Abstract

This project presents the design and implementation of an original programming language named **GökDil**, which uses Turkish characters and syntax. Developed with the Java programming language, GökDil is specifically designed for Turkish-speaking students who are learning programming, allowing them to understand core programming concepts in their native language.

The grammar of GökDil has been carefully designed to provide a clear and intuitive structure. By using Turkish programming terms such as *"tanımla" (define)*, *"ise" (if)*, *"böyleyken" (while)*, and *"fonksiyon" (function)*, the language enhances readability and accessibility for learners.

The language supports fundamental programming constructs including variable declarations, conditional statements, loops, and functions, all using Turkish syntax. It introduces unique features such as using the $ symbol at the end of each statement and @ for comments, which distinguish GökDil from traditional languages.

The implementation consists of three main components:

- A **Lexer**, which breaks the source code into tokens,
- A **Parser**, which converts the tokens into a syntax tree,
- An **Interpreter**, which executes the parsed code.

To demonstrate its practical application, several test files have been created. These examples cover core programming topics such as arithmetic operations, conditionals, and loops. A graphical user interface (GUI) has also been developed to allow users to easily write and test code.

GökDil enhances the learning experience by providing **error messages in Turkish**, making debugging easier and more intuitive for beginners. Syntax errors, type mismatches, and runtime exceptions are all communicated in Turkish.

Overall, this project emphasizes the importance of using native language in programming education. As an open-source language, GökDil welcomes community contributions and is ready to be expanded with features like array support, object-oriented programming, and robust libraries in the future.

Grammar Of GökDil Language As BNF Format

```
<program> ::= <statement_list>

<statement_list> ::= <statement> "$"
                   | <statement> "$" <statement_list>
                   | <comment> <statement_list>
                   | ε

<statement> ::= <variable_declaration>
              | <assignment>
              | <if_statement>
              | <while_statement>
              | <function_declaration>
              | <function_call>
              | <print_statement>
              | <return_statement>



<comment> ::= "@" <any_text>

<variable_declaration> ::= "tanımla" <identifier> "tür"

<type> "başlangıç" <expression>

<type> ::= "tamsayı" | "ondalık" | "yazı" | "mantıksal"

<expression> ::= <logical_expression> | <arithmetic_expression> | <comparison_expression> | <input_expression> | <primary>

<logical_expression> ::= <expression> "ve" <expression> | <expression> "veya" <expression>

<comparison_expression> ::= <expression> <comparison_operator> <expression>

<comparison_operator> ::= ">" | "<" | ">=" | "<=" | "==" | "!="

<arithmetic_expression> ::= <expression> "+" <expression>
                          | <expression> "-" <expression>
                          | <expression> "*" <expression>
                          | <expression> "/" <expression>
                          | <expression> "%" <expression>



<input_expression> ::= "??" <text> "??"
```

```
<primary> ::= <identifier>

              | <number>

               | <string>

              | <boolean>

              | <function_call>

              | "(" <expression> ")"


<boolean> ::= "doğru" | "yanlış"

<if_statement> ::= "ise" "(" <expression> ")" "{" <statement_list> "}"

              | "ise" "(" <expression> ")" "{" <statement_list> "}" "değilse" "{" <statement_list> "}"


<while_statement> ::= "böyleyken" "(" <expression> ")" "{" <statement_list> "}"

<function_declaration> ::= "fonksiyon" <identifier> "(" <parameter_list> ")" "{" <statement_list> "}"

<parameter_list> ::= <identifier> | <identifier> "," <parameter_list> | ε

<function_call> ::= <identifier> "(" <argument_list> ")"

<argument_list> ::= <expression> | <expression> "," <argument_list> | ε

<print_statement> ::= "yazdır" "(" <expression> ")"

<return_statement> ::= "dön" <expression>

<assignment> ::= <identifier> "=" <expression>

<number> ::= <integer> | <float>

<integer> ::= [0-9]+

<float> ::= [0-9]+ "." [0-9]+

<string> ::= """ <any_character>* """

<identifier> ::= <letter> (<letter> | <digit>)*

 <letter> ::= [a-zA-ZğĞüÜşŞıİöÖçÇ]

<digit> ::= [0-9]

<any_character> ::= <letter> | <digit> | <special_character>

 <special_character> ::= [!@#$%^&*()_+\-=\[\]{};':",.<>?/\| ]

<text> ::= <any_character>
```

