**Compilers & Languages**

**Compiler Implementation**

Project - Final Report

**Team 4**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Sec** | **BN** | **ID** | **Email** |
| بموا عريان عياد | 1 | 17 | 9202391 | bemoi.tawadros00@eng-st.cu.edu.eg |
| مارك ياسر نبيل | 2 | 14 | 9203106 | mark.ibrahim00@eng-st.cu.edu.eg |
| بيتر عاطف فتحي | 1 | 18 | 9202395 | peter.zaki00@eng-st.cu.edu.eg |
| كريم محمود كمال | 2 | 12 | 9203076 | karim.mohamed003@eng-st.cu.edu.eg |

**Supervision**

**Dr. Ayman Abo El Hassan**

**Eng. Omar Samir Galal**

**Project Overview**

In our project, we're building a compiler for a language that's similar to C/C++. We're using Lex and Yacc to make this happen. Our main goals are to break down the source code into small chunks called tokens, create a structured code's grammar, carefully check the meaning of the code, and finally, produce a kind of code that's easier for computers to work with.

**Tools & Technologies**

Lexical Analysis: Flex

Parsing: Bison

GUI: Flask

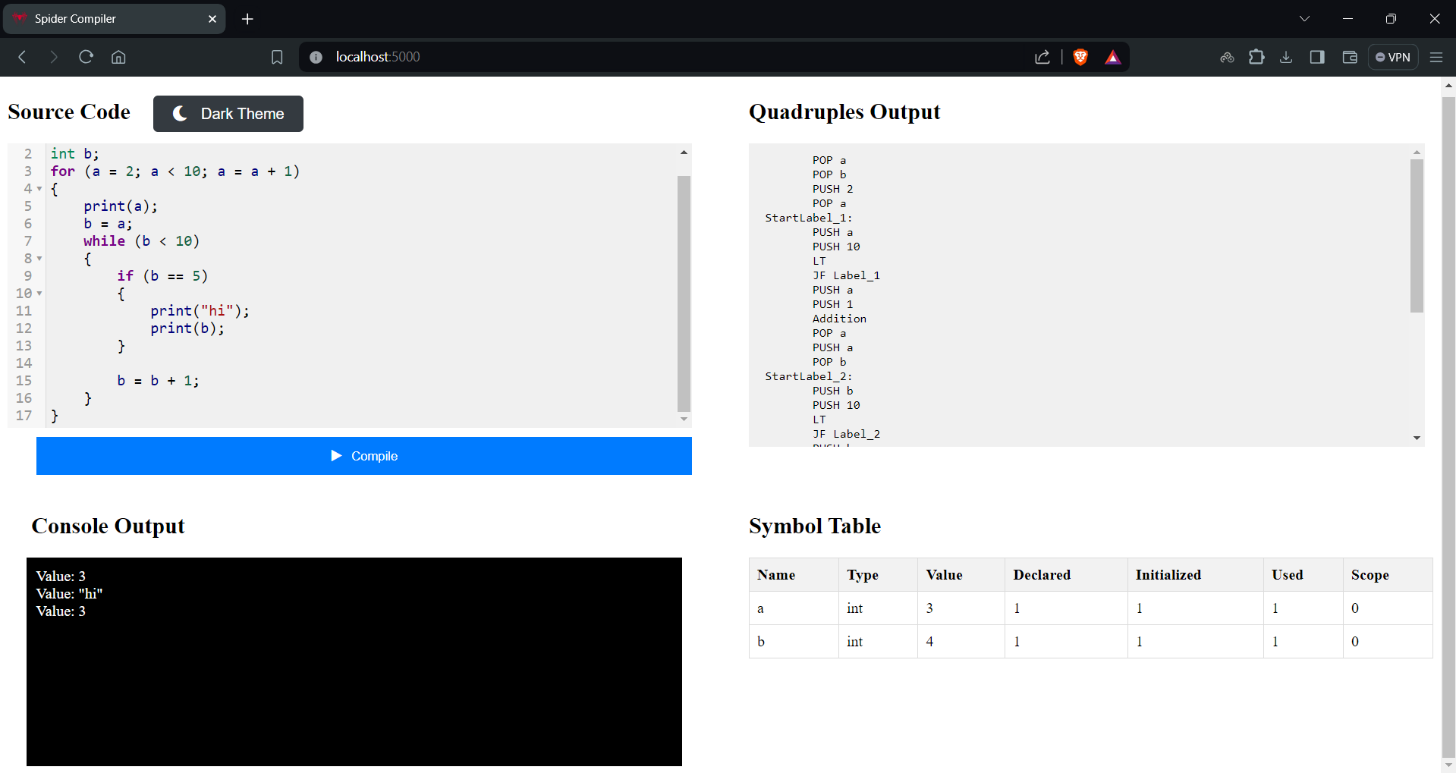
**List of tokens and a description of each**

