency: A syntax checker helps to enforce consistent coding practices across an organization or team. By ensuring that all code is written in a consistent style and adheres to the same syntax rules, the syntax checker can help to improve code maintainability and reduce confusion.

4. Efficiency: A syntax checker can help to speed up the compilation process by automatically detecting errors and preventing the need for manual error correction. This can help to save time and improve the overall efficiency of the development process.

Overall, using a syntax checker in compiler design is an essential step in ensuring that the input source code is syntactically correct and can be compiled into executable code. It helps to detect errors, improve code quality and consistency, and ultimately save time and improve efficiency.

MOTIVATION

A syntax checker is a critical component in compiler design as it helps to detect errors early, improve code quality, ensure consistency, save time and effort, and enhance the user experience.

Having a syntax checker in compiler design is essential for several reasons:

1. Detecting errors early: A syntax checker helps to detect errors in the input source code early in the development process. By identifying syntax errors early, developers can save time and effort that would otherwise be spent searching through their code for errors.

2. Improving code quality: By enforcing syntax rules and best practices, a syntax checker can help to improve the quality of the input source code. This can help to prevent errors and bugs in the compiled code and ultimately lead to a better product.

3. Ensuring consistency: A syntax checker can help to enforce consistent coding practices across an organization or team. By ensuring that all code is written in a consistent style and adheres to the same syntax rules, the syntax checker can help to improve code maintainability and reduce confusion.

4. Saving time and effort: A syntax checker can help to speed up the development process by automatically detecting errors and preventing the need for manual error correction. This can help to save time and improve the overall efficiency of the development process.

5. Enhancing user experience: A syntax checker can provide appropriate error messages that help users to identify and correct errors in their code. This can help to enhance the overall user experience and improve user satisfaction.

6. Education: Syntax checkers can be a valuable educational tool for developers, helping them learn about language syntax and best practices, and guiding them towards writing better code.

Syntax checkers are used to improve the quality and correctness of source code in programming. They are an essential tool in the software development process because they can help detect and correct errors in code before it is compiled or executed.

Overall, syntax checkers are an important tool for improving the quality and correctness of code, and can help save time, effort, and resources in the software development process.

ALGORITHM FOR SYNTAX CHECKER

The algorithm for a syntax checker typically involves the following steps:

1. Read input source code from a file or other source.

2. Tokenize the input source code into a series of tokens, using a lexical analyzer or scanner.

3. Pass the tokens to the parser, which applies a set of syntax rules to determine if the code is syntactically correct.

4. If the parser identifies an error in the code, it generates an appropriate error message indicating the location of the error and the nature of the problem. The error message should be clear and easy to understand, so that the user can easily identify and correct the error.

5. If an error is detected, the parser attempts to recover from the error and continue parsing the input source code. This may involve skipping over erroneous input or inserting missing tokens.

6. Once the entire input source code has been parsed, the parser generates a parse tree that represents the structure of the code.

7. If no errors are detected, the parse tree can be used for further processing, such as semantic analysis or code generation.

8. Output the results of the syntax checking process, including any error messages and the parse tree.

PROPOSED METHOD FOR CREATING A SYNTAX CHECKER

Creating a syntax checker involves the following steps:

1. Choose a programming language: You need to choose a programming language that you will use to implement the syntax checker. The choice of language depends on your familiarity and comfort with the language, as well as the specific features you need to implement the syntax checker.

2. Define the grammar: Define the grammar for the language you want to check. This involves specifying the syntax rules that the syntax checker will use to analyze the input source code.

3. Implement the scanner: The scanner reads the input source code and converts it into a series of tokens. The scanner should identify each token and its type (such as a keyword, identifier, operator, or punctuation mark).

4. Implement the parser: The parser uses the grammar to analyze the tokens and determine if the input source code is syntactically correct. The parser should check that the code follows the specified syntax rules, and generate an appropriate error message if it encounters an error.

5. Implement error recovery: If the parser encounters an error, it should attempt to recover from the error and continue checking the rest of the input source code. This may involve skipping over erroneous input or inserting missing tokens.

6. Implement the output: The syntax checker should output the results of the syntax checking process, including any error messages and the parse tree.

7. Test the syntax checker: Test the syntax checker with various input source code to ensure that it correctly detects syntax errors and generates appropriate error messages.