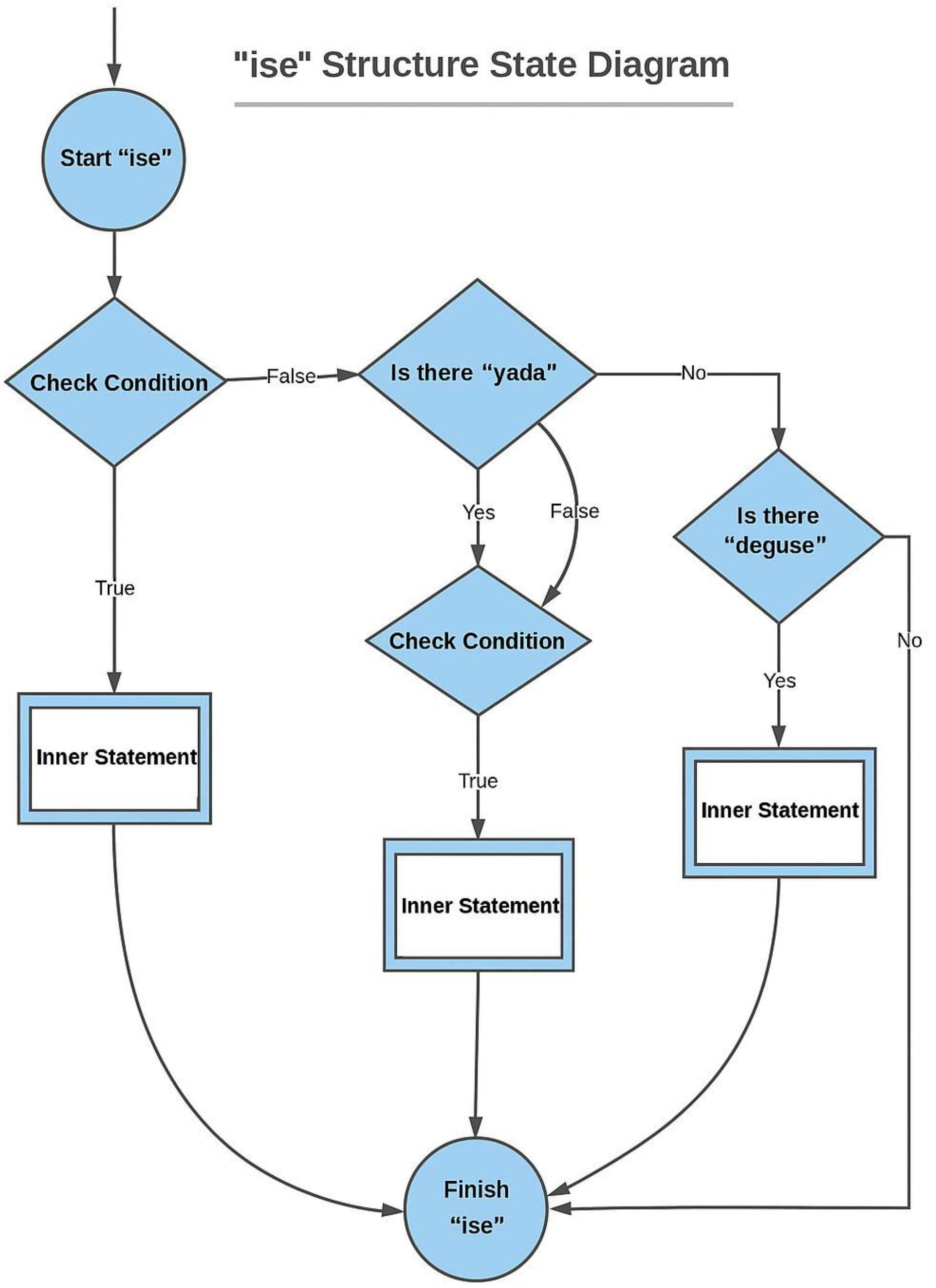
Lookup Table

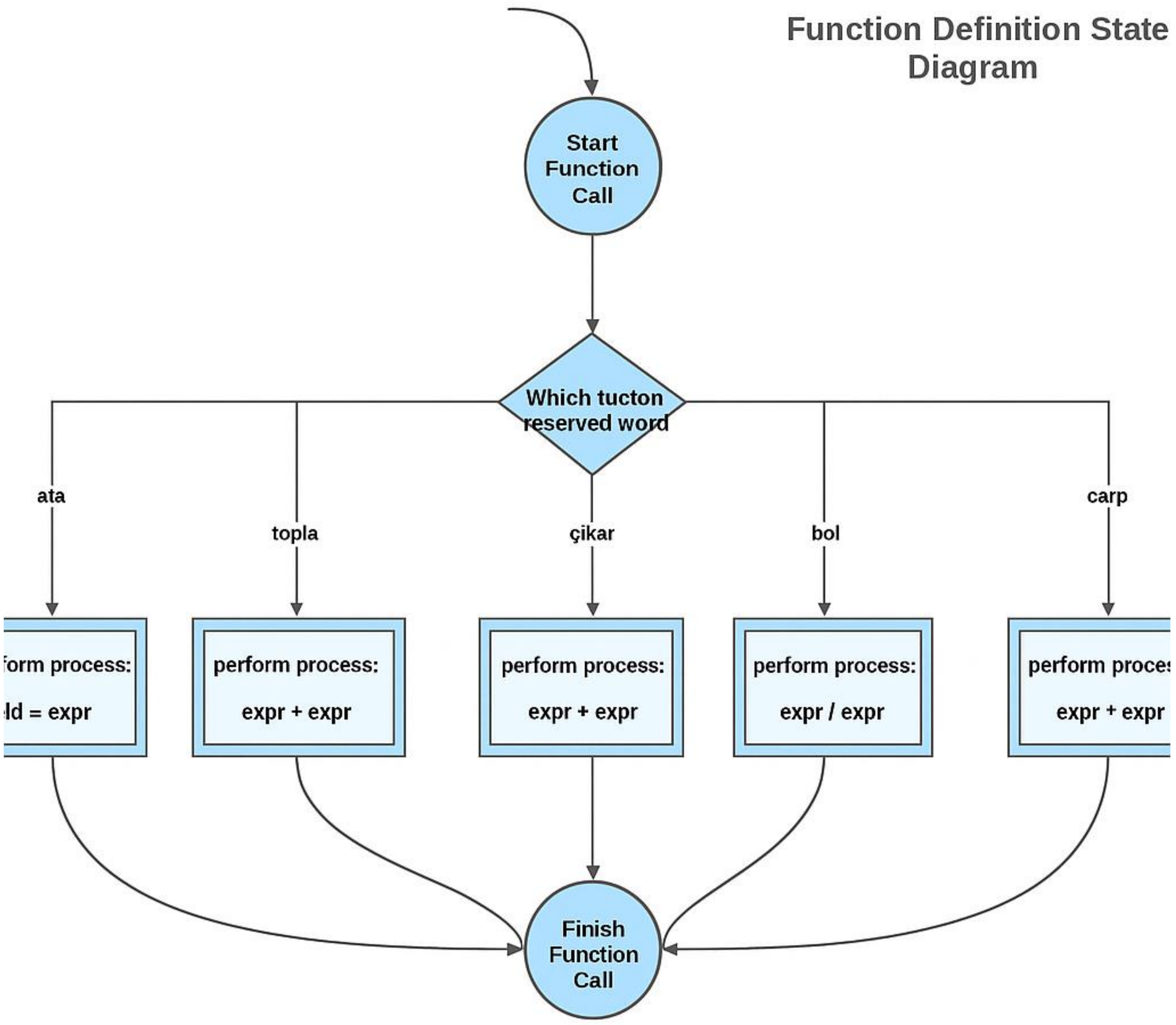
GökDil	**Description**
tanımla	Variable declaration
tür	Type specification
başlangıç	Initial value assignment
ise	If statement
değilse	Else clause
böyleyken	While loop structure
yazdır	Print statement
dön	Return value
ve	Logic AND operator
veya	Logic OR operator
tamsayı	Integer data type
ondalık	Floating-point data type
yazı	String data type
mantıksal	Boolean data type
doğru	Boolean true value
yanlış	Boolean false value
+	Addition operator
-	Subtraction operator
*	Multiplication operator
/	Division operator
%	Modulo operator
=	Assignment operator
==	Equality operator
>	Greater than comparison
<	Less than comparison
>=	Grater or equal comparison
<=	Less or equal comparison
(	Left parenthesis
)	Right parenthesis
{	Left curly brace
}	Right curly brace
@	Comment line
??..??	User input

