Application Development Laboratory (CS 33002)

KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY

School of Computer Engineering

Strictly for internal circulation (within KIIT) and reference only. Not for outside circulation without permission

2 Credit

Analytics Application Development using R

Lab Contents

Sr #	Major and Detailed Coverage Area	Lab#
1	Decision Making ❑ simple If ❑ if…else if…else ❑ switch	2
2	Loops ❑ while ❑ for ❑ repeat	

Lab Contents

Sr #	Major and Detailed Coverage Area	Lab#
3	Function ❑ Function Definition ❑ Function Declaration ❑ Function Components ❑ Function Calling ❑ Named Arguments ❑ Default Value of Arguments ❑ Built-in Function, User-defined Function ❑ Recursion	2

Decision Making

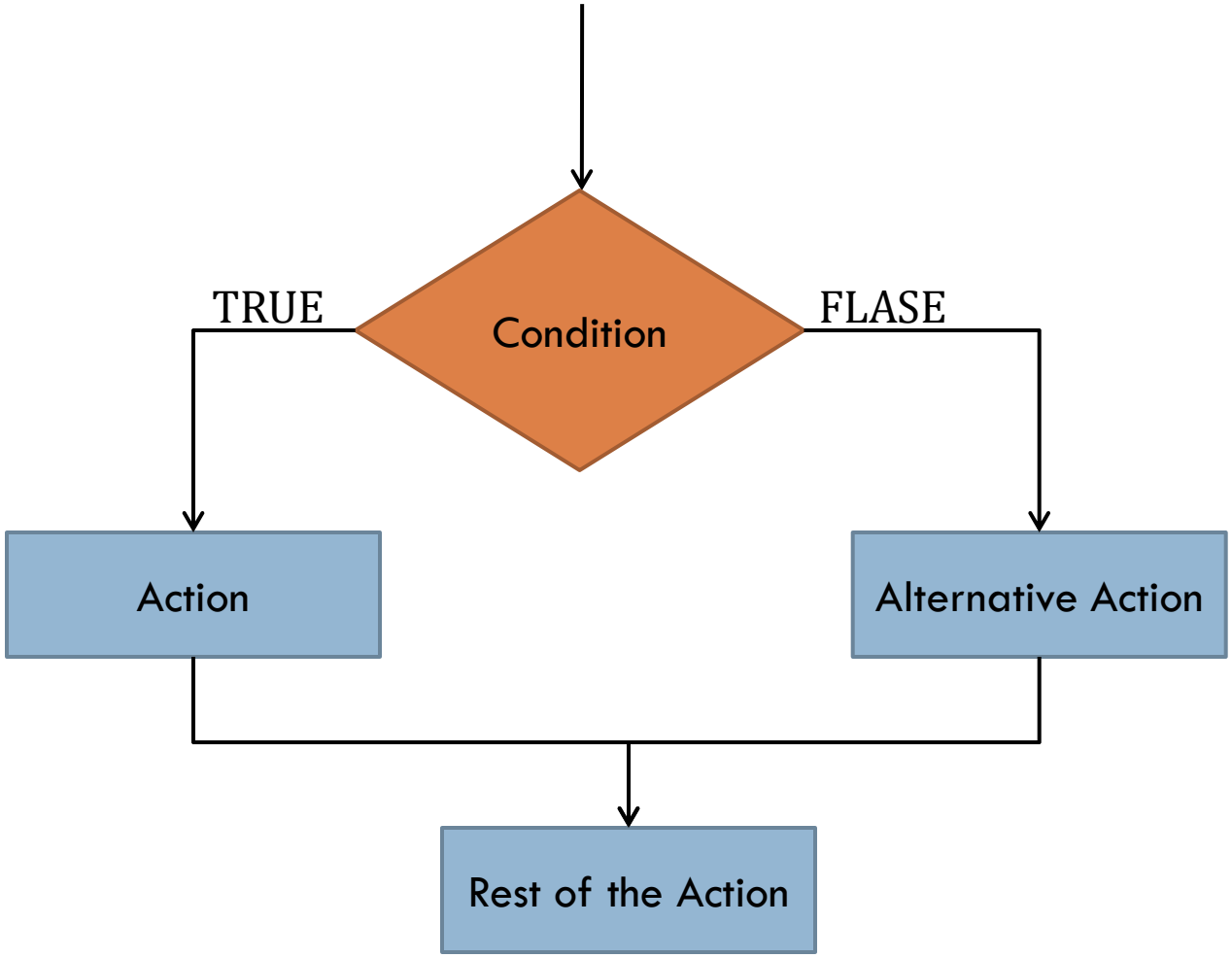
Decision making structures require the programmer to specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

