CS 315 Programming Languages

Homework 1 Report

Boolean Expressions in C, Go, Javascript, PHP, Python, and Rust

İlke Doğan 21702215 Section 1

Python Language

Code Explanation:

In Part 1, the OR, AND and NOT operators were compared and tested for their suitability for use i the language. This operation was performed using if-else statements. In the second part, based on the data types found in Python language, it was tested whether they can be used as boolean operators and which ones are appropriate. In chapter 3, the order of priority among AND, OR, (), and NOT operators was achieved by matching pairs between the 4 given operators. In order to investigate the associativity of the AND, OR, (), and NOT operators mentioned above in Chapter 4, the == signs or variables are replaced in the logic with left | right | They were tested for non-associativity. In Chapter 5, 4 functions are defined to test whether there is a certain order when processing operators, and a statement is written in each of them. In this way, it was possible to test whether the function was working and whether there was any associativity. In the or statement written in Chapter 6, if only the statement written in the first function is output, it is tested that Short-circuit evaluation runs from here.

Compiler URL:
https://www.programiz.com/python-programming/online-compiler/

1. *Boolean operators provided*

In python, ***Boolean operators*** that are AND,OR, NOT are tested one by one with different examples. These operators are written as what it is called etc., “true and true”.

Code:

```
# [ 1.1 and oparator test ]

print( "\n")

print( " ----       1. BOOLEAN OPERATORS TEST      ---- ")

print( " ----       BOOLEAN OPERATORS: AND      ---- \n")
```

```
if boolTrue1 and boolTrue2 == True:

    print( "Boolean test for True  - True  and operator: True")

if boolTrue1 and boolFalse1 == False:

    print( "Boolean test for True  - False and operator: False")

print( "\n")

# [ 1.2 or  oparator test ]

print( " ----       BOOLEAN OPERATORS: OR      ---- \n")

if boolTrue1 or boolTrue2 == True:

    print( "Boolean test for True  - True  or operator: True")

if boolTrue1 or boolFalse1 == True:

    print( "Boolean test for True  - False or operator: : True")


print( "\n")

# [ 1.3 not oparator test ]

print( " ----   BOOLEAN OPERATORS: AND/OR NOT  ---- \n")

print("Boolean test for Not True for not operator:", not
boolTrue1 )

print("Boolean test for Not False for not operator:", not
boolFalse1 )

if boolTrue1 and not boolTrue2 == False:

    print( "Boolean test for True  - True  'and not' operator:
False")

if boolTrue1 and not boolFalse1 == True:

    print( "Boolean test for True  - False 'and not' operator:
True")
```

Output:

---- 1. BOOLEAN OPERATORS TEST ----
---- BOOLEAN OPERATORS: AND ----

Boolean test for True - True and operator: True
Boolean test for True - False and operator: False

---- BOOLEAN OPERATORS: OR ----

Boolean test for True - True or operator: True
Boolean test for True - False or operator: : True

---- BOOLEAN OPERATORS: AND & AND NOT ----

Boolean test for Not False for not operator
Boolean test for True - True 'and not' operator: False
Boolean test for True - False 'and not' operator: True

2. Data types for operands of boolean operators

In python, data types such as int, float, boolean are used for operands of boolean operators. For example, 1 for true, 0 for false, float number 1.2 (my example code below) are perceived as True or False.

Code:

```
varFloat = 1.2

# 1 for True

if int( boolTrue1 == True ) == 1 :

    print( "Boolean test for True = 1 is working")

# 0 for True

if bool(0) == False:

    print( "Boolean test for False = 0 is working")

if varFloat:

    print( "Boolean test for Float as boolean operator is
working")

if boolTrue3:

    print( "Boolean test for boolean variable for boolean
operator is working")
```

Output:

---- 2. DATA TYPES FOR BOOLEAN OPERATORS ----
Boolean test for True = 1 is working
Boolean test for False = 0 is working
Boolean test for Float as boolean operator is working
Boolean test for boolean variable for boolean operator is working

3. Operator precedence rules

In python, Precedence of operators between are ranked as 'Parentheses' > AND >OR as can be seen in the below.

Code:

```
print( "Boolean test for True and True  or False condition: ",
boolTrue1 and boolTrue2 or boolFalse1)

print( "Boolean test for True or True  and False condition: ",
boolTrue1 or boolTrue2  and boolFalse1) # and has precedence compared
to or

print( " -- BOOLEAN OPERATORS: 'AND' VERSUS 'Parentheses' -- \n")

print( "Boolean test for False and False  and True condition : ",
boolFalse1 and boolFalse2  or boolTrue1)

print( "Boolean test for False and (False and True) condition: ",
boolFalse1 and (boolFalse2  or boolTrue1)) # '()' has precedence
compared to and
```

Output:

```
----    BOOLEAN OPERATORS: 'AND' VERSUS 'OR'   ----

Boolean test for True and True  or False condition:  True
Boolean test for True or True  and False condition:  True

 -- BOOLEAN OPERATORS: 'AND' VERSUS 'Parentheses' --

Boolean test for False and False  and True condition :  True
Boolean test for False and (False and True) condition:  False
```

4. Operator associativity rules

Precedence Rank	***Associativity***
1: ()	non associative
2: NOT	right associativity
3: AND	left associativity
4: OR	left associativity

In python, Left associative operators are And, Or. Right associative operator is Not. Non- associative operators are 'Parentheses' that are tested in the given code.

Code:

```
name = "ilke"

favHobby = "space"

if( (favHobby == ' space' and favHobby == 'reading') or name ==
"ilke"):

    print("Left associativity is working for and") # if it was
right associativity, this line would work

else:

    print("Right associativity is working for and")

if( (favHobby == ' space' or favHobby == 'reading') and name ==
"ilke"):

    print("Right associativity is working") # and and or has left
associativity

else:

    print("Left associativity is working  for or") # if it was
right associativity, this line would work

# Left associativity test for not opartor

if( not boolTrue1 or boolTrue2):

    print(" 'Not' has Right associativity")

else:

    print(" 'Not' has Left associativity")
```

Output:

Left associativity is working for and

Left associativity is working

'Not' has Right associativity

5. Operand evaluation order

In python, based on the code given below, operand evaluation is realized LEFT -> RIGHT. It is understood based on the print statements are written inside the functions etc,. print("Inside Test Function 2") .