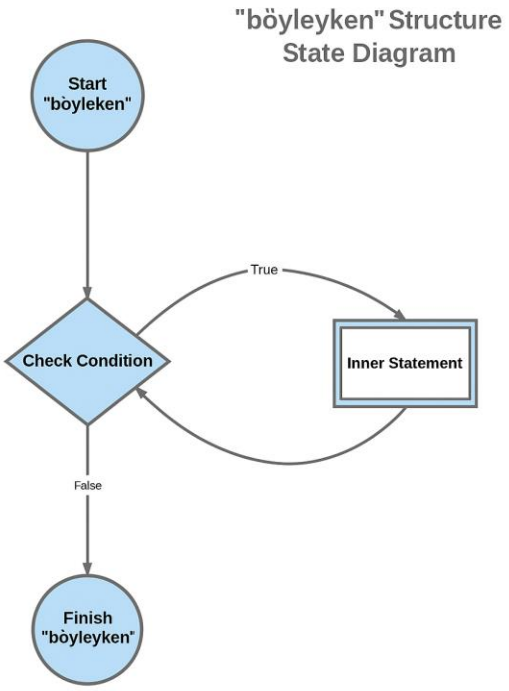
This lookup table defines the reserved keywords, data types, boolean values, operators, delimiters, and special constructs for a Turkish-based programming language.

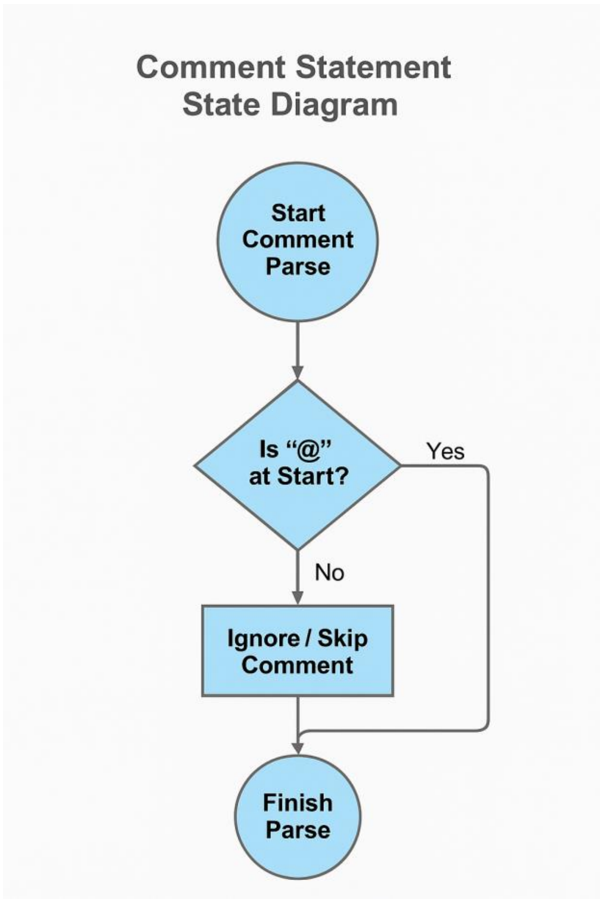
Each entry associates a unique token type with its corresponding keyword or symbol and provides an English description of its function in the language.
The table is designed to be used in the lexer and parser components of a compiler or interpreter, enabling accurate tokenization and syntactic analysis of source code written in this language.

State Diagrams




Function Definition State Diagram
Start Function Call
Which tucton reserved word
ata
topla
çikar
bol
carp
form process:
ld = expr
perform process:
expr + expr
perform process:
expr + expr
perform process:
expr / expr
perform proces
expr + expr
Finish Function Call


"böyleyken" Structure
State Diagram
Start
"bòyleken"
Check Condition
True
Inner Statement
False
Finish
"bòyleyken"


Comment Statement
State Diagram
Start
Comment
Parse
Is "@"
at Start?
Yes
No
Ignore / Skip
Comment
Finish
Parse

GökDil Programming Language - Usage and Setup Guide

This section provides essential information about compiling, running, and understanding the directory structure of the GökDil programming language project. By following these instructions, users can set up and use the project smoothly. Practical tips and troubleshooting steps are also included below.

