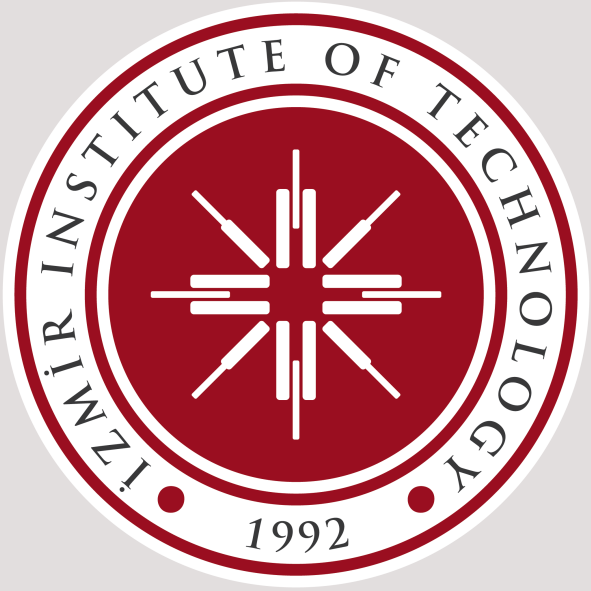
Izmir Institute of Technology
Computer Engineering Department
CENG513 Programming Assignment 2

Student Name: Gökay Gülsoy Student No: 270201072

March 30, 2024



List of Code

1 Adding New Tokens

In order to add support for recognizing and parsing repeat-until loop construct, I have modified the lexer[1] and parser[2] components of Kaleidoscope language. First modification I have done is on Token enum. I have introduced to new tokens which are **repeat** and **until** which can be seen in the code listing 1.

```
//==========================================
enum Token {
  tok_eof = -1,

  // commands
  tok_def = -2,
  tok_extern = -3,

  // primary
  tok_identifier = -4,
  tok_number = -5,
  // newly added tokens for supporting
  // repeat expression until expression construct
  tok_repeat = -6,
  tok_until = -7,
};
//==========================================
```

Code 1: Token for repeat and until added

2 Modifying Lexer

Second modification I have done is on lexer. I have added two new equality comparisons for new tokens repeat and until as can be seen in the code listing 2.

```
//==========================================
static int gettok() {
  static int LastChar = ' ';

  // Skip any whitespace.
  while (isspace(LastChar))
    LastChar = getchar();

  if (isalpha(LastChar)) { // identifier: [a-zA-Z][a-zA-Z0-9]*
    IdentifierStr = LastChar;
    while (isalnum((LastChar = getchar())))
      IdentifierStr += LastChar;

    if (IdentifierStr == "def")
      return tok_def;
    if (IdentifierStr == "extern")
      return tok_extern;

```

```
19      // modifying lexer to recognize
20      // repeat and until tokens
21      if (IdentifierStr == "repeat")
22        return tok_repeat;
23
24      if (IdentifierStr == "until")
25        return tok_until;
26
27      return tok_identifier;
28    }
29
30    if (isdigit(LastChar) || LastChar == '.') { // Number: [0-9.]+
31      std::string NumStr;
32      do {
33        NumStr += LastChar;
34        LastChar = getchar();
35      } while (isdigit(LastChar) || LastChar == '.');
36
37      NumVal = strtod(NumStr.c_str(), nullptr);
38      return tok_number;
39    }
40
41    if (LastChar == '#') {
42      // Comment until end of line.
43      do
44        LastChar = getchar();
45      while (LastChar != EOF && LastChar != '\n' && LastChar != '\r');
46
47      if (LastChar != EOF)
48        return gettok();
49    }
50
51    // Check for end of file.  Don't eat the EOF.
52    if (LastChar == EOF)
53      return tok_eof;
54
55    // Otherwise, just return the character as its ascii value.
56    int ThisChar = LastChar;
57    LastChar = getchar();
58    return ThisChar;
59  }
60  //===----------------------------------------------------------------------===
```

Code 2: Recognizing new tokens repeat and until

3 Adding New AST Expression for Repeat Until Loop Construct

Third modification done was adding a new AST expression which is **RepeatUntilExprAST** in order to represent parsed repeat-until construct. Newly added AST expression can be seen as in the code listing 3.

```
//=========================================
// RepeatUntilExprAST - This class represents a repeat-until looping
// construct
class RepeatUntilExprAST : public ExprAST {
  std::unique_ptr<ExprAST> Body;
  std::unique_ptr<ExprAST> Condition;

  public:
    RepeatUntilExprAST(std::unique_ptr<ExprAST> Body, std::unique_ptr<
        ExprAST> Condition) :
    Body(std::move(Body)),Condition(std::move(Condition)) {}
};
//=========================================
```

Code 3: Adding RepeatUntilExprAST for representing parsed repeat-until loop construct

4 Modifying ParsePrimary Function

Fourth modification I have done was to extend the **ParsePrimary** function to call the **ParseRepeatExpr** function whenever **tok_repeat** token is encountered. Modified ParsePrimary function can be seen in the code listing 4.

```
//=========================================
/// primary
///   ::= identifierexpr
///   ::= numberexpr
///   ::= parenexpr
///   ::= repeatexpr
static std::unique_ptr<ExprAST> ParsePrimary() {
  switch (CurTok) {
  case tok_identifier:
    return ParseIdentifierExpr();
  case tok_number:
    return ParseNumberExpr();

  // adding Parsing support for repeat-until construct
  case tok_repeat:
    return ParseRepeatExpr();

  case '(':
    return ParseParenExpr();
  default:
    return LogError("unknown token when expecting an expression");
  }
}
//=========================================
```