8. Refine the syntax checker: Make any necessary adjustments to the syntax checker based on feedback and testing.

Overall, creating a syntax checker involves defining the grammar, implementing the scanner and parser, handling errors and recovery, generating output, testing, and refining. It can be a complex task, but with careful planning and attention to detail, you can create a powerful and effective syntax checker for your chosen programming language.

DESCRIPTION OF MODULES

For a syntax checker in C++, some of the modules required are:

1. Lexical analyzer (scanner): This module reads the input C++ source code and breaks it down into a series of tokens, such as keywords, identifiers, operators, and punctuation.

2. Syntax analyzer (parser): This module analyzes the relationships between the tokens and determines if the code is syntactically correct. This involves using a set of rules defined by the C++ grammar to identify the syntax structure of the code.

3. Error handling module: This module is responsible for detecting and handling errors that occur during the syntax checking process. It generates appropriate error messages indicating the location of the error and the nature of the problem.

4. Symbol table: This module maintains a symbol table that stores information about variables, functions, and other symbols used in the input source code. The symbol table is used by the syntax checker to ensure that variables are declared before they are used, and to verify that functions are called with the correct number and types of arguments.

5. Intermediate code generation module: This module generates intermediate code that can be used for further processing, such as semantic analysis or code generation.

6. Output module: This module outputs the results of the syntax checking process, including any error messages and the parse tree.

- In addition to the above modules, a syntax checker for C++ may also need to include support for the various language features and constructs, such as

templates, namespaces, and exception handling. It may also need to include support for preprocessor directives such as #include and #define, which can significantly alter the structure of the input code.

- Overall, the modules required for a syntax checker in C++ are similar to those required for a syntax checker for other programming languages. However, the specific requirements for C++ syntax checking may vary depending on the features and constructs used in the input source code.

REQUIREMENTS TO RUN SCRIPT

The requirements to run a syntax checker can depend on the specific implementation of the syntax checker, the programming language being checked, and the features of the input source code. However, some general requirements to run a syntax checker are:

1. A computer: A syntax checker runs on a computer, which must meet the minimum hardware and software requirements of the syntax checker. This typically includes a CPU, RAM, and a storage device.

2. A programming language: The syntax checker is designed to check the syntax of a specific programming language. Therefore, the computer must have a compiler or interpreter for that programming language installed.

3. Input source code: The syntax checker requires the input source code to check for syntax errors. This can be provided as a file or as input from the user.

4. Enough free memory and disk space: The input source code can be quite large, and the syntax checker may require significant memory and disk space to operate. Therefore, the computer should have enough free memory and disk space available to run the syntax checker.

5. Any specific libraries or dependencies: Depending on the implementation of the syntax checker and the features of the input source code, there may be specific libraries or dependencies required to run the syntax checker. These should be installed and configured correctly to ensure the syntax checker runs properly.

Overall, the requirements to run a syntax checker can vary depending on the implementation and the programming language being checked. However, a computer, a programming language, input source code, enough free memory and disk space, and any specific libraries or dependencies are some of the general requirements needed to run a syntax checker.

CODE

```
#include <iostream>
#include <string>
#include <stack>

bool check_syntax(const std::string& code)
{    std::stack<char> stk;    for (char c :
code) {        if (c == '(' || c == '[' || c == '{')
{            stk.push(c);
       } else if (c == ')' || c == ']' || c == '}')
{            if (stk.empty()) {                return
false;
           }
           char top = stk.top();            stk.pop();            if ((c == ')' && top != '(') || (c
== ']' && top != '[') || (c == '}' && top != '{')) {                return false;
           }
       }
   }
   return stk.empty();
}

int main() {    std::string code;
std::cout << "Enter code: ";
std::getline(std::cin, code);    if
(check_syntax(code)) {        std::cout
<< "Syntax is correct!\n";
   }


 else {        std::cout << "Syntax is
incorrect!\n";
   }
   return 0;
}
```

INPUT

Enter Code: 12+24

OUTPUT

main.cpp

Run

```cpp
#include <iostream>
#include <string>
#include <stack>

bool check_syntax(const std::string& code) {
    std::stack<char> stk;
    for (char c : code) {
        if (c == '(' || c == '[' || c == '{') {
            stk.push(c);
        } else if (c == ')' || c == ']' || c == '}') {
            if (stk.empty()) {
                return false;
            }
            char top = stk.top();
            stk.pop();
            if ((c == ')' && top != '(') || (c == ']' && top !=
                '[') || (c == '}' && top != '{')) {
                return false;
```

main.cpp

Run

```
                [ ) || (c == '}' && top != '{')) {
17                  return false;
18              }
19          }
20      }
21      return stk.empty();
22  }
23
24  int main() {
25      std::string code;
26      std::cout << "Enter code: ";
27      std::getline(std::cin, code);
28      if (check_syntax(code)) {
29          std::cout << "Syntax is correct!\n";
30      } else {
31          std::cout << "Syntax is incorrect!\n";
32      }
33      return 0;
34  }
```