Code:

```
result = True
def testFunc1():
    result = True
    print("Inside Test Function 1")
    return result
def testFunc2():
    result = False
    print("Inside Test Function 2")
    return result
def testFunc3():
    result = True
    print("Inside Test Function 3")
    return result
def testFunc4():
    result = False
    print("Inside Test Function 4")
    return result

#  Prove operands work LEFT -> RIGHT
boolTest = testFunc1() and testFunc2() and testFunc3() and testFunc4()
print("Function-1 and Function-2 and Function-3 and Function-4:
",boolTest)
boolTest = testFunc1() or testFunc2() or testFunc3() or testFunc4()
print("Function-1 or Function-2 or Function-3 or Function-4: ",
boolTest)
```

Output:

```
Inside Test Function 1
Inside Test Function 2
Function-1 and Function-2 and Function-3 and Function-4:  False
Inside Test Function 1
Function-1 or Function-2 or Function-3 or Function-4:  True
```

6. Short-circuit evaluation

In python, while checking the statements, it is started from left to right. If the left side fulfills the condition, there is no need to check the right side that can be seen in the example given below. As a result there is short- circuit evaluation.

Code:

```
    if(testFunc1() or testFunc2() ):
        print("Short-circuit evaluation is working \n")
```

Output:

```
Inside Test Function 1
Short-circuit evaluation is working
```

Javascript Language

Code Explanation:

Firstly, the website page is designed for good-looking with using html terms. The Javascript part is started between <script> brackets. In Part 1, the OR, AND and NOT operators were compared and tested for their suitability for use in the language. This operation was performed using if-else statements. In the second part, based on the data types found in JavaScript language, it was tested whether they can be used as boolean operators and which ones are appropriate. In chapter 3, the order of priority among AND, OR, (), and NOT operators was achieved by matching pairs between the 4 given operators. In order to investigate the associativity of the AND, OR, (), and NOT operators mentioned above in Chapter 4, the == signs or variables are replaced in the logic with left | right | They were tested for non-associativity. In Chapter 5, 4 functions are defined to test whether there is a certain order when processing operators, and a statement is written in each of them. In this way, it was possible to test whether the function was working and whether there was any associativity. In the or statement written in Chapter 6, if only the statement written in the first function is output, it is tested that Short-circuit evaluation runs from here.

Compiler URL: https://js.do/

1. *Boolean operators provided*

In JavaScript, ***Boolean operators*** that are AND,OR, NOT are tested one by one with different examples. These operators are written as what it is called etc., “true and true”.

Code:

```
 // [ 1.1 and oparator test ]

        subheading += " *********   1. BOOLEAN OPERATORS TEST
*********<br><br>BOOLEAN OPERATORS: AND<br>"

        boolvar = boolTrue1 && boolTrue2;
        subheading += "Boolean test for True  - True and operator: " +
boolvar +  "<br>"
        boolvar = boolTrue1 && boolFalse1;
        subheading += "Boolean test for True  - False and operator: " +
boolvar +  "<br>"
```

```
 // [ 1.2 or  oparator test ]
        subheading += " BOOLEAN OPERATORS: OR<br>"

        boolvar = boolTrue1 || boolTrue2;
        subheading += "Boolean test for True  - True or operator: " +
boolvar +  "<br>"
        boolvar = boolTrue1 || boolFalse1;
        subheading += "Boolean test for True  - False or operator: " +
boolvar +  "<br>"

 // [ 1.3 not oparator test ]
        subheading += "BOOLEAN OPERATORS: AND/OR NOT<br>"

        boolvar = !(boolTrue1 && boolTrue2);
        subheading += "Boolean test for True  - True and not operator:
" + boolvar +  "<br>"
        boolvar = !(boolTrue1 && boolFalse1);
        subheading += "Boolean test for True  - False and not operator:
" + boolvar +  "<br><br>"
```

Output:

BOOLEAN OPERATORS: AND
Boolean test for True - True and operator: true
Boolean test for True - False and operator: false
BOOLEAN OPERATORS: OR
Boolean test for True - True or operator: true
Boolean test for True - False or operator: true
BOOLEAN OPERATORS: AND/OR NOT
Boolean test for True - True and not operator: false
Boolean test for True - False and not operator: true

2. *Data types for operands of boolean operators*

In JavaScript, data types such as int, float, bool are used for operands of boolean operators like Python Language. For example, 1 for true, 0 for false, float number 1.2 (my example code below) are perceived as True or False.

Code:

```
intBool = 1
    doubleBool = 0
    floatBool = 1.2
    // 1 for True
    if (intBool){
        subheading +="Boolean test for True = 1 is working<br>"
    }
    // 0 for True
    if (intBool){
        subheading +="Boolean test for False = 0 is working<br>"
    }
    // Float = 1.2 for True
    if (floatBool){
```

```
        subheading +="Boolean test for Float as boolean operator is
working<br>"
    }
    if (boolTrue2){
        subheading +="Boolean test for boolean variable for boolean
operator is working<br><br>"
    }
```

Output:

Boolean test for True = 1 is working
Boolean test for False = 0 is working
Boolean test for Float as boolean operator is working
Boolean test for boolean variable for boolean operator is working

3. Operator precedence rules

In JavaScript, Precedence of operators between are ranked as 'Parentheses' > AND >OR as can be seen in the below. It is working as similar as Python, C, and Go languages.