if it's raining
  grab an umbrella
  put on boots
otherwise
  wear sunglasses
  put on sneakers
go to Lab

Things we do when raining

Things we do when it's not raining

Decision Making cont'd



Decision Making Statement

R provides the following types of decision making statements.

Sr#	Statement	Description
1	if	An if statement consists of a Boolean expression followed by one or more statements.
2	if … else	An if statement can be followed by an optional else statement, which executes when the Boolean expression is false.
3	switch	A switch statement allows a variable to be tested for equality against a list of values.

if statement

The basic syntax for creating an **if** statement in R is:

Syntax:

```
if(boolean_expression)
{
  // statement(s) will execute if the boolean expression is true.
}
```

Example:

```
x <- 30L
if(is.integer(x))
{
   print("X is an Integer")
}
```

if... else statement

The basic syntax for creating an **if...else** statement in R is:

Syntax:

```
if(boolean_expression) {
  // statement(s) will execute if the boolean expression is true.
}
else {
 // statement(s) will execute if the boolean expression is false.
}
```

Example:

```
x <- c("what","is","truth")
if("Truth" %in% x){
   print("Truth is found")
} else {
print("Truth is not found")
}
```

if...else if...else statement

When using if, else if, else statements there are few points to keep in mind.

- An if can have zero or one else and it must come after any else if's.
- An if can have zero to many else if's and they must come before the else.
- Once an else if succeeds, none of the remaining else if's or else's will be tested.

Syntax:

```
if(boolean_expression 1) {
// Executes when the boolean expression 1 is true.
}else if( boolean_expression 2) {
// Executes when the boolean expression 2 is true.
}else if( boolean_expression 3) {
// Executes when the boolean expression 3 is true.
}else {
// executes when none of the above condition is true.
}
```

if...else if...else Example


```
x <- c("what","is","truth")
if("Truth" %in% x){
print("Truth is found the first time")
} else if ("truth" %in% x) {
print("truth is found the second time")
} else {
print("No truth found")
}
```

switch statement

The basic syntax for creating a switch statement in R is :

switch(expression, case1, case2, case3....)

There are two ways in which one of the cases is selected.

- **Based on Index** – If the cases are just values (like a Character Vector), and if the expression is evaluated to a number, the expression's result is used as index to select the case.
- **Based on Matching Value** – If the cases have both case value and output value like ["case_1"="value1"], then the expression value is matched against case values, and the matching case is the output.

switch statement – Based on Index


```
# R program to switch statement - Based on Index
y = 3

x = switch(
   y,
   "Good Morning",
   "Good Afternoon",
   "Good Evening",
   "Good Night"
)

print (x)
```

switch statement – Based on Value


```
# R program to switch statement - Based on matching value
y = "12"

x = switch(
   y,
   "9"="Good Morning",
   "12"="Good Afternoon",
   "18"="Good Evening",
   "21"="Good Night"
)

print (x)
```

switch Statement Rules

The following rules apply to a switch statement:

- If there is more than one match, the first matching element is returned
- If the value of the integer is between 1 and nargs()-1 (The max number of arguments) then the corresponding element of case condition is evaluated and the result returned.
- Any number of case statements are allowed within a switch.
- If expression evaluates to a character string then that string is matched (exactly) to the names of the elements.
- No default argument is available.
- If the value of expression is not a character string it is coerced to integer.