Output

```
/tmp/HUmNNPb0AX.o
Enter code: 12+24
Syntax is correct!
```

Output: Syntax is correct

Enter Code: (4

```
/tmp/HUmNNPb0AX.o
Enter code: (4
Syntax is incorrect!
```

Output: Syntax is incorrect!

RESULT

The result of a syntax checker project in compiler design is a software tool that can check the syntax of input source code and detect any syntax errors. The syntax checker analyzes the input source code according to the rules defined by the programming language grammar, and generates appropriate error messages indicating the location and nature of any syntax errors.

1. A standalone software tool that can be used by developers to check the syntax of their C++ source code. This tool can be integrated into an IDE or text editor to provide real-time syntax checking and error highlighting.

2. A component of a larger compiler or interpreter project, which checks the syntax of the input source code as part of a larger compilation or interpretation process.

3. A command-line tool that can be used to check the syntax of C++ source code files in batch mode. This tool can be used as part of a build process or automated testing system to ensure that all source code files are syntactically correct.

4. A web-based syntax checker that can be accessed through a web browser. This tool can be used by developers who do not have a local C++ compiler or IDE installed, or who want to check their code from a remote location.

Overall, the outcome of a syntax checker project in compiler design is a software tool that helps developers ensure that their C++ source code is syntactically correct and free of errors. This tool can save developers time and effort by identifying syntax errors early in the development process, before the code is compiled or executed.

CONCLUSION

Syntax checkers have several benefits for software development, including:

1. Early detection of syntax errors: Syntax checkers can detect syntax errors early in the development process, before the code is compiled or executed. This saves time and effort in the development process, as developers can fix errors before they become more complex and difficult to debug.

2. Improved code quality: Syntax checkers help ensure that code is written according to the rules defined by the programming language grammar, leading to improved code quality and maintainability.

3. Reduced errors: By detecting syntax errors early, syntax checkers can help reduce errors in the final software product. This leads to a more stable and reliable product.

4. Faster development: Syntax checkers can save time and effort in the development process by detecting errors early and reducing the time required for manual testing and debugging.

5. Improved collaboration: Syntax checkers can improve collaboration among developers by providing a consistent standard for code formatting and style. This leads to a more uniform codebase and easier collaboration among team members.

6. Improved security: Syntax checkers can help prevent security vulnerabilities by detecting potentially dangerous code patterns, such as buffer overflows or SQL injection attacks.

Overall, syntax checkers provide numerous benefits for software development, including early detection of syntax errors, improved code quality, reduced errors, faster development, improved collaboration, and improved security. By using a syntax checker as part of the development process, developers can improve the quality and reliability of their software products, leading to increased customer satisfaction and a competitive advantage in the marketplace.

In conclusion, a syntax checker is an important component of a compiler or interpreter that checks the syntax of input source code and detects any syntax errors. The syntax checker analyzes the input source code according to the rules defined by the

programming language grammar, and generates appropriate error messages indicating the location and nature of any syntax errors.

The syntax checker project in compiler design involves designing and implementing a software tool that can check the syntax of input source code written in a specific programming language. This project requires an understanding of the programming language grammar, as well as knowledge of algorithms and data structures for parsing and syntax analysis.

The outcome of a syntax checker project can be a standalone software tool, a component of a larger compiler or interpreter project, a command-line tool, or a webbased tool. The tool can help developers ensure that their source code is syntactically correct and free of errors, thus saving time and effort in the development process.

Overall, a syntax checker is a crucial part of the compilation or interpretation process, and its development is an important task in compiler design. A well-designed syntax checker can help improve code quality, reduce errors, and make the development process more efficient.

REFERENCES

https://stackoverflow.com/

https://www.geeksforgeeks.org/type-checking-in-compiler-design/

https://www.tutorialspoint.com/compiler_design/compiler_design_syntax_analysis.htm

https://www.brainkart.com/article/Type-Checking_8086/

https://www.guru99.com/syntax-analysis-parsing-types.html

https://chat.openai.com/

https://www.datatrained.com/post/type-checking-in-compiler-design/