Code:

```
 subheading += " *********   3. BOOLEAN PRECEDENCE TEST
*********<br><br>BOOLEAN OPERATORS: 'AND' VERSUS 'OR': AND<br>"

        boolvar = boolTrue1 && boolTrue2 || boolFalse1;
        subheading += "Boolean test for True and True  or False
condition: " + boolvar +  "<br>"
        boolvar = boolTrue1 || boolTrue2 && boolFalse1;
        subheading += "Boolean test for True or True  and False
condition: " + boolvar +  "<br>" //# and has precedence compared to or

        subheading += " <br> BOOLEAN OPERATORS: 'AND' VERSUS
'Parantheses<br>"
        boolvar = boolFalse1 && boolFalse2 || boolTrue1;
        subheading += "Boolean test for True and True  or False
condition: " + boolvar +  "<br>"
        boolvar = boolFalse1 && (boolFalse2 || boolTrue1);
        subheading += "Boolean test for True or True  and False
condition: " + boolvar +  "<br><br>" //# '()' has precedence compared
to and
```

Output:

BOOLEAN OPERATORS: 'AND' VERSUS 'OR': AND
Boolean test for True and True or False condition: true
Boolean test for True or True and False condition: true

BOOLEAN OPERATORS: 'AND' VERSUS 'Parentheses
Boolean test for True and True or False condition: true
Boolean test for True or True and False condition: false

4. Operator associativity rules

In JavaScript, Left associative operators are And, Or. Right associative operator is Not. Non- associative operators are 'Parentheses' that are tested in the given code.

Code:

```
  name = "ilke";
    favHobby = "space";

    if ((favHobby == ' space' && favHobby == 'reading') || name ==
"ilke"){
        subheading +="Left associativity is working<br><br>" //if it
was right associativity, this line would work
    }
    else{
        subheading +="Right associativity is working for and<br><br>"
    }
    if ((favHobby == ' space' || favHobby == 'reading')&& name ==
"ilke"){
        subheading +="Right associativity is working for or<br><br>"
    }
    else{
        subheading +="Left associativity is working<br><br>" // if it
was right associativity, this line would work
    }
    if ( !boolTrue2 || boolTrue3){
        subheading +="'Not' has Right associativity<br><br>"
    }
    else{
        subheading +="'Not' has Left associativity<br><br>"
    }
```

Output:

Left associativity is working for and
Left associativity is working
'Not' has Right associativity

5. Operand evaluation order

In JavaScript, based on the code given below, operand evaluation is realized LEFT -> RIGHT like Python and Go. It is understood based on the print statements are written inside the functions etc,. print("Inside Test Function 2") .

Code:

```
function myFunction1() {
        boolvar = true;
        subheading += "Inside Test Function 1 <br>";
        return boolvar;
    }
```

```
    function myFunction2() {
        boolvar = false;
        subheading += "Inside Test Function 2 <br>";
        return boolvar;
    }
    function myFunction3() {
        boolvar = true;
        subheading += "Inside Test Function 3 <br>";
        return boolvar;
    }
    function myFunction4() {
        boolvar = false;
        subheading += "Inside Test Function 4 <br>";
        return boolvar;
    }
    // #  Prove operands work LEFT -> RIGHT
    boolvar = myFunction1() && myFunction2() && myFunction3() &&
myFunction4();
    subheading += "Function-1 and Function-2 and Function-3 and
Function-4: " + boolvar + "<br>"

    boolvar = myFunction1() || myFunction2() || myFunction3() ||
myFunction4();
    subheading += "Function-1 or Function-2 or Function-3 or
Function-4: " + boolvar + "<br><br>"
```

Output:

Inside Test Function 1
Inside Test Function 2
Function-1 and Function-2 and Function-3 and Function-4: False
Inside Test Function 1
Function-1 or Function-2 or Function-3 or Function-4: True

6. Short-circuit evaluation

In JavaScript, while checking the statements, it is started from left to right. If the left side fulfills the condition, there is no need to check the right side that can be seen in the example given below. As a result there is short- circuit evaluation like Python, C, and Go.

Code:

```
if ( myFunction1() || myFunction2()){
        subheading += "Short-circuit evaluation is working <br>"
    }
```

Output:

Inside Test Function 1
Short-circuit evaluation is working

C Language

Code Explanation:

Firstly in Part 1, the OR, AND and NOT operators were compared and tested for their suitability for use in the language. This operation was performed using if-else statements. In the second part, based on the data types found in C language, it was tested whether they can be used as boolean operators and which ones are appropriate. In chapter 3, the order of priority among AND, OR, (), and NOT operators was achieved by matching pairs between the 4 given operators. In order to investigate the associativity of the AND, OR, (), and NOT operators mentioned above in Chapter 4, the == signs or variables are replaced in the logic with left | right | They were tested for non-associativity. In Chapter 5, 4 functions are defined to test whether there is a certain order when processing operators, and a statement is written in each of them. In this way, it was possible to test whether the function was working and whether there was any associativity. In the or statement written in Chapter 6, if only the statement written in the first function is output, it is tested that Short-circuit evaluation runs from here.

Compiler URL:
https://www.programiz.com/c-programming/online-compiler/

1. *Boolean operators provided*

In C language, ***Boolean operators*** that are AND,OR, NOT are tested one by one with different examples. These operators are written as what it is called etc., “true and true”. In this language, boolean operators are the same as Python, JavaScript, and Go.

Code:

```
  // [ 1.1 and oparator test ]
printf( "\n");
printf( " ----       1. BOOLEAN OPERATORS TEST  ---- \n");
printf( " ----       BOOLEAN OPERATORS: AND      ---- \n");
if (boolTrue1 && boolTrue2 == true){
    printf( "Boolean test for True  - True  and operator: True\n");
}
if (boolTrue1 && boolFalse1 == false){
    printf( "Boolean test for True  - False and operator: False\n");
}
printf( "\n");
```

```
// [ 1.2 or  oparator test ]
printf( " ----       BOOLEAN OPERATORS: OR      ---- \n");
if (boolTrue1 || boolTrue2 == true){
    printf( "Boolean test for True  - True  or operator: True\n");
}
if (boolTrue1 || boolFalse1 == true){
   printf( "Boolean test for True  - False or operator: : True\n");
}

printf( "\n");

// [ 1.3 not oparator test ]
printf( " ----   BOOLEAN OPERATORS: AND/OR NOT  ---- \n");
printf("Boolean test for Not True for not operator:", !(boolTrue1) );
printf( "\n");
printf("Boolean test for Not False for not operator:", !(boolFalse1) );
printf( "\n");
if (!(boolTrue1 && boolTrue2) == false){
    printf( "Boolean test for True  - True  'and not' operator:
False\n");
}
if (!(boolTrue1 && boolFalse1)== true){
        printf( "Boolean test for True  - False 'and not' operator:
True\n");
}
```

