|  |  |  |
| --- | --- | --- |
|  | FACULTY OF ENGINEERING TECHNOLOGY, UNIVERSITI TEKNIKAL MALAYSIA MELAKA | |
| DISCRETE MATHEMATICS | | |
| BEEC 3413 | SEMESTER 1 | SESI 2021/2022 |
| LAB 1: LOGICAL OPERATORS | | |
| NAME AND MATRIX NUMBER |  |  |
|  |  |
|  |  |
| COURSE | BEEC | |
| DATE | OCT 7, 2021 | |
| NAME OF INSTRUCTOR | AZMAN AWANG TEH | |
| EXAMINER’S COMMENT | | VERIFICATION STAMP |
| TOTAL MARKS |

## OBJECTIVES

1. To familiarize students with R programming Language.
2. To understand the used of logical operators in programming.

## EQUIPMENTS

1. Personal Computer.
2. R Software.
3. SYNOPSIS & THEORY
   1. INTRODUCTION TO R

R is a **free software** environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS.

Website Link: <http://www.r-project.org/>

* 1. VARIABLES IN R
     1. VECTOR

Vectors can be thought of as contiguous cells containing data. Cells are accessed through indexing operations such as x [5].

R has six basic ("atomic") vector types: logical, integer, real, complex, string (or character) and raw. The modes and storage modes for the different vector types are listed in the follow- ing table.

|  |  |  |
| --- | --- | --- |
| **typeof** | **mode** | **storage.mode** |
| logical | logical | logical |
| integer | numeric | integer |
| double | numeric | double |
| complex | complex | complex |
| character | character | character |
| raw | raw | raw |

These are several example of vectors in R

1. *a* <- *c*(1, 2, 5.3, 6, −2, 4) # numeric vector

1. *b* <- *c*("one", "two", "three") # character vector
2. *c* <- *c*(TRUE, TRUE, TRUE, FALSE, TRUE, FALSE) # logical vector
   * 1. INDEXING

R contains several constructs which allow access to individual elements or subsets through indexing operations. In the case of the basic vector types one can access the i-th element using x[i], but there is also indexing of lists, matrices, and multi-dimensional arrays. There are several forms of indexing in addition to indexing with a single integer. Indexing can be used both to extract part of an object and to replace parts of an object (or to add parts).

R has three basic indexing operators, with syntax displayed by the following examples

# x[i] x[i, j]

x[[i]]

x[[i, j]] x$a

x$"a"

For vectors and matrices the [[ forms are rarely used, although they have some slight se- mantic differences from the [ form (e.g. it drops any names or dimnames attribute, and that partial matching is used for character indices). When indexing multi-dimensional structures with a single index, x[[i]] or x[i] will return the ith sequential element of x.

For lists, one generally uses [[ to select any single element, whereas [ returns a list of the selected elements.

The [[ form allows only a single element to be selected using integer or character indices, whereas [ allows indexing by vectors. Note though that for a list or other recursive object, the index can be a vector and each element of the vector is applied in turn to the list, the selected component, the selected component of that component, and so on. The result is still a single element.

The form using $ applies to recursive objects such as lists and pairlists. It allows only a lit- eral character string or a symbol as the index. That is, the index is not computable: for cases where you need to evaluate an expression to find the index, use x[[expr]]. When $ is applied to a non-recursive object the result used to be always NULL: as from R 2.6.0 this is an error.

* 1. OPERATORS

R contains a number of operators. They are listed in the table below.

- Minus, can be unary or binary

+ Plus, can be unary or binary

! Unary not

˜ Tilde, used for model formulae, can be either unary or binary

? Help

: Sequence, binary (in model formulae: interaction)

\* Multiplication, binary

/ Division, binary

ˆ Exponentiation, binary

%x% Special binary operators, x can be replaced by any valid name

%% Modulus, binary

%/% Integer divide, binary

%\*% Matrix product, binary

%o% Outer product, binary

%x% Kronecker product, binary

%in% Matching operator, binary (in model formulae: nesting)

< Less than, binary

> Greater than, binary

== Equal to, binary

>= Greater than or equal to, binary

<= Less than or equal to, binary & And, binary, vectorized

&& And, binary, not vectorized

| Or, binary, vectorized

|| Or, binary, not vectorized

<- Left assignment, binary

-> Right assignment, binary

$ List subset, binary

* 1. IF

The if/else statement conditionally evaluates two statements. There is a *condition* which is evaluated and if the *value* is TRUE then the first statement is evaluated; otherwise the second statement will be evaluated. The if/else statement returns, as its value, the value of the statement that was selected. The formal syntax is

# if ( statement1 ) statement2

else

statement3

First, *statement1* is evaluated to yield *value1*. If *value1* is a logical vector with first element TRUE then *statement2* is evaluated. If the first element of *value1* is FALSE then *statement3* is evaluated. If *value1* is a numeric vector then *statement3* is evaluated when the first element of *value1* is zero and otherwise *statement2* is evaluated. Only the first element of *value1* is

used. All other elements are ignored. If *value1* has any type other than a logical or a numeric vector an error is signalled.

if/else statements can be nested.

# if ( statement1 ) statement2

else if ( statement3 ) statement4

else if ( statement5 ) statement6

else

statement8

* 1. FUNCTION

The R language allows the user to create objects of mode *function*. These are true R functions that are stored in a special internal form and may be used in further expressions and so on. In the process, the language gains enormously in power, convenience and elegance, and learn- ing to write useful functions is one of the main ways to make your use of R comfortable and productive.

It should be emphasized that most of the functions supplied as part of the R system, such as mean(), var(), postscript() and so on, are themselves written in R and thus do not differ materially from user written functions.

A function is defined by an assignment of the form

# > name <- function(arg\_1, arg\_2, ...) expression

The *expression* is an R expression, (usually a grouped expression), that uses the arguments,

*arg\_i*, to calculate a value. The value of the expression is the value returned for the function.

A call to the function then usually takes the form name(expr\_1, expr\_2, âĂę) and may occur anywhere a function call is legitimate.

## PROCEDURE

4.1 LOGICAL FUNCTION

1. Open R Gui, and create a new script.
2. Type the following program code, of **and** function with only using If statement and without using any **Logical Operators**.

# dan <- function(x,y){ z = F

if (x==T){if (y==T) {z=T}} return(z)

}

1. Run the program and observe the output.
2. Create new functions for the following logical expressions:
   * ¬*p*, negation *p*.
   * *p q*, *p* exclusive-or *q*.
   * *p*  *q*, *p* disjunction *q*.
   * *p* → *q*, if *p* then *q*.
   * *p* ↔ *q*, *p* bi-conditional *q*.
   * (¬*p*  ( ¬*q* *r*) ) (*r*  *q*) (*r*  *p*).
   * (*p* (*q* → ( *r* ¬ *p*)))↔ ( *q* ¬*s*)

## RESULT

1. Follow all the procedures
2. copy paste all your source codes and output as a result.

## DISCUSSION

Kindly discuss on the lab result and findings.

## CONCLUSION

Conclude what you have learned in this lab session.

## REFERENCE

* The R Manuals, <http://www.r-project.org/>