Practical No. 5 Date: / / 201

Aim: Program to perform vector operations on Five day Earning of a Person.

Objectives:

- To study R Vectors & its operations.
- Implement a program to perform operations on Earning vector.

Theory:

R Vectors

Vectors are the most basic R data objects. Vector contain elements of same type. Vector performs automatic coercion (conversion) of elements if necessary. There are six types of atomic vectors:

- Logical
- Integer
- Double
- Complex
- Character
- Raw

Creating Vector

Single Element Vector

```
print("abc")                  # Atomic vector of type character.
print(12.5)                   # Atomic vector of type double.
print(63L)                    # Atomic vector of type integer.
print(TRUE)                   # Atomic vector of type logical.
print(2+3i)                   # Atomic vector of type complex.
print(charToRaw('hello'))     # Atomic vector of type raw (68 65 6c 6c 6f).
```

Multiple Element Vector

```
V <- 5:13                     # Creating a sequence from 5 to 13.
V <- 6.6:12.6                 # Creating a sequence from 6.6 to 12.6.
V <- 3.8:11.4                 # Creating a sequence from 3.8 to 10.8 (11.4 doesn't
                              belongs to sequence).
```

Using seq()

```
print(seq(5, 9, 0.4))
# Creates vector from 5 to 9 with 0.4 increment, output will be
[1] 5.0 5.4 5.8 6.2 6.6 7.0 7.4 7.8 8.2 8.6 9.0
```

Using c()

```
S <- c('apple', 'red', 5, TRUE)
print(S)
```

```
# The logical and numeric values are converted to characters, i.e.
[1] “apple” “red” “5” “TRUE”
```

Naming Vector

Suppose, we have a vector remain representing cards remained in suits after some of the cards are drawn.

```
remain <- c(11, 12, 11, 13)
# Option #1
suits <- c("spades", "hearts", "diamonds", "clubs")
names(remain) <- suits
# Option #2
remain <- c(spades=11, hearts=12, diamonds=11, clubs=13)
# Option #3
remain <- c("spades"=11, "hearts"=12, "diamonds"=11, "clubs"=13)
print(remain)
[1] spades hearts diamonds clubs
        11     12       11    13
```

Vector Arithmetic

Arithmetic operations on vectors are performed element-wise.

Single Vector Examples

```
earnings <- c(50, 100, 30)
earnings + 100    # Vector Addition
[1] 150     200   130
earnings - 20     # Vector Subtraction
[1] 30      80    10
earnings * 3      # Vector Multiplication
[1] 150     300   90
earnings / 10     # Vector Division
[1] 5 10    3
earnings ^ 2      # Vector Exponent
[1] 2500    10000 900
```

Multiple Vector Examples

```
earnings <- c(50, 100, 30)
expenses <- c(30, 40, 80)
earnings + c(10, 20, 30)     # Vector Addition
[1] 60      120   60
earnings - expenses    # Vector Subtraction
[1] 20      60    -50
earnings * c(1, 2, 3)  # Vector Multiplication
```

```
[1] 50     200   90
earnings / c(1, 2, 3)  # Vector Division
[1] 50     50    10
```

Accessing (Subsetting) Vector

Elements of a Vector are accessed using indexing. The [ ] brackets are used for indexing. Indexing starts with position 1. Elements of a Vector can also be accessed using names. Giving a negative value in the index drops that element from result. TRUE, FALSE or 0 and 1 can also be used for indexing.

By Index (Position)

```
remain <- c(spades = 11, hearts = 12,diamonds = 11, clubs = 13)
# Access element at index 1
print(remain[1])
[1] spades
        11
# Access element at index 3
print(remain[3])
[1] diamonds
          11
```

By Name

```
remain <- c(spades = 11, hearts = 12,diamonds = 11, clubs = 13)
# Access “spades”
print(remain[“spades”])
[1] spades
        11
# Access “diamonds”
print(remain[“diamonds”])
[1] diamonds
           11
```

Accessing Multiple Elements

```
remain <- c(spades = 11, hearts = 12,diamonds = 11, clubs = 13)
# Access only black cards
remain_black <- remain[c(1,4)]
print(remain_black)
[1] spades  clubs
         11        13
# Order of selection matters
print(remain[c(4,1)])
[1] clubs   spades
```

```
         13          11
print(remain[c("clubs", "spades")])
[1] clubs   spades
         13          11
```

Dropping Element(s)

```
remain <- c(spades = 11, hearts = 12,diamonds = 11, clubs = 13)
# Access all elements but index 1 element
print(remain[-1])
[1] hearts  diamonds clubs
         12           11    13
# Dropping multiple elements
print(remain[-c(1,2)])
[1] diamonds clubs
           11    13
# Dropping element with name results in error
print(remain[-"spades")])
Error in -"spades" : invalid argument to unary operator
```

By Logical Value

```
remain <- c(spades = 11, hearts = 12,diamonds = 11, clubs = 13)
# Access second & fourth elements
print(remain[c(FALSE, TRUE, FALSE, TRUE)])
[1] hearts  clubs
         12        13
# Logical Vector using TRUE/1 or FALSE/0
selection_vector <- c(0, 1, 0, 1)
print(remain[selection_vector])
[1] hearts  clubs
         12        13
```

Recycling Vector Elements

With Multiple Vectors

```
V1 <- c(3,8,4,5,0,11)
V2 <- c(4,11)
add <- V1 + V2          # V2 <- c(4,11,4,11,4,11)
print(result)
[1] 7 19 8 16 4 22
sub <- V1 - V2          # V2 <- c(4,11,4,11,4,11)
print(result)
```

```
[1] -1 -3 0 -6 -4 0
With Logical Vector
remain <- c(spades = 11, hearts = 12,diamonds = 11, clubs = 13)
# R recycles c(T,F) to c(T,F,T,F)
print(remain[c(TRUE, FALSE)])
[1] spades  diamonds
         11           11
# R recycles c(T,F,T) to c(T,F,T,T)
print(remain[c(TRUE, FALSE, TRUE)])
[1] spades  diamonds clubs
         11           11    13
```

sort() Function

With Numeric Elements

```
V <- c(3, 8, 4, 5, 0, 11, -9, 304)
# Sort the elements of the vector
sortedV <- sort(V)
print(sortedV)
[1] -9 0 3 4 5 8 11 304
# Sort the elements in the reverse order.
revSortedV <- sort(V,decresing = TRUE)
print(revSortedV)
[1] 304 11 8 5 4 3 0 -9
```

With Character Elements

```
V <- c("Red", "Blue", "Yellow", "Violet")
# Sorting character vector
sortedV <- sort(V)
print(sortedV)
[1] "Blue" "Red" "Violet" "Yellow"
# Sorting character vector in the reverse order.
revSortedV <- sort(V,decresing = TRUE)
print(revSortedV)
[1] "Yellow" "Violet" "Red" "Blue"
```

Algorithm

1. Start.
2. Create a vector "Earnings" containing five numeric elements.
3. Name vector elements.
4. Read choice for vector operations from menu as

a. Arithmetic Operation
b. Access Element
c. Sum
d. Sort
e. Display

5. As per choice perform stack operations as

a. If choice is "a", perform any one arithmetic operation.
b. If choice is "b", access any one element using any one method.
c. If choice is "c", use sum() function to add elements together.
d. If choice is "d", use sort() function for sorting of elements (ascending/decending).
e. If choice is "e", display the contents of given vector.

6. Stop.