Loops

Repeating execution of a block of statements in a controlled way is an important aspect in any functional programming language. It helps to keep the code concise and clean. Programming languages provide various control structures that allow for more complicated execution paths. A loop statement allows to execute a statement or group of statements multiple times. R programming language provides the following kinds of loop to handle looping requirements

Sr#	Statement	Description
1	repeat	Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
2	while	Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
3	for	Like a while statement, except that it tests the condition at the end of the loop body.

repeat Loop

The basic syntax for creating a repeat loop in R is:

```
repeat
{
  commands or block of statements
  if(condition)
  {
    break
  }
}
```

Breaking a condition: Breaking is done using an R if statement. It is optional. But if breaking condition is not placed in the repeat loop statement, the statements in the repeat block will get executed for ever in an infinite loop. Breaking Condition should return a boolean value, either TRUE or FALSE. The placement of breaking condition is up to the developer. It can be kept before the “block of statements” block or after it.

repeat Loop Example

Example 1

```
v <- c("Hello","loop")
cnt <- 2
repeat
{
 print(v)
 cnt <- cnt+1
 if(cnt > 5)
 {
  break
 }
}
```

Example 2

```
a = 1
repeat
{
   # starting of repeat statements block
   print(a)
   a = a+1
   # ending of repeat statements block
   if(a>6) # breaking condition
   {
      break
   }
}
```

while Loop

The basic syntax for creating a while loop in R is :

```
while (test_expression) {
   statement
}
```

As long as the test boolean expression evaluates to TRUE, the statements inside the while block are executed. The point at which the boolean expression results FALSE, the execution flow is out of the while loop statement.

Example 1

```
v <- c("Hello","while loop")
cnt <- 2
while (cnt < 7){
print(v)
cnt = cnt + 1
}
```

Example 2

```
a=1
while (a < 10){
print(a)
a = a + 1
}
```

break statement in while Loop

Break Statement is a loop control statement which can be used to terminate the while loop.

Example:

```
a = 1
b = 4


while(a<5)
{
  if(b+a>6)
   {
      break
   }
}
```

for Loop

The basic syntax for creating a for loop statement in R is:

```
for (value in vector) {
  statements
}
```

The statements in the for loop are executed for each element in vector and when there are no further elements in the vector, the execution control is out of the for loop and continues executing statements after for loop. R's for loops are particularly flexible in that they are not limited to integers, character vectors, logical vectors, lists or expressions can be passed.

Example 1

```
a = c(2, 45, 9, 12)
for(i in a) {
   print(i)
}
```

Example 2

```
v <- LETTERS[1:4]
for ( i in v) {
print(i)
}
```

break statement in for Loop

Break Statement can be used to terminate the for loop.

Example:

```
a = c(12, 45, 9, 12)

for(i in a)
{
   if(i==9)
   {
      break
   }
   print(i)
}
```

Loop Control Statement

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed. R supports the following control statements.

Sr#	Statement	Description
1	break	Terminates the loop statement and transfers execution to the statement immediately following the loop.
2	next	The next statement simulates the behavior of R switch.

next Example

```
v <- LETTERS[1:6]
for ( i in v){
if (i == "D"){next}
print(i)
}
```

Function Definition

A function is a set of statements organized together to perform a specific task. Functions are used to logically break our code into simpler parts which become easy to maintain and understand.

In R, a function is an object so the R interpreter is able to pass control to the function, along with arguments that may be necessary for the function to accomplish the actions.

The function in turn performs its task and returns control to the interpreter as well as any result which may be stored in other objects.

Function Declaration

The basic syntax of an R function definition is as follows:

```
function_name <- function(arg_1, arg_2, ...)
{
   function body
}
```

- The reserved word **function** is used to declare a function.
- The statements within the curly braces form the body of the function. These braces are optional if the body contains only a single expression.
- Finally, this function object is given a name by assigning it to a variable, func_name.

Function Component

The different parts of a function are:

- **Function Name:** This is the actual name of the function. It is stored in R environment as an object with this name.
- **Arguments:** An argument is a placeholder. When a function is invoked, you pass a value to the argument. Arguments are optional; that is, a function may contain no arguments. Also arguments can have default values.
- **Function Body:** The function body contains a collection of statements that defines what the function does.
- **Return Value:** The return value of a function is the last expression in the function body to be evaluated.