|  |  |
| --- | --- |
| **Token** | **Description** |
| **PRINT** | Represents the "print" command |
| **CONSTANT** | Indicates a constant value |
| **EXIT** | Represents the "exit" command |
| **BOOL\_DATA\_TYPE** | Denotes the Boolean data type |
| **STRING\_DATA\_TYPE** | Denotes the string data type |
| **INTEGER\_DATA\_TYPE** | Denotes the integer data type |
| **FLOAT\_DATA\_TYPE** | Denotes the floating-point data type |
| **VOID\_DATA\_TYPE** | Denotes the void data type |
| **IF** | Indicates the "if" control flow statement |
| **ELSE** | Indicates the "else" control flow statement |
| **FOR** | Indicates the "for" control flow statement |
| **WHILE** | Indicates the "while" control flow statement |
| **REPEAT** | Indicates the "repeat" control flow statement |
| **UNTIL** | Indicates the "until" control flow statement |
| **SWITCH** | Indicates the "switch" control flow statement |
| **CASE** | Indicates the "case" control flow statement |
| **DEFAULT** | Indicates the "default" control flow statement |
| **CONTINUE** | Indicates the "continue" control flow statement |
| **BREAK** | Indicates the "break" control flow statement |
| **RETURN** | Indicates the "return" control flow statement |
| **ENUM** | Indicates the "Enum" keyword |
| **SHIFT\_LEFT** | Represents the bitwise left shift operator |
| **SHIFT\_RIGHT** | Represents the bitwise right shift operator |
| **LT** | Represents the less than comparison operator |
| **GT** | Represents the greater than comparison operator |
| **LEQ** | Represents the less than or equal to comparison operator |
| **GEQ** | Represents the greater than or equal to comparison operator |
| **EQ** | Represents the equality comparison operator |
| **NEQ** | Represents the inequality comparison operator |
| **AND** | Represents the logical AND operator |
| **OR** | Represents the logical OR operator |
| **NOT** | Represents the logical NOT operator |
| **Bitwise Operator** | Represents various bitwise operators |
| **End Of Statement** | Represents the end of a statement |
| **Arithmetic Operator** | Represents various arithmetic operators |
| **Punctuators** | Represents various punctuators like parentheses, braces, etc. |
| **TRUE\_VALUE** | Represents the Boolean value true |
| **FALSE\_VALUE** | Represents the Boolean value false |
| **Identifier** | Represents an identifier (variable name) |
| **Integer Value** | Represents an integer value |
| **Float Value** | Represents a floating-point value |
| **String Value** | Represents a string value |
| **In Line Comment** | Indicates an inline comment |
| **Multi Line Comment** | Indicates a multi-line comment |
| **Un Expected Token** | Represents an unexpected token, triggering an error message |

**List of quadruples and a description of each**

|  |  |
| --- | --- |
| **Quadruple** | **Description** |
| **PROC function\_name** | Marks the beginning of a function with the specified name |
| **ENDPROC function\_name** | Marks the end of a function with the specified name |
| **CALL function\_name** | Calls a function with the specified name |
| **RET** | Indicates a return from a function |
| **PUSH value** | Pushes a value onto the stack |
| **POP symbol** | Pops a value from the stack and assigns it to the specified symbol |
| **JMP Label** | Unconditionally jumps to the specified label |
| **JF Label\_number** | Jumps to the specified label if the top of the stack is false |
| **ENUM enum\_name** | Marks the beginning of an Enum block with the specified name |
| **ENDENUM enum\_name** | Marks the end of an Enum block with the specified name |
| **NEGATIVE** | Represents the negation operation (unary minus) |
| **NOT** | Represents the logical NOT operation |
| **Bitwise OR** | Represents the bitwise OR operation |
| **Bitwise AND** | Represents the bitwise AND operation |
| **Bitwise XOR** | Represents the bitwise XOR operation |
| **Addition** | Represents the addition operation |
| **Subtraction** | Represents the subtraction operation |
| **Multiplication** | Represents the multiplication operation |
| **Division** | Represents the division operation |
| **Modulus** | Represents the modulus operation |
| **Logical AND** | Represents the logical AND operation |
| **Logical OR** | Represents the logical OR operation |
| **Shift Left** | Represents the bitwise left shift operation |
| **Shift Right** | Represents the bitwise right shift operation |
| **EQ (Equal)** | Represents the equality comparison operation |
| **NEQ (Not Equal)** | Represents the inequality comparison operation |
| **GT (Greater Than)** | Represents the greater than comparison operation |
| **GEQ (Greater Than or Equal to)** | Represents the greater than or equal to comparison operation |
| **LT (Less Than)** | Represents the less than comparison operation |
| **LEQ (Less Than or Equal to)** | Represents the less than or equal to comparison operation |
| **Conversion** | Represents the casting or conversion operation |

|  |  |  |
| --- | --- | --- |
| **Test Case** | **Quadruples** | **Symbol Table** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**GUI**

**Completed Requirements**

* We designed a suitable programming language covering essential constructs:
  + Variables and Constants declaration
  + Mathematical and logical expressions
  + Assignment statements
  + If-then-else statement, while loops, repeat-until loops, for loops, switch statement
  + Block structure (nested scopes where variables may be declared at the beginning of blocks)
  + Functions
* We designed a suitable and extensible format for the symbol table.
* We implemented the lexical analyzer using Lex.
* We designed suitable action rules to produce the output quadruples and implemented parser using YACC.
* We implemented a proper syntax error handler.
* We built a simple semantic analyzer that checks for the following:
  + Variable declaration conflicts. i.e. multiple declarations of the same variable
  + Improper usage of variables regarding their type.
  + Variables used before being initialized and unused variables.
* [Bonus] We implemented a simple GUI
* [Bonus] We implemented the type conversion quadruples to match operators’ semantic requirements, i.e. converting integer to float, etc.

**Workload Distribution**

|  |  |
| --- | --- |
| **Name** | **Workload** |
| Peter Atef | Phase 1 + GUI |
| Bemoi Erian |
| Mark Yasser | Phase 2 |
| Karim Mahmoud Kamal |