- 1. Compiling the Project

Before using the GökDil programming language, all source code must be compiled. To do this, enter the project directory in your terminal or command prompt and run the following commands in order:

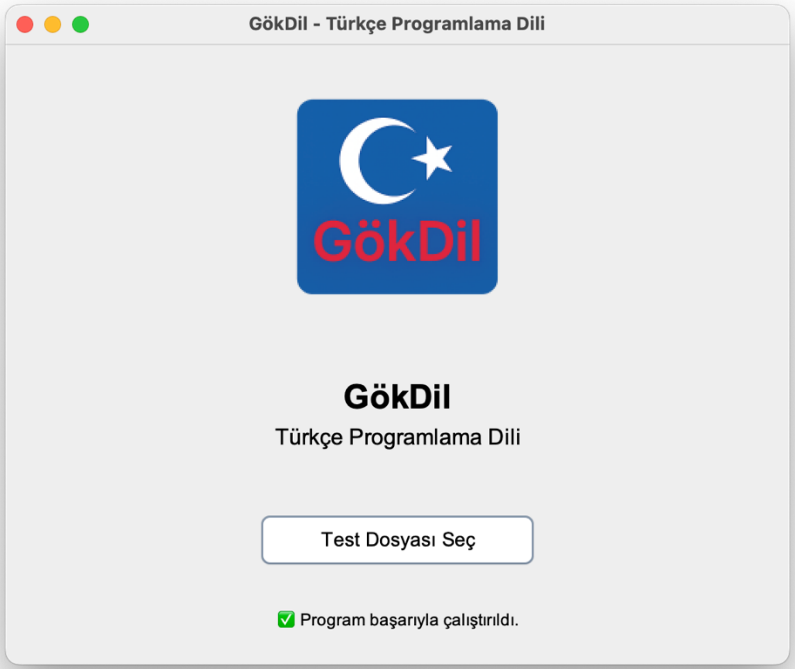
```
cd /Users/yagizgokay/Desktop/GokDilProgLang

    javac -d out src/**/*.java
```

These commands compile the source files and place the compiled classes in the out directory.

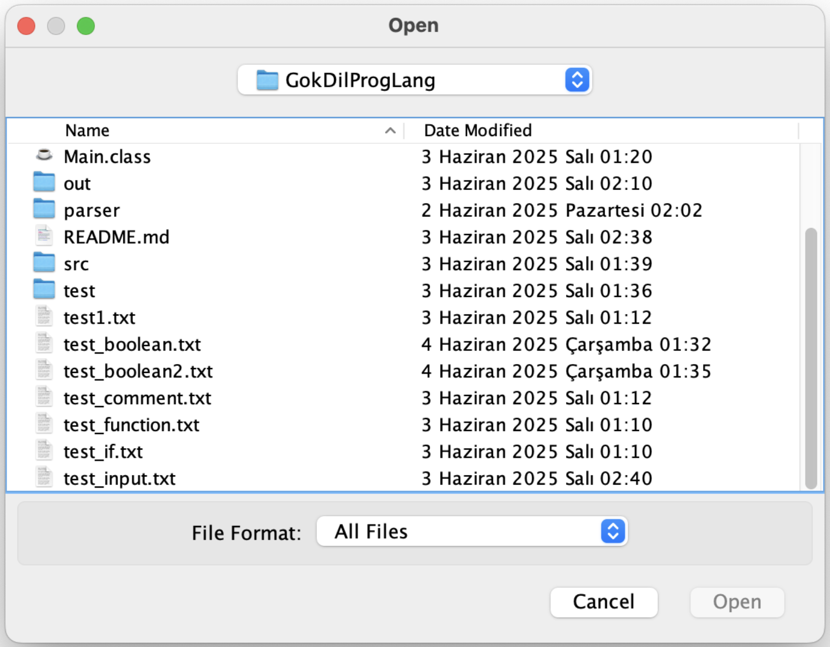
- 2. Running the Program

GökDil features a user-friendly graphical programming interface that allows users to write code, open files, and run their projects. To start the GUI, use:

```
java -cp out main.Main

java -cp out main.Main test_<test_file.txt>

    java -cp out main.Main test_boolean.txt

    java -cp out main.Main test_function.txt
```