Function Example

Example 1: function to print x raised to the power y

```
pow <- function(x, y)
{
  result <- x^y
  print(paste(x,"raised to the power", y, "is", result))
}
```

Example 2: function for conversion from fahrenheit to kelvin

```
fahrenheit_to_kelvin <- function(temp_F)
{
  temp_K <- ((temp_F - 32) * (5 / 9)) + 273.15
  return(temp_K)
}
```

Note: In R, it is not necessary to include the return statement. R automatically returns whichever variable is on the last line of the body of the function. While in the learning phase, we will explicitly define the return statement.

Calling a Function

We can call the above function as follows.

Example 1: calling pow

```
pow(8, 2)
[1] "8 raised to the power 2 is 64"
```

```
pow(2, 8)
[1] "2 raised to the power 8 is 256"
```

Example 2: calling fahrenheit_to_kelvin

```
# freezing point of water
output = fahrenheit_to_kelvin(32)
print(output)
[1] 273.15

# boiling point of water
fahrenheit_to_kelvin(212)
[1] 373.15
```

```
# freezing point of water
print(fahrenheit_to_kelvin(32))
[1] 273.15
```

Lab Work

Write a function called fence that takes two vectors as arguments, called original and wrapper, and returns a new vector that has the wrapper vector at the beginning and end of the original.

Solution

```
best_practice <- c("Write", "programs", "for", "people", "not", "computers")
asterisk <- "***"  # R interprets a variable with a single value as a vector with one element.
print(best_practice, asterisk)

fence <- function(original, wrapper)
{
 answer <- c(wrapper, original, wrapper)
 return(answer)
}
```

Lab Work

Write a function called fence that takes one vector as argument, and that returns a vector made up of just the first and last elements of its input.

Solution

Named Arguments

In the function calls, the argument matching of formal argument to the actual arguments takes place in positional order. This means that, in the call pow(8,2), the formal arguments x and y are assigned 8 and 2 respectively.

We can also call the function using **named arguments**. When calling a function in this way, the order of the actual arguments doesn't matter. For example, all of the function calls given below are equivalent.

```
pow(8, 2)

[1] "8 raised to the power 2 is 64“

pow(x = 8, y = 2)

[1] "8 raised to the power 2 is 64“

pow(y = 2, x = 8)

[1] "8 raised to the power 2 is 64"
```

Named Arguments cont'd

Furthermore, we can use named and unnamed arguments in a single call.

In such case, all the named arguments are matched first and then the remaining unnamed arguments are matched in a positional order.

```
pow(x=8, 2)
[1] "8 raised to the power 2 is 64“
pow(2, x=8)
[1] "8 raised to the power 2 is 64"
```

In all the examples above, x gets the value 8 and y gets the value 2.

Default Value for Arguments

Default values to arguments can be assigned in a function in R. This is done by providing an appropriate value to the formal argument in the function declaration. Below is the function with a default value for y.

```
pow <- function(x, y = 2)
{
  result <- x^y
  print(paste(x,"raised to the power", y, "is", result))
}
```

The use of default value to an argument makes it optional when calling the function.

```
pow(3)
[1] "3 raised to the power 2 is 9“
pow(3,1)
[1] "3 raised to the power 1 is 3"
```

Built-in Function

R has many **in-built** functions which can be directly called in the program without defining them first. We can also create and use our own functions referred as **user defined functions**.

Built-in Function: Simple examples of in-built functions are seq(), mean(), max(), sum(x) and paste(...) etc. They are directly called by user written programs.