Code 4: Modifying ParsePrimary function to add parsing support for repeat-until construct

5 Adding ParseRepeatExpr Function

Fifth and final modification I have done was to implement **ParseRepeatExpr** function. This function is used to correctly parse repeat-until looping expressions. Implementation of ParseRepeatExpr function can be seen as given in the code listing 5.

```
//=========================================
/// repeatexpr ::= repeat expression until expression
static std::unique_ptr<ExprAST> ParseRepeatExpr() {
  getNextToken(); // eat the repeat
  auto Body = ParseExpression();

  if (!Body)
    return nullptr;

  if (CurTok != tok_until)
    return LogError("expected 'until' after repeat loop body");

  getNextToken(); // eat the until
  auto Condition = ParseExpression();

  if (!Condition)
    return nullptr;


  fprintf(stderr,"Parsed a repeat-until expr\n");
  return std::make_unique<RepeatUntilExprAST>(std::move(Body),std::move(
      Condition));
}
//=========================================
```

Code 5: ParseRepeatExpr function parses the repeat-until loop construct

Some outputs from Execution

Some outputs from the execution of the Kaleidoscope can be seen in the figure 1.

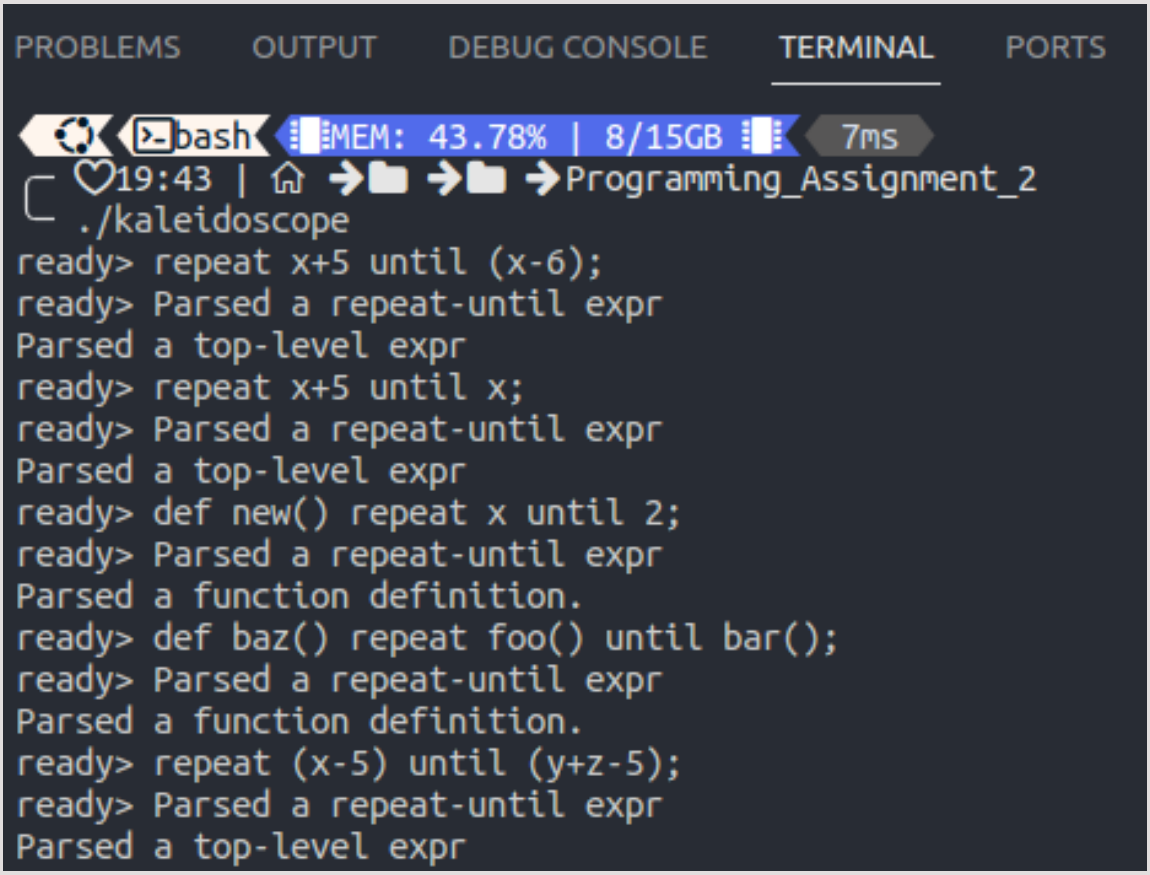


Figure 1: Some Execution Outputs

References

[1] URL: https://llvm.org/docs/tutorial/MyFirstLanguageFrontend/LangImpl01.html.
[2] URL: https://llvm.org/docs/tutorial/MyFirstLanguageFrontend/LangImpl02.html.