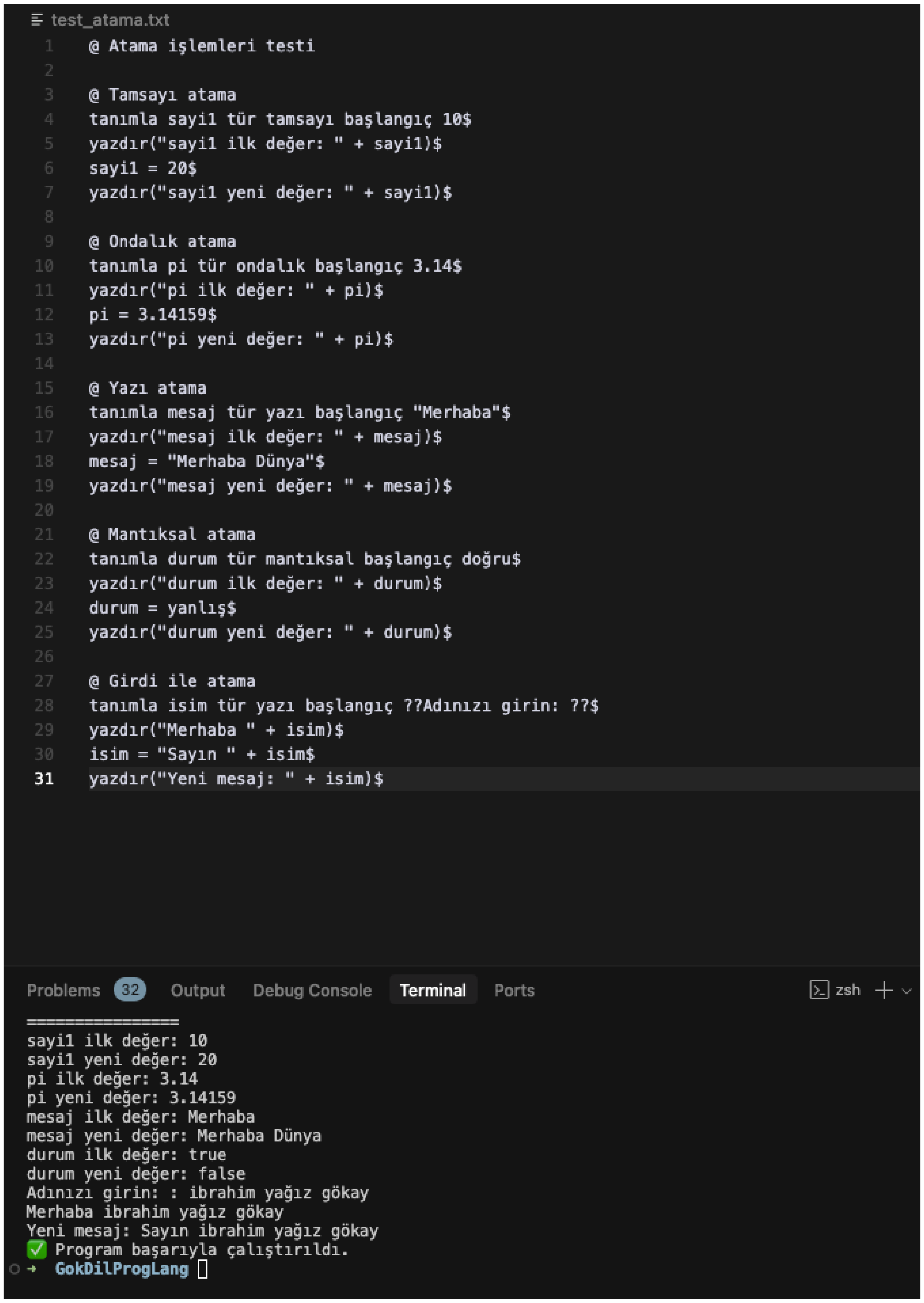


✅ Program başarıyla çalıştırıldı.



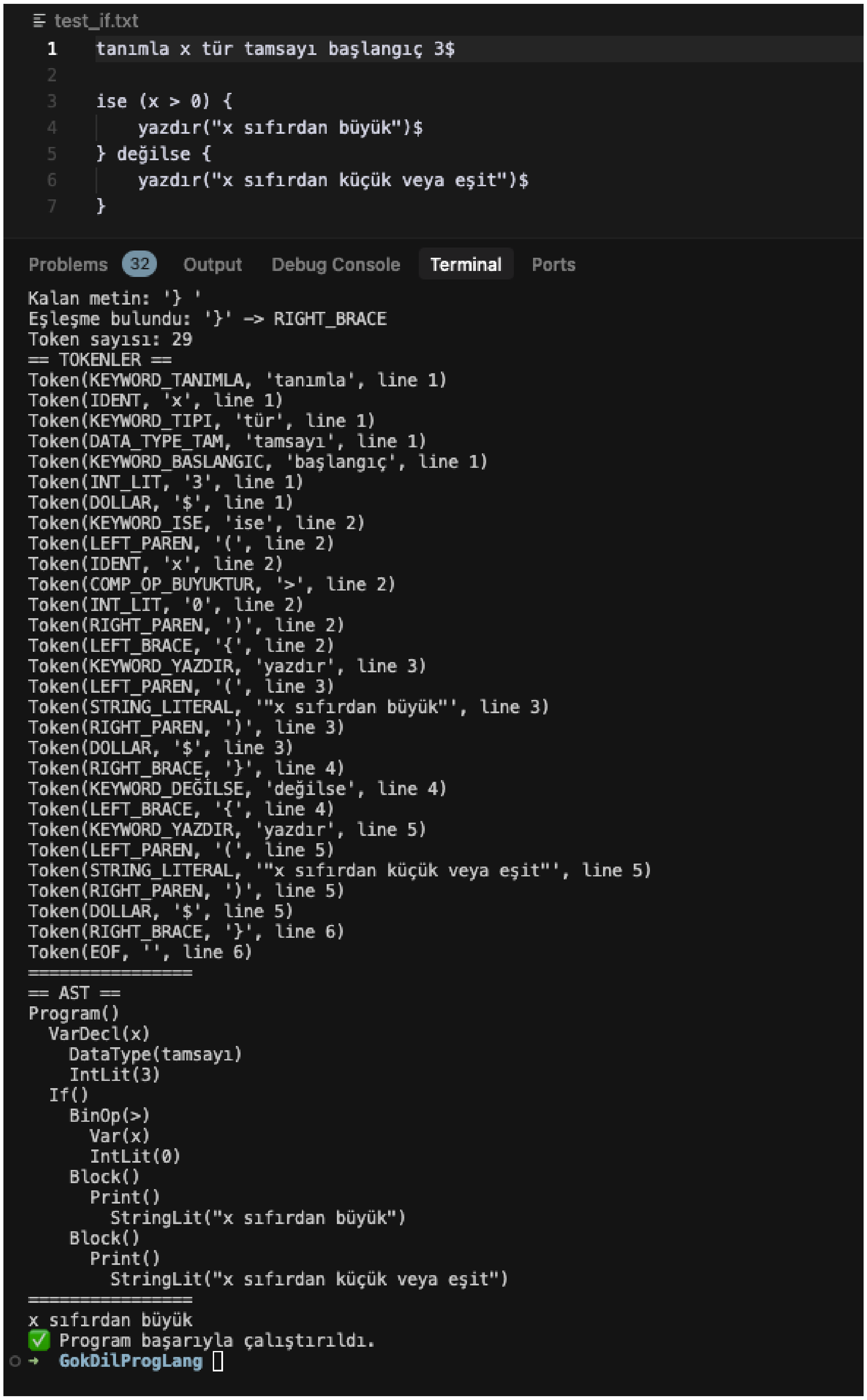
Name	Date Modified
Main.class	3 Haziran 2025 Salı 01:20
out	3 Haziran 2025 Salı 02:10
parser	2 Haziran 2025 Pazartesi 02:02
README.md	3 Haziran 2025 Salı 02:38
src	3 Haziran 2025 Salı 01:39
test	3 Haziran 2025 Salı 01:36
test1.txt	3 Haziran 2025 Salı 01:12
test_boolean.txt	4 Haziran 2025 Çarşamba 01:32
test_boolean2.txt	4 Haziran 2025 Çarşamba 01:35
test_comment.txt	3 Haziran 2025 Salı 01:12
test_function.txt	3 Haziran 2025 Salı 01:10
test_if.txt	3 Haziran 2025 Salı 01:10
test_input.txt	3 Haziran 2025 Salı 02:40