Output:

```
----      1. BOOLEAN OPERATORS TEST  ----
 ----      BOOLEAN OPERATORS: AND      ----
Boolean test for True  - True  and operator: True
Boolean test for True  - False and operator: False
 ----      BOOLEAN OPERATORS: OR      ----
Boolean test for True  - True  or operator: True
Boolean test for True  - False or operator: : True
 ----   BOOLEAN OPERATORS: AND/OR NOT  ----
Boolean test for Not True for not operator: False
Boolean test for Not False for not operator: True
Boolean test for True  - True  'and not' operator: False
Boolean test for True  - False 'and not' operator: True
```

2. Data types for operands of boolean operators

In C Language,data types such as int, float, bool are used for operands of boolean operators like Python and Javascript languages. For example, 1 for true, 0 for false, float number 1.2 (my example code below) are perceived for.

Code:

```
float varFloat = 1.2;
int varInt = 1;
int varInt2 = 0;
```

```
// 1 for True
if (varInt){
    printf( "Boolean test for True = 1 is working\n");
}
// 0 for True
if (varInt2){
    printf( "Boolean test for False = 0 is working\n");
}
if (varFloat){
    printf( "Boolean test for Float for True\n");
}
else{
    printf( "Boolean test for Float for False\n");
}
if (boolTrue3){
   printf( "Boolean test for boolean variable for boolean operator is
working\n");
}
```

Output:

Boolean test for True = 1 is working
Boolean test for False = 0 is working
Boolean test for Float for True
Boolean test for boolean variable for boolean operator is working

3. Operator precedence rules

In C Language, Precedence of operators between are ranked as 'Parentheses' > AND >OR as can be seen in the below.

Code:

```
bool temp1 =  boolTrue1 && boolTrue2  || boolFalse1;
    if(temp1){
        printf("Boolean test for True and True  or False condition:
True");
    }
    printf("\n");
    bool temp2 =  boolTrue1  || boolTrue2 && boolFalse1;
    if(temp2 != false){
        printf("Boolean test for True or True  and False condition:
True");
    }
    printf("\n");

    printf(" BOOLEAN OPERATORS: 'AND' VERSUS 'Parantheses'   \n");
    temp1 =  boolFalse1 && boolFalse2  || boolTrue1;
    if(temp1){
        printf("Boolean test for False and False  and True condition:
True");
    }
    printf("\n");
    temp1 =  boolFalse1  &&  (boolFalse2 ||  boolTrue1);
    if(!temp1){
```

```
        printf("Boolean test for False and (False and True) condition:
False");
    }
```

Output:

BOOLEAN OPERATORS: 'AND' VERSUS 'OR'
Boolean test for True and True or False condition: True
Boolean test for True or True and False condition: True
BOOLEAN OPERATORS: 'AND' VERSUS 'Parentheses'
Boolean test for False and False and True condition: True
Boolean test for False and (False and True) condition: False

4. Operator associativity rules

In C Language, Left associative operators are And, Or. Right associative operator is Not. Non- associative operators are 'Parentheses' that are tested in the given code. Additionally, in this language there is no string and because of that I change my example that I used in the programs can be found above(Python,Js,C) to test operator associativity.

Code:

```
int varInt1  = 22;
    int varInt2  = 24;
    int varInt3  = 24;

    if ((varInt1 == varInt2) && varInt3 == 24) {
        printf("Right associativity is working for and\n"); // if it
was right associativity, this line would work
    }
    else{
        printf("Left associativity is working for and\n");
    }

    if ((varInt1 == varInt2) || varInt3 == 24){
        printf("Left associativity is working\n"); // and and or has
left associativity
    }
    else{
        printf("Right associativity is working  for or\n"); // if it
was right associativity, this line would work
    }
    // Left associativity test for not opartor
```

```
    if (!boolTrue1 || boolTrue2) {
        printf(" 'Not' has Right associativity\n");
    }
    else{
        printf(" 'Not' has Left associativity\n");
    }
```

Output:

Left associativity is working for and
Left associativity is working
'Not' has Right associativity

5. Operand evaluation order

In C Language, based on the code given below, there is no order for the operand evaluation. There is no Left->Right or Right->Left concept. To clarify that, both for AND and OR operators the same statement is tested with one '()' difference. As a result, there were no changes, so that there is no order for the operand evaluation unlike Python, Javascript, and Go.

Code:

```
printf("\n  5. OPERAND EVALUATION ORDER \n");
        bool boolvar2 = true;
        printf("Same statement with no '()'\n");
        boolvar2 = myFunction1() && myFunction2() && myFunction3() &&
myFunction4();
        printf("Same statement no '()'\n");
        boolvar2 = (myFunction1() && myFunction2()) && myFunction3() &&
myFunction4(); // same results are taken for and
        printf("Same statement with no '()'\n");
        boolvar2 = myFunction1() || myFunction2() || myFunction3() ||
myFunction4();
        printf("Same statement no '()'\n");
        boolvar2 = (myFunction1() || myFunction2()) || myFunction3() ||
myFunction4(); // same results are taken for or
        return 0;
```

Output:

Same statement with no '()'

Inside Test Function 1
Inside Test Function 2
Same statement no '()'

Inside Test Function 1
Inside Test Function 2
Same statement with no '()'
Inside Test Function 1
Same statement no '()'
Inside Test Function 1

6. Short-circuit evaluation

In C Language, while checking the statements, it is started from left to right. If the left side fulfills the condition, there is no need to check the right side that can be seen in the example given below. As a result there is short- circuit evaluation like Python, Javascript, and Go.

