Compilers & Languages Project

Team 3

Name	ID	SEC	BN
Asmaa Adel Abdelhamed kawashty	9202285	1	13
Samaa Hazem Mohamed Abdel-latif	9202660	1	31
Norhan Reda Abdelwahed Ahmed	9203639	2	31
Hoda Gamal Hamouda Ismail	9203673	2	33

Project Overview:

- CompilersCraft is a simple programming language compiler similar to C++.
- CompilersCraft is handling the most common data types like integers, floats, bools, char and strings.
- CompilersCraft is supporting:
    - Variables and Constants declaration.
    - Mathematical, logical, and bitwise expressions.
    - Assignment statements.
    - If-then-else statement, while loops, do-while loops, for loops, switch statement.
    - Block structure (nested scopes where variables may be declared at the beginning of blocks).
    - Functions.

Tools and Technologies used:

GCC:

GNU Compiler Collections which is used to compile mainly C and C++ language.

Flex:

Faster scanner generator like lex.

Bison:

Parser generator like yacc.

tkinter:

Python library for implementing GUI.

How to run:

Using terminal:

- flex lexer.l
- bison -d parser.y
- gcc *.c -o out.exe
- write your Code inside a file, for example in.txt
- .\out.exe in.txt

Using GUI:

- Open CMD in the GUI directory, type python gui.py
- Write your Code inside
- Press Compile!

Lexer :

Keywords:

- int, float, string, char, bool
- print, void, return, continue
- switch, break
- case, default
- if, else
- for, while, do
- true, false, and, or

Operators:

- +, -, *, /, %, ^
- |, &, <<, >>
- and, or, !
- ==, !=, >, <, >=, <=

Constants:

- [A-Z]+

Identifier:

- [a-zA-Z_][a-zA-Z0-9_]*

Parser:

Print:

- print("test");

Variables declaration:

- int a = 5; float y=3.4;

Constant declaration:

- Capital letters only
    - Example: int X=6;

Mathematical operations:

- +, -, *, /, %, ^
    - x = x+1;

Bitwise operations:

- |, &, <<, >>
    - x = x|y;

Logical operations:

- and, or, !
    - if (x and y){...}

Control expressions:

- ==, !=, >, <, >=, <=

Assignment statements

- Example: a = 5;

If-then-else statement:

```
if (x > 0) {
    print("x is positive");
  } else if (x < 0)  {
        print("x is negative");
  } else {
        print("x is zero");
  }
```

For Loop:

```
for (i = 0; i < 10; i = i + 1;) {
    print(i);
}
```

While Loop:

```
while x > 0 {
    x = x - 1;
}
```

Do-While Loop:

```
do {
    x = x - 1;
} while (x == 0);
```

Switch Statements

```
switch (x) {
```

```
    case 0: print("x is zero"); break;
    case 1: print("x is one"); break;
    default: print("x is neither zero nor one");
}
```

Block structure:

```
{
    int x;
    {
        int y;
    }
}
```

Functions:

```
int square(int num) {
    return num * num;
}
result = square(5);
```

Comments:

- Start line with '//'
    - Example:// this is a comment

Tokens:

DIGIT	[0-9]+
FLOAT	[0-9]+\.[0-9]+
IDENTIFIER	[a-zA-Z_][a-zA-Z0-9_]*
CHAR	\'[a-zA-Z0-9_]\'
STRING	\"[a-zA-Z0-9_]+\"

CONSTANT	[A-Z]+
DATA TYPES	int, float, double, string, char, bool
OTHERS TOKENS	print, void, return, switch, break, continue, case default, if, else, for, while, do
SPACE	[\t]+
NEW LINE	\n
COMMENT	\/\/[\s\t]*.*[\s\t]*
Mathematical Ops	%, +, -, *, /, ^
BITWISE Ops	|, &, <<, >>
Logical Ops	and, or, !
CONTROL Ops	==, !=, >, <, >=, <=
SEMICOLON	;
true	true
false	false
PUNCTUATION	[){}(:,]

Quadruples:

Logical Operations

LOGICAL_OR	Get the logical or of two numbers
LOGICAL_AND	Get the logical or of two numbers
EQ	Check if two numbers are equal
NEQ	Check if two numbers are not equal

GT	Check if the first number is greater than the second
GEQ	Check if the first number is greater than or equal the second
LT	Check if the first number is less than the second
LEQ	Check if the first number is less than or equal the second

Arithmetic Operations

NEG	pops the top operand from stack, applies neg operation, then pushes the result to stack.
ADD	pops the top 2 operands from stack, applies plus operation, then pushes the result to stack.
SUB	pops the top 2 operands from stack, applies minus operation, then pushes the result to stack.
MUL	pops the top 2 operands from stack, applies multiply operation, then pushes the result to stack.
DIV	pops the top 2 operands from stack, applies division operation, then pushes the result to stack.
POW	pops the top 2 operands from stack, applies power operation, then pushes the result to stack.
MOD	pops the top 2 operands from stack, applies modulus operation, then pushes the result to stack.

Bitwise Operations

BITWISE_OR	pops the top 2 operands from stack, applies bitwise or operation, then pushes the result to stack.
BITWISE_AND	pops the top 2 operands from stack,

	applies bitwise and operation, then pushes the result to stack.
SHL	pops the top 2 operands from stack, applies shift left operation, then pushes the result to stack.
SHR	pops the top 2 operands from stack, applies shift tight operation, then pushes the result to stack.

Branching & Jumps

JMP label	Unconditional jump to the label
JF label	Jumps to the label if the result of the last operation was false

Function

Function name	Start of a function
END function name	End of a function
CALL function name	Calls a function
RET	Return from a function.

Variables

PUSH value	Push to the stack frame
POP variable	Pop from the stack frame