Examples:

```
# Create a sequence of numbers from 32 to 44.
print(seq(32,44))
# Find mean of numbers from 25 to 82.
print(mean(25:82))
# Find sum of numbers frm 41 to 68.
print(sum(41:68))
```

User-defined Function

We can create user-defined functions in R. They are specific to what a user wants and once created they can be used like the built-in functions. Below is an example of how a function is created and used.

```
# Create a function to print squares of numbers in sequence.
new.function <- function(a)
{
  for(i in 1:a)
  {
    b <- i^2
    print(b)
  }
}
# Call the function new.function supplying 6 as an argument.
new.function(6)
```

Recursive Function

A function that calls itself is called a recursive function and this technique is known as recursion. This special programming technique can be used to solve problems by breaking them into smaller and simpler sub-problems.

```
# Recursive function to find factorial
recursive.factorial <- function(x)
{
    if (x == 0)
        return (1)
    else
        return (x * recursive.factorial(x-1))
}
# Call the function recursive.factorial supplying 5 as an argument.
recursive.factorial(5)
[1] 120
```

Thank You
End of Lab 2.1

Lab Experiments

1. Write an R-script to analyze the given number is positive using simple if statement.
2. Write an R-script to check whether the given number is positive or not using if…else statement.
3. Write an R-script to analyze whether the given year is a leap year or not?
4. Write an R-script to enter two numbers and find out the biggest one.
5. Write an R-script to enter a 3-digits number and check whether it is palindrome no. or not?
6. Write an R-script to enter marks in 3 subjects and then calculate the total mark and average. Assign the grade according to the B.Tech evaluation system.
7. Write an R-script to design a menu driven program as follows and then evaluate any one of the operation according to your choice using switch case statement.
   1. Area of circle, 2. Area of rectangle, 3. Area of Triangle

Lab Experiments cont…

8. Write an R-script to design a menu driven program as follows and then display any one of the color according to your choice using switch case statement.

   R- Red

   G- Green

   B- Blue

9. Write an R-script to generate the number series as follows using while loop- 1 4 9……n^2
10. Write an R-script to find out the factorial of the given no. using for loop
11. Write an R-script to generate the Fibonacci series up to n terms.
12. Write an R-script to check whether a number n is prime number or not
13. Write an R-script to check whether an input integer is perfect number or not.
14. Write an R-script to sum the series S=1+(1+2)+(1+2+3)+…+(1+2+3+…+n)

Lab Experiments cont…

15. Write an R-script to reverse the number
16. Write an R-script to check whether an integer number is an Armstrong number or not. If sum of cubes of each digit of the number is equal to the number itself, then the number is called an Armstrong number. For example, 153 = (1 * 1 * 1) + (5 * 5 * 5) + (3 * 3 * 3)
17. Write an R-script to print the following pattern for n rows. Ex. for n=5 rows

    1

    2 1

    1 2 3

    4 3 2 1

    1 2 3 4 5

Lab Experiments cont…

18. Write an R-script to evaluate average of 3 numbers using function
19. Write an R-script to find out the factorial of a number using function
20. Write an R-script to find out HCF and LCM of the given two numbers using function
21. Write an R-script to evaluate sum of the following series using recursive function 1+2+3+…….. +N
22. Write an R-script to display the reverse of the given no. using recursive function
23. Write an R-script to evaluate the simple interest of the given P, T, R using function, where function takes the default value for R
24. Write an R-script to convert decimal into binary using recursive function
25. Write an R-script to find the factorial of a number using recursive function
26. Write an R-script to find the Find Sum of Series $1^2+2^2+3^2+.....+n^2$ using recursive function
27. Write an R-script to develop a function that receives 5 numbers and display the sum, average and standard deviation of these numbers.

Lab Experiments cont…

28. Write an R-script to generate a set of numbers and run the numbers with the following built-in statistic functions.

Function	Description
mean(x)	Mean of the numbers in vector x
median(x)	Median of the numbers in vector x
var(x)	Variance of the numbers in vector x
sd(x)	Standard deviation of the numbers in vector x
scale(x)	Standard scores (z-scores) of the numbers in vector x
summary(x)	max(x) of the numbers in vector x min(x) of the numbers in vector x
rank(x)	Ranks of the numbers (in increasing order) in vector x.
quantile(x)	The 0^{th}, 25^{th}, 50^{th}, 75^{th}, and 100^{th} percentiles (i.e. the quartiles) of the numbers in vector x.