Code:

```
if (myFunction1() || myFunction2()) {
            printf(" Short-circuit evaluation is working ");
        }
```

Output:

Inside Test Function 1
Short-circuit evaluation is working

Go Language

Code Explanation:

Firstly in Part 1, the OR, AND and NOT operators were compared and tested for their suitability for use in the language. This operation was performed using if-else statements. In the second part, based on the data types found in GO language, it was tested whether they can be used as boolean operators and which ones are appropriate. In chapter 3, the order of priority among AND, OR, (), and NOT operators was achieved by matching pairs between the 4 given operators. In order to investigate the associativity of the AND, OR, (), and NOT operators mentioned above in Chapter 4, the == signs or variables are replaced in the logic with left | right | They were tested for non-associativity. In Chapter 5, 4 functions are defined to test whether there is a certain order when processing operators, and a statement is written in each of them. In this way, it was possible to test whether the function was working and whether there was any associativity. In the or statement

written in Chapter 6, if only the statement written in the first function is output, it is tested that Short-circuit evaluation runs from here.

Compiler URL:
https://www.onlinegdb.com/online_go_compiler

1. Boolean operators provided

In Go Language, ***Boolean operators*** that are AND,OR, NOT are tested one by one with different examples. These operators are written as what it is called etc., “true and true”. Additionally, I got errors for not used variables that are declared in the scope, so in this language, there is no permission for keeping not used variables.

Code:

```
fmt.Println(" 1 . BOOLEAN OPERATORS TESTBOOLEAN OPERATORS: AND")

    fmt.Println(" BOOLEAN OPERATORS: AND")
    if boolTrue1 && boolTrue2 == true {
     fmt.Println("Boolean test for True  - True  and operator: True")
    }
    if boolTrue1 && boolFalse3 == false {
     fmt.Println("Boolean test for True  - False and operator:
False\n")
    }
    // [ 1.2 or  operator test ]
    /*
        BOOLEAN OPERATORS: OR
    */
    fmt.Println(" BOOLEAN OPERATORS: OR ")
    if boolTrue1 || boolTrue3 == true {
     fmt.Println("Boolean test for True  - True  or operator: True")
    }

    if boolTrue1 || boolFalse2 == true {
     fmt.Println("Boolean test for True  - False or operator: True\n")
    }
    // [ 1.3 not operator test ]
    /*
        BOOLEAN OPERATORS: AND/OR NOT
    */
    fmt.Println(" BOOLEAN OPERATORS: AND/OR NOT ")
    if !(boolTrue1 && boolTrue3) == false {
     fmt.Println("Boolean test for True  - True  and not operator:
False")
    }

    if !(boolTrue1 && boolFalse1) == true {
     fmt.Println("Boolean test for True  - False and not operator:
True")
    }
    fmt.Println("\n")
```

Output:

BOOLEAN OPERATORS: AND

Boolean test for True - True and operator: true
Boolean test for True - False and operator: false
BOOLEAN OPERATORS: OR
Boolean test for True - True or operator: true
Boolean test for True - False or operator: true
BOOLEAN OPERATORS: AND/OR NOT
Boolean test for True - True and not operator: false
Boolean test for True - False and not operator: true

2. Data types for operands of boolean operators

In Go Language, data types such as int, float,double are not used for operands of boolean operators. Additionally, in this language there are no floating numbers such as float, double. Therefore, it cannot be used as a boolean operator as it is in Python and Javascript.

Code:

```
if boolTrue3{
        Println( "Boolean test for boolean variable for boolean
operator is working")
    }
```

Output:

Boolean test for boolean variable for boolean operator is working

3. Operator precedence rules

In Go Language, Precedence of operators between are ranked as 'Parentheses' > AND >OR as can be seen in the below as same as Python, JavaScript, and C languages.

Code:

```
fmt.Println(" BOOLEAN OPERATORS: 'AND' VERSUS 'OR'  ")
    fmt.Println("Boolean test for True and True  or False condition:",
boolTrue1 && boolTrue2  || boolFalse1)
    fmt.Println("Boolean test for True or True  and False condition: ",
boolTrue1  || boolTrue2 && boolFalse1)
    //and has precedence compared to or
    fmt.Println("\n")

    fmt.Println(" BOOLEAN OPERATORS: 'AND' VERSUS 'Parantheses'   ")
    fmt.Println("Boolean test for False and False  and True condition
:", boolFalse1 && boolFalse2  || boolTrue1)
```

```
    fmt.Println("Boolean test for False and (False and True) condition:
", boolFalse1  &&  (boolFalse2 ||  boolTrue1))
    // '()' has precedence compared to and
    fmt.Println("\n")
```

Output:

BOOLEAN OPERATORS: 'AND' VERSUS 'OR': AND
Boolean test for True and True or False condition: true
Boolean test for True or True and False condition: true

BOOLEAN OPERATORS: 'AND' VERSUS 'Parentheses
Boolean test for True and True or False condition: true
Boolean test for True or True and False condition: false

4. Operator associativity rules

In Go Language, Left associative operators are And, Or. Right associative operator is Not. Non- associative operators are 'Parentheses' that are tested in the given code.

One the tricky point for this language is if-else statements compared the other language as while writing if-else statements together else statements have to merge with if's end bracket. Additionally, in this language there is no string and because of that I change my example that I used in the programs can be found above(Python,Js,C) to test operator associativity.

Code:

```
var varInt1 uint8 = 22
    var varInt2 uint8 = 24
    var varInt3 uint8 = 24

    if (varInt1 == varInt2) || varInt3 == 24 {
        fmt.Println("Left associativity is working for and") // if it
was right associativity, this line would work
    }else{
        print("Right associativity is working for and")
    }
    if (varInt1 == varInt2) || varInt3 == 24{
        fmt.Println("Right associativity is working") // and and or has
left associativity
    }else{
        fmt.Println("Left associativity is working  for or") // if it
was right associativity, this line would work
    }
    // Left associativity test for not opartor
    if !boolTrue1 || boolTrue2 {
        fmt.Println(" 'Not' has Right associativity")
    }else{
        fmt.Println(" 'Not' has Left associativity")
    }
```

Output:

Left associativity is working for and
Left associativity is working
'Not' has Right associativity

5. Operand evaluation order

In Go Language, based on the code given below, operand evaluation is realized LEFT -> RIGHT. It is understood based on the print statements are written inside the functions etc,. print("Inside Test Function 2") .