Examples for GökDil Language



```
@ Atama işlemleri testi

@ Tamsayı atama
tanımla sayi1 tür tamsayı başlangıç 10$
yazdır("sayi1 ilk değer: " + sayi1)$
sayi1 = 20$
yazdır("sayi1 yeni değer: " + sayi1)$

@ Ondalık atama
tanımla pi tür ondalık başlangıç 3.14$
yazdır("pi ilk değer: " + pi)$
pi = 3.14159$
yazdır("pi yeni değer: " + pi)$

@ Yazı atama
tanımla mesaj tür yazı başlangıç "Merhaba"$
yazdır("mesaj ilk değer: " + mesaj)$
mesaj = "Merhaba Dünya"$
yazdır("mesaj yeni değer: " + mesaj)$

@ Mantıksal atama
tanımla durum tür mantıksal başlangıç doğru$
yazdır("durum ilk değer: " + durum)$
durum = yanlış$
yazdır("durum yeni değer: " + durum)$

@ Girdi ile atama
tanımla isim tür yazı başlangıç ??Adınızı girin: ??$
yazdır("Merhaba " + isim)$
isim = "Sayın " + isim$
yazdır("Yeni mesaj: " + isim)$
```

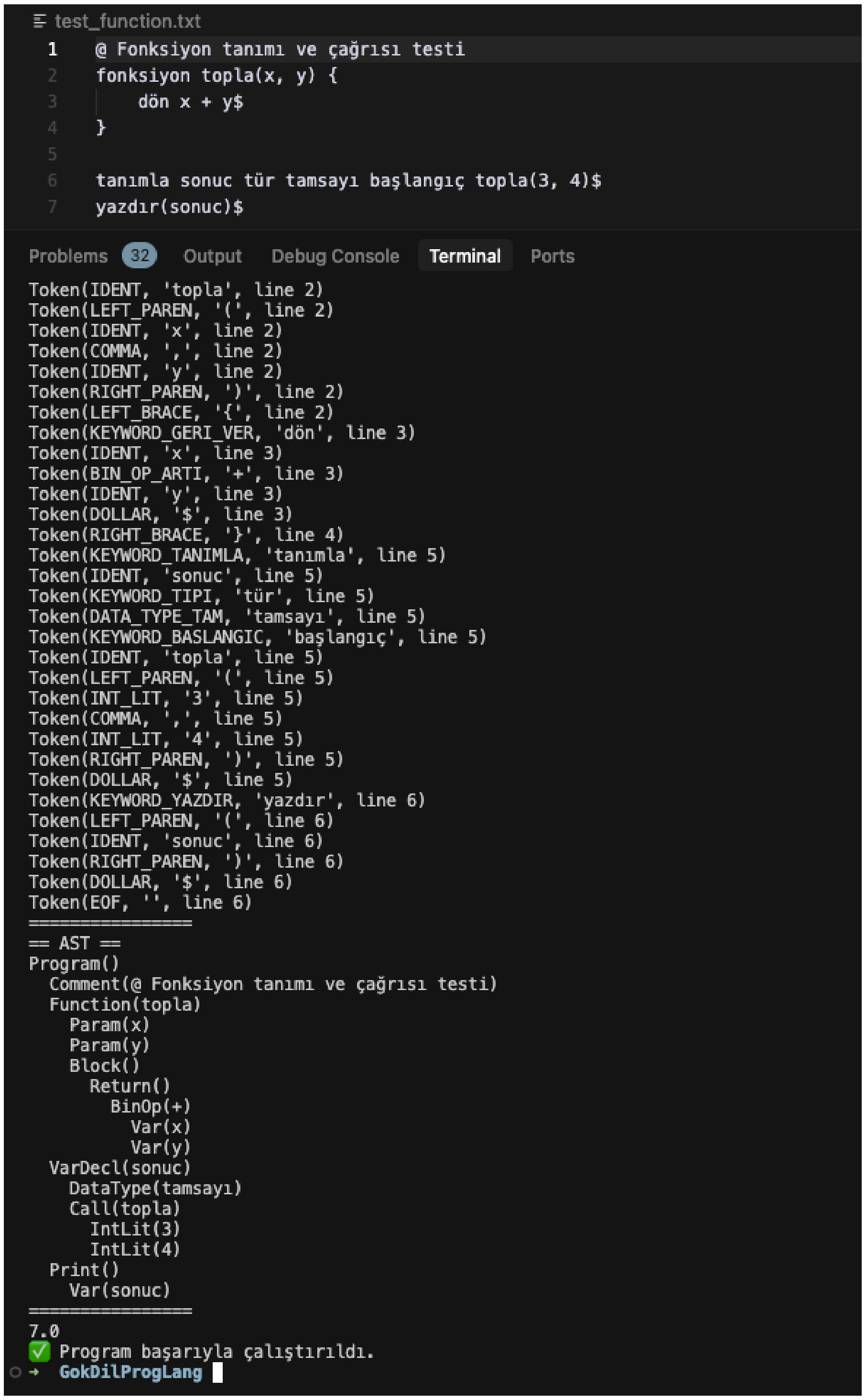
```
================
sayi1 ilk değer: 10
sayi1 yeni değer: 20
pi ilk değer: 3.14
pi yeni değer: 3.14159
mesaj ilk değer: Merhaba
mesaj yeni değer: Merhaba Dünya
durum ilk değer: true
durum yeni değer: false
Adınızı girin: : ibrahim yağız gökay
Merhaba ibrahim yağız gökay
Yeni mesaj: Sayın ibrahim yağız gökay
✅ Program başarıyla çalıştırıldı.
○ → GokDilProgLang
```


test_if.txt

```
tanımla x tür tamsayı başlangıç 3$

ise (x > 0) {
    yazdır("x sıfırdan büyük")$
} değilse {
    yazdır("x sıfırdan küçük veya eşit")$
}
```