Code:

```
boolvar = myFunction1() && myFunction2() && myFunction3() &&
myFunction4()
        fmt.Println("Function-1 and Function-2 and Function-3 and
Function-4: ", boolvar)

        boolvar = true
        boolvar = myFunction1() || myFunction2() || myFunction3() ||
myFunction4();
        fmt.Println("Function-1 or Function-2 or Function-3 or
Function-4: ", boolvar)
```

Output:

Inside Test Function 1
Inside Test Function 2
Function-1 and Function-2 and Function-3 and Function-4: False
Inside Test Function 1
Function-1 or Function-2 or Function-3 or Function-4: True

6. Short-circuit evaluation

In Go Language, while checking the statements, it is started from left to right. If the left side fulfills the condition, there is no need to check the right side that can be seen in the example given below. As a result there is short- circuit evaluation like Python, C, and Javascript.

Code:

```
if myFunction1() || myFunction2() {
            fmt.Println(" Short-circuit evaluation is working ")
        }
```

Output:

Inside Test Function 1
Short-circuit evaluation is working

Rust Language

Code Explanation:

Firstly in Part 1, the OR, AND and NOT operators were compared and tested for their suitability for use in the language. This operation was performed using if-else statements. In the second part, based on the data types found in Rust language, it was tested whether they can be used as boolean operators and which ones are appropriate. In chapter 3, the order of priority among AND, OR, (), and NOT operators was achieved by matching pairs between the 4 given operators. In order to investigate the associativity of the AND, OR, (), and NOT operators mentioned above in Chapter 4, the == signs or variables are replaced in the logic with left | right | They were tested for non-associativity. In Chapter 5, 4 functions are defined to test whether there is a certain order when processing operators, and a statement is written in each of them. In this way, it was possible to test whether the function was working and whether there was any associativity. In the or statement written in Chapter 6, if only the statement written in the first function is output, it is tested that Short-circuit evaluation runs from here.

Compiler URL: https://play.rust-lang.org/

1. *Boolean operators provided*

In Rust language, ***Boolean operators*** that are AND,OR, NOT are tested one by one with different examples. These operators are written as what it is called etc., “true and true”. In this language, any uppercase letter is accepted. Additionally, I got errors for not used variables that are declared in the scope, so in this language, there is no permission for keeping not used variables.

Code:

```
if bool_true1 && bool_true2 == true{
          println!( "Boolean test for True  - True  and operator:
True\n");
       }
       if bool_true1 && bool_false2 == false{
          println!( "Boolean test for True  - False and operator:
False\n");
       }
       println!( "\n");
```

```
        // [ 1.2 or  oparator test ]
        println!( " ----        BOOLEAN OPERATORS: OR       ---- \n");
        if bool_true1 || bool_true2 == true{
            println!( "Boolean test for True  - True  or operator:
True\n");
        }
        if bool_true3 || bool_false1 == true{
           println!( "Boolean test for True  - False or operator: :
True\n");
        }

        println!( "\n");

        // [ 1.3 not oparator test ]
        println!( " ----   BOOLEAN OPERATORS: AND/OR NOT  ---- \n");

        if !(bool_false3){
            println!("Boolean test for Not False for not operator");
        }
        println!( "\n");
        if !(bool_true1 && bool_true2) == false{
            println!( "Boolean test for True  - True  'and not'
operator: False");
        }
        if !(bool_true1 && bool_false1)== true{
                println!( "Boolean test for True  - False 'and not'
operator: True");
        }
```

Output:

--- BOOLEAN OPERATORS: AND ----
Boolean test for True - True and operator: True
Boolean test for True - False and operator: False

---- BOOLEAN OPERATORS: OR ----

Boolean test for True - True or operator: True
Boolean test for True - False or operator: : True

---- BOOLEAN OPERATORS: AND/OR NOT ----

Boolean test for Not False for not operator
Boolean test for True - True 'and not' operator: False
Boolean test for True - False 'and not' operator: True

2. Data types for operands of boolean operators

In Rust language, data types such as int, float,are not used for operands of boolean operators. Only bool operator is used for operands of boolean operators. Additionally, in this language there are no floating numbers such as float, double. Therefore, it cannot be used as a boolean operator as it is in Python and Javascript.

Code:

```
if bool_true3{
          println!( "Boolean test for True boolean variable for boolean
operator is working\n");
       }
       if !bool_false3{
          println!( "Boolean test for False boolean variable for
boolean operator is working\n");
       }
```

Output:

Boolean test for True boolean variable for boolean operator is working
Boolean test for False boolean variable for boolean operator is working

3. Operator precedence rules

In Rust language, Precedence of operators are ranked as 'Parentheses' > AND > OR as can be seen in the below as the same as Python, JavaScript, and C languages.

Code:

```
if bool_true1 && bool_true2  || bool_false1 {
           println!("Boolean test for True and True  or False
condition: True");
       }
       println!("\n");
       if (bool_true1  || bool_true2 && bool_false1) != false {
           println!("Boolean test for True or True  and False
condition: True");
       }
       println!("\n");

       println!(" BOOLEAN OPERATORS: 'AND' VERSUS 'Parantheses'   \n");
       if bool_false1 && bool_false2  || bool_true1 {
           println!("Boolean test for False and False  and True
condition: True");
       }
       println!("\n");
       if !(bool_false1  &&  (bool_false2 ||  bool_true1)) {
           println!("Boolean test for False and (False and True)
condition: False");
       }
```

Output:

BOOLEAN OPERATORS: 'AND' VERSUS 'OR': AND
Boolean test for True and True or False condition: True
Boolean test for True or True and False condition: True

BOOLEAN OPERATORS: 'AND' VERSUS 'Parentheses'

Boolean test for False and False and True condition: True
Boolean test for False and (False and True) condition: False

4. Operator associativity rules

In Rust language, Left associative operators are And, Or. Right associative operator is Not. Non- associative operators are 'Parentheses' that are tested in the given code.

One the tricky point for this language is if-else statements compared the other language with no parenthesis. Additionally, in this language there is no string, just char can be found and because of that I change my example that I used in the programs can be found above(Python,Js,C) to test operator associativity.

Code:

```
let var_int1  = 22;
       let var_int2  = 24;
       let var_int3  = 24;

       if (var_int1 == var_int2) && var_int3 == 24 {
           println!("Right associativity is working for and\n"); // if
it was right associativity, this line would work
       }
       else{
           println!("Left associativity is working for and\n");
       }

       if (var_int1 == var_int2) || var_int3 == 24{
           println!("Left associativity is working\n"); // and and or
has left associativity
       }
       else{
           println!("Right associativity is working  for or\n"); // if
it was right associativity, this line would work
       }

       // Left associativity test for not opartor
       if !bool_true1 || bool_true2 {
           println!(" 'Not' has Right associativity\n");
       }
       else{
           println!(" 'Not' has Left associativity\n\n");
       }
```

Output:

Left associativity is working for and

Left associativity is working

'Not' has Right associativity

5. Operand evaluation order

In Rust language, based on the code given below, operand evaluation is realized LEFT -> RIGHT. It is understood based on the print statements are written

inside the functions etc,. print("Inside Test Function 2") . As a result there is operand evaluation order as similar as Python, C, and Javascript.

Code:

```
let mut bool_var = myFunction1() && myFunction2() && myFunction3() &&
myFunction4();
    println!("{} = Function-1 and Function-2 and Function-3 and
Function-4", bool_var);
    println!("\n");
    bool_var = myFunction1() || myFunction2() || myFunction3() ||
myFunction4();
    println!("{} =Function-1 or Function-2 or Function-3 or
Function-4", bool_var);
```

Output:

```
Inside Test Function 1
Inside Test Function 2
false = Function-1 and Function-2 and Function-3 and Function-4
Inside Test Function 1
true =Function-1 or Function-2 or Function-3 or Function-4
```

6. *Short-circuit evaluation*

In Rust Language, while checking the statements, it is started from left to right. If the left side fulfills the condition, there is no need to check the right side that can be seen in the example given below. As a result there is short- circuit evaluation like Python, C,Go, and Javascript.

Code:

```
println!("  \n6. SHORT - CIRCUIT EVALUATION  ");
        if myFunction1() || myFunction2() {
            println!(" Short-circuit evaluation is working ");
        }
```

Output:

```
Inside Test Function 1
Short-circuit evaluation is working
```

PHP Language

Code Explanation:

Firstly in Part 1, the OR, AND, XOR, and NOT operators were compared and tested for their suitability for use in the language. This operation was performed using if-else statements. In the second part, based on the data types found in Python

language, it was tested whether they can be used as boolean operators and which ones are appropriate. In chapter 3, the order of priority among AND, OR, (), XOR, and NOT operators was achieved by matching pairs between the 4 given operators. In order to investigate the associativity of the AND, OR, (), XOR, and NOT operators mentioned above in Chapter 4, the == signs or variables are replaced in the logic with left | right | They were tested for non-associativity. In Chapter 5, 4 functions are defined to test whether there is a certain order when processing operators, and a statement is written in each of them. In this way, it was possible to test whether the function was working and whether there was any associativity. In the or statement written in Chapter 6, if only the statement written in the first function is output, it is tested that Short-circuit evaluation runs from here.

Compiler URL:
https://paiza.io/projects/H4yZEt7JgEp2onRociF6Gw

1. Boolean operators provided

In PHP language, ***Boolean operators*** that are AND,OR, XOR, NOT are tested one by one with different examples. These operators are written as what it is called etc., “true and true”. This and C language are quite similar in terms of syntax. Compared to other languages, PHP use XOR operator as boolean operator and to the result is seen in integer like 1 or 0.

Code:

```
if ($boolTrue1 && $boolTrue2 == true){
    printf( "Boolean test for True  - True  and operator: True\n");
}
if ($boolTrue1 && $boolFalse1 == false){
    printf( "Boolean test for True  - False and operator: False\n");
}
printf( "\n");

// [ 1.2 or  oparator test ]
printf( " ----       BOOLEAN OPERATORS: OR      ---- \n");
if ($boolTrue1 || $boolTrue2 == true){
    printf( "Boolean test for True  - True  or operator: True\n");
}
if ($boolTrue1 || $boolFalse1 == true){
   printf( "Boolean test for True  - False or operator: : True\n");
}

printf( "\n");

// [ 1.3 not oparator test ]
```

```
 printf( " ----   BOOLEAN OPERATORS: AND/OR NOT  ---- \n");

 if(!($boolFalse1)){
     printf("Boolean test for Not False for not operator");
 }
 printf( "\n");
 if (!($boolTrue1 && $boolTrue2) == false){
     printf( "Boolean test for True  - True  'and not' operator:
False\n");
 }
 if (!($boolTrue1 && $boolFalse1)== true){
         printf( "Boolean test for True  - False 'and not' operator:
True\n");
 }
// [ 1.4 XOR oparator test ]
    printf( "Boolean test for True  - True  'XOR' operator:%d
",$boolTrue1 xor $boolTrue2);
    printf( "\n");
    printf( "Boolean test for True  - False 'XOR' operator:%d ",
$boolTrue1 xor $boolFalse1);
    printf( "\n");
```

Output:

```
---      BOOLEAN OPERATORS: AND    ----
Boolean test for True  - True  and operator: True
Boolean test for True  - False and operator: False

 ----      BOOLEAN OPERATORS: OR      ----

Boolean test for True  - True  or operator: True
Boolean test for True  - False or operator: : True

 ----   BOOLEAN OPERATORS: AND/OR NOT  ----

Boolean test for Not False for not operator
Boolean test for True  - True  'and not' operator: False
Boolean test for True  - False 'and not' operator: True

Boolean test for True  - True  'XOR' operator:0
Boolean test for True  - False 'XOR' operator:1
```

2. Data types for operands of boolean operators

In PHP language, data types such as int, float, boolean are used for operands of boolean operators. For example, 1 for true, 0 for false, float number 1.2 (my example code below) are perceived as True or False. This language is similar to Python and Javascript in terms of data types for boolean operands.