Problems 32 Output Debug Console Terminal Ports

```
Kalan metin: '} '
Eşleşme bulundu: '}' -> RIGHT_BRACE
Token sayısı: 29
== TOKENLER ==
Token(KEYWORD_TANIMLA, 'tanımla', line 1)
Token(IDENT, 'x', line 1)
Token(KEYWORD_TIPI, 'tür', line 1)
Token(DATA_TYPE_TAM, 'tamsayı', line 1)
Token(KEYWORD_BASLANGIC, 'başlangıç', line 1)
Token(INT_LIT, '3', line 1)
Token(DOLLAR, '$', line 1)
Token(KEYWORD_ISE, 'ise', line 2)
Token(LEFT_PAREN, '(', line 2)
Token(IDENT, 'x', line 2)
Token(COMP_OP_BUYUKTUR, '>', line 2)
Token(INT_LIT, '0', line 2)
Token(RIGHT_PAREN, ')', line 2)
Token(LEFT_BRACE, '{', line 2)
Token(KEYWORD_YAZDIR, 'yazdır', line 3)
Token(LEFT_PAREN, '(', line 3)
Token(STRING_LITERAL, '"x sıfırdan büyük"', line 3)
Token(RIGHT_PAREN, ')', line 3)
Token(DOLLAR, '$', line 3)
Token(RIGHT_BRACE, '}', line 4)
Token(KEYWORD_DEĞİLSE, 'değilse', line 4)
Token(LEFT_BRACE, '{', line 4)
Token(KEYWORD_YAZDIR, 'yazdır', line 5)
Token(LEFT_PAREN, '(', line 5)
Token(STRING_LITERAL, '"x sıfırdan küçük veya eşit"', line 5)
Token(RIGHT_PAREN, ')', line 5)
Token(DOLLAR, '$', line 5)
Token(RIGHT_BRACE, '}', line 6)
Token(EOF, '', line 6)
================
== AST ==
Program()
  VarDecl(x)
    DataType(tamsayı)
    IntLit(3)
  If()
    BinOp(>)
      Var(x)
      IntLit(0)
    Block()
      Print()
        StringLit("x sıfırdan büyük")
    Block()
      Print()
        StringLit("x sıfırdan küçük veya eşit")
================
x sıfırdan büyük
✅ Program başarıyla çalıştırıldı.
○ → GokDilProgLang
```

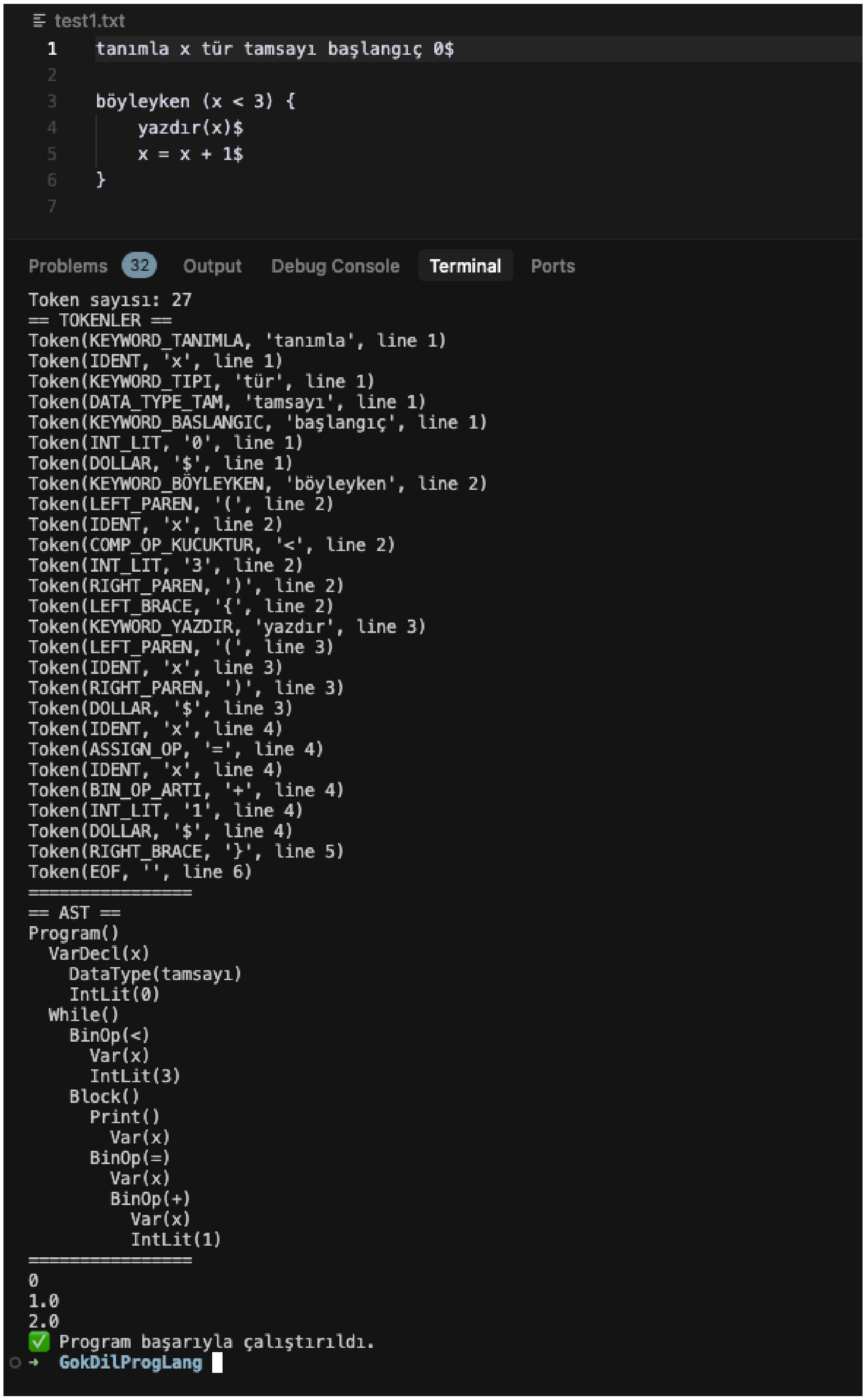


test_function.txt

```
@ Fonksiyon tanımı ve çağrısı testi
fonksiyon topla(x, y) {
    dön x + y$
}

tanımla sonuc tür tamsayı başlangıç topla(3, 4)$
yazdır(sonuc)$
```

Problems 32 Output Debug Console Terminal Ports

```
Token(IDENT, 'topla', line 2)
Token(LEFT_PAREN, '(', line 2)
Token(IDENT, 'x', line 2)
Token(COMMA, ',', line 2)
Token(IDENT, 'y', line 2)
Token(RIGHT_PAREN, ')', line 2)
Token(LEFT_BRACE, '{', line 2)
Token(KEYWORD_GERI_VER, 'dön', line 3)
Token(IDENT, 'x', line 3)
Token(BIN_OP_ARTI, '+', line 3)
Token(IDENT, 'y', line 3)
Token(DOLLAR, '$', line 3)
Token(RIGHT_BRACE, '}', line 4)
Token(KEYWORD_TANIMLA, 'tanımla', line 5)
Token(IDENT, 'sonuc', line 5)
Token(KEYWORD_TIPI, 'tür', line 5)
Token(DATA_TYPE_TAM, 'tamsayı', line 5)
Token(KEYWORD_BASLANGIC, 'başlangıç', line 5)
Token(IDENT, 'topla', line 5)
Token(LEFT_PAREN, '(', line 5)
Token(INT_LIT, '3', line 5)
Token(COMMA, ',', line 5)
Token(INT_LIT, '4', line 5)
Token(RIGHT_PAREN, ')', line 5)
Token(DOLLAR, '$', line 5)
Token(KEYWORD_YAZDIR, 'yazdır', line 6)
Token(LEFT_PAREN, '(', line 6)
Token(IDENT, 'sonuc', line 6)
Token(RIGHT_PAREN, ')', line 6)
Token(DOLLAR, '$', line 6)
Token(EOF, '', line 6)
================
== AST ==
Program()
  Comment(@ Fonksiyon tanımı ve çağrısı testi)
  Function(topla)
    Param(x)
    Param(y)
    Block()
      Return()
        BinOp(+)
          Var(x)
          Var(y)
  VarDecl(sonuc)
    DataType(tamsayı)
    Call(topla)
      IntLit(3)
      IntLit(4)
  Print()
    Var(sonuc)
================
7.0
✅ Program başarıyla çalıştırıldı.
○ → GokDilProgLang
```



test1.txt

```
tanımla x tür tamsayı başlangıç 0$

böyleyken (x < 3) {
    yazdır(x)$
    x = x + 1$
}
```

Problems 32 Output Debug Console Terminal Ports

```
Token sayısı: 27
== TOKENLER ==
Token(KEYWORD_TANIMLA, 'tanımla', line 1)
Token(IDENT, 'x', line 1)
Token(KEYWORD_TIPI, 'tür', line 1)
Token(DATA_TYPE_TAM, 'tamsayı', line 1)
Token(KEYWORD_BASLANGIC, 'başlangıç', line 1)
Token(INT_LIT, '0', line 1)
Token(DOLLAR, '$', line 1)
Token(KEYWORD_BÖYLEYKEN, 'böyleyken', line 2)
Token(LEFT_PAREN, '(', line 2)
Token(IDENT, 'x', line 2)
Token(COMP_OP_KUCUKTUR, '<', line 2)
Token(INT_LIT, '3', line 2)
Token(RIGHT_PAREN, ')', line 2)
Token(LEFT_BRACE, '{', line 2)
Token(KEYWORD_YAZDIR, 'yazdır', line 3)
Token(LEFT_PAREN, '(', line 3)
Token(IDENT, 'x', line 3)
Token(RIGHT_PAREN, ')', line 3)
Token(DOLLAR, '$', line 3)
Token(IDENT, 'x', line 4)
Token(ASSIGN_OP, '=', line 4)
Token(IDENT, 'x', line 4)
Token(BIN_OP_ARTI, '+', line 4)
Token(INT_LIT, '1', line 4)
Token(DOLLAR, '$', line 4)
Token(RIGHT_BRACE, '}', line 5)
Token(EOF, '', line 6)
================
== AST ==
Program()
  VarDecl(x)
    DataType(tamsayı)
    IntLit(0)
  While()
    BinOp(<)
      Var(x)
      IntLit(3)
    Block()
      Print()
        Var(x)
      BinOp(=)
        Var(x)
        BinOp(+)
          Var(x)
          IntLit(1)
================
0
1.0
2.0
✅ Program başarıyla çalıştırıldı.
○ → GokDilProgLang
```