Code:

```
$varFloat = 1.2;
    $varInt = 1;
    $varIntTemp = 0;
    // 1 for True
```

```
    if ($varInt){
        printf( "Boolean test for True = 1 is working\n");
    }
    // 0 for True
    if (!($varIntTemp && $boolFalse1)){
        printf( "Boolean test for False = 0 is working\n");
    }
    if ($varFloat){
        printf( "Boolean test for Float for True\n");
    }
    else{
        printf( "Boolean test for Float for False\n");
    }
    if ($boolTrue3){
       printf( "Boolean test for boolean variable for boolean operator
is working\n");
    }
```

Output:

Boolean test for True = 1 is working
Boolean test for False = 0 is working
Boolean test for Float for True
Boolean test for boolean variable for boolean operator is working

3. Operator precedence rules

Code:

```
$temp1 =  $boolTrue1 && $boolTrue2  || $boolFalse1;
    if($temp1){
        printf("Boolean test for True and True  or False condition:
True");
    }
    printf("\n");
    $temp2 =  $boolTrue1  || $boolTrue2 && $boolFalse1;
    if($temp2 != false){
        printf("Boolean test for True or True  and False condition:
True");
    }
    printf("\n");
    printf(" BOOLEAN OPERATORS: 'AND' VERSUS 'Parantheses'   \n");
    $temp1 =  $boolFalse1 && $boolFalse2  || $boolTrue1;
    if($temp1){
        printf("Boolean test for False and False  and True condition:
True");
    }
    printf("\n");
    $temp1 =  $boolFalse1  &&  ($boolFalse2 ||  $boolTrue1);
    if(!$temp1){
        printf("Boolean test for False and (False and True) condition:
False");
    }
```

Output:

BOOLEAN OPERATORS: 'AND' VERSUS 'OR': AND

Boolean test for True and True or False condition: True
Boolean test for True or True and False condition: True

BOOLEAN OPERATORS: 'AND' VERSUS 'Parentheses'

Boolean test for False and False and True condition: True
Boolean test for False and (False and True) condition: False

4. Operator associativity rules

In PHP language, Left associative operators are And, Or. Right associative operator is Not. Non- associative operators are 'Parentheses' that are tested in the given code.

Code:

```
$varInt1  = 22;
    $varInt2  = 24;
    $varInt3  = 24;
    if (($varInt1 == $varInt2) && $varInt3 == 24) {
        printf("Right associativity is working for and\n"); // if it
was right associativity, this line would work
    }
    else{
        printf("Left associativity is working for and\n");
    }
    if (($varInt1 == $varInt2) || $varInt3 == 24){
        printf("Left associativity is working\n"); // and and or has
left associativity
    }
    else{
        printf("Right associativity is working  for or\n"); // if it
was right associativity, this line would work
    }
    // Left associativity test for not opartor
    if (!$boolTrue1 || $boolTrue2) {
        printf(" 'Not' has Right associativity\n");
    }
    else{
        printf(" 'Not' has Left associativity\n\n");
    }
```

Output:

Left associativity is working for and

Left associativity is working

'Not' has Right associativity

5. Operand evaluation order

In PHP language, based on the code given below, there is no order for the operand evaluation like C language. There is no Left->Right or Right->Left concept.

To clarify that, both for AND and OR operators the same statement is tested with one ‘()’ difference. As a result, there were no changes, so that there is no order for the operand evaluation unlike Python, Javascript, and Go.

Code:

```
function  myFunction1()  {
            $boolvar = true;
            printf("Inside Test Function 1\n");
            return $boolvar;
        }
        function  myFunction2() {
            $boolvar = false;
            printf("Inside Test Function 2\n");
            return $boolvar;
        }
        function  myFunction3() {
            $boolvar  = true;
            printf("Inside Test Function 3\n");
            return $boolvar;
        }
        function  myFunction4() {
            $boolvar = false;
            printf("Inside Test Function 4\n");
            return $boolvar;
        }
        printf("\n  5. OPERAND EVALUATION ORDER \n");
        $boolvar2 = true;
        printf("Same statement with no '()'\n");
        $boolvar2 = myFunction1() && myFunction2() && myFunction3() &&
myFunction4();
        printf("Same statement no '()'\n");
        $boolvar2 = (myFunction1() && myFunction2()) && myFunction3()
&& myFunction4(); // same results are taken for and
        printf("Same statement with no '()'\n");
        $boolvar2 = myFunction1() || myFunction2() || myFunction3() ||
myFunction4();
        printf("Same statement no '()'\n");
        $boolvar2 = (myFunction1() || myFunction2()) || myFunction3()
|| myFunction4(); // same results are taken for or
```

Output:

```
Same statement with no '()'
Inside Test Function 1
Inside Test Function 2
Same statement no '()'
Inside Test Function 1
Inside Test Function 2
Same statement with no '()'
Inside Test Function 1
Same statement no '()'
Inside Test Function 1
```

6. Short-circuit evaluation

In PHP Language, while checking the statements, it is started from left to right. If the left side fulfills the condition, there is no need to check the right side that can be seen in the example given below. As a result there is short- circuit evaluation like Python, C, and Javascript.

Code:

```
if (myFunction1() || myFunction2()) {
            printf(" Short-circuit evaluation is working ");
        }
```

Output:

```
Inside Test Function 1
Short-circuit evaluation is working
```

Learning Methodology

While I was learning boolean operators for the homework, I read the slides, documentations, and example codes from the Internet. After that, one by one I completed the languages for the six following design issues. For the 5th point, in class, it is discussed and learned what to do.

Conclusion (Readability & Writability)

The syntax of PHP Language is very similar to C language, however, I think it is hard to understand data type due to absence of identifiers. Most convenient language to write is Python for boolean operators since in total, the data types can be used in range like int, float, boolean, and also it costs not much lines of code. Secondly, in terms of writability, from my background I knew Javascript language, however, I think it is hard to read as to show in the webpage, there is need for the little bit of html knowledge. Last point to clarify my point for writability, In Go and Rust language, there is a restriction about not used variables, and I think it affects the writability. Additionally, in terms of writability, PHP has quite a lot of choices for boolean expressions and also compared to other 5 languages, it has XOR and, &&, or, ||.