V. Venkata Chandu

192210078

# ACTIVITY 2

1)

**Question 1**

Analyze the following R program and identify the correct output. This program demonstrates why R does not support pointers.

R Program:

```
x <- 10
y <- x
y <- y + 5
print(x)
print(y)
z <- y
z <- z * 2
print(z)
print(x)
print(y)
```

Output:

```
> x <- 10
> y <- x
> y <- y + 5
> print(x)
[1] 10
> print(y)
[1] 15
> z <- y
> z <- z * 2
> print(z)
[1] 30
> print(x)
[1] 10
>
```

2)

## Question 2

Analyze the behavior of variable assignment in R and choose the correct output.

R Program:

```
a <- c(1,2,3)
b <- a
b[1] <- 99
print(a)
print(b)
c <- b
c[2] <- 88
print(c)
print(a)
print(b)
```

Output:

```
> a <- c(1,2,3)
> b <- a
> b[1] <- 99
> print(a)
[1] 1 2 3
> print(b)
[1] 99  2  3
> c <- b
> c[2] <- 88
> print(c)
[1] 99 88  3
> print(a)
[1] 1 2 3
>
```

3)

## Question 3

Analyze how R handles function arguments without pointers.

R Program:

```
modify <- function(x){
 x <- x + 10
 return(x)
}
v <- 5
modify(v)
print(v)
v <- modify(v)
print(v)
print(v + 1)
```

Output:

```
> modify <- function(x){
+     x <- x + 10
+     return(x)
+ }
> v <- 5
> modify(v)
[1] 15
> print(v)
[1] 5
> v <- modify(v)
> print(v)
[1] 15
> print(v + 1)
[1] 16
>
```

4)

**Question 4**

Analyze the following recursive function and identify the output.

R Program:

```
fact <- function(n){
 if(n == 1)
  return(1)
 else
  return(n * fact(n-1))
}
result <- fact(4)
print(result)
print(result/2)
```

Output:

```
> fact <- function(n){
+     if(n == 1)
+         return(1)
+     else
+         return(n * fact(n-1))
+ }
> result <- fact(4)
> print(result)
[1] 24
> print(result/2)
[1] 12
>
```

5)

## Question 5

Analyze how recursion terminates in R.

R Program:

```
sumN <- function(n){
 if(n == 0)
  return(0)
 else
  return(n + sumN(n-1))
}
s <- sumN(5)
print(s)
print(s - 5)
```

Output:

```
> sumN <- function(n){
+     if(n == 0)
+         return(0)
+     else
+         return(n + sumN(n-1))
+ }
> s <- sumN(5)
> print(s)
[1] 15
> print(s - 5)
[1] 10
>
```

6)

## Question 6

Analyze stack behavior in recursion.

R Program:

```
countDown <- function(n){
 if(n > 0){
  print(n)
  countDown(n-1)
 }
}
countDown(3)
print('Done')
```

Output:

```
> countDown <- function(n){
+     if(n > 0){
+         print(n)
+         countDown(n-1)
+     }
+ }
> countDown(3)
[1] 3
[1] 2
[1] 1
> print('Done')
[1] "Done"
```

7)

## Question 7

Analyze recursion versus iteration output.

R Program:

```
f <- function(n){
 if(n <= 1) return(1)
 return(f(n-1) + f(n-2))
}
x <- f(5)
print(x)
print(x - 1)
```

Output:

```
> f <- function(n){
+     if(n <= 1) return(1)
+     return(f(n-1) + f(n-2))
+ }
> x <- f(5)
> print(x)
[1] 8
> print(x - 1)
[1] 7
>
```

8)

## Question 8

Analyze memory behavior in recursive calls.

R Program:

```
test <- function(n){
 if(n == 1) return(1)
 return(test(n-1))
}
x <- test(3)
print(x)
print(x + 2)
```

Output:

```
> test <- function(n){
+     if(n == 1) return(1)
+     return(test(n-1))
+ }
> x <- test(3)
> print(x)
[1] 1
> print(x + 2)
[1] 3
> 
```

9)

## Question 9

Analyze why recursion needs a base condition.

R Program:

```
fun <- function(n){
 if(n < 0) return(0)
 return(fun(n-1))
}
x <- fun(2)
print(x)
print(is.numeric(x))
```

Output:

```
> fun <- function(n){
+     if(n < 0) return(0)
+     return(fun(n-1))
+ }
> x <- fun(2)
> print(x)
[1] 0
> print(is.numeric(x))
[1] TRUE
> 
```

10)

## Question 10

Analyze the combined concept of value passing and recursion.

R Program:

```
inc <- function(x){
 if(x == 0) return(0)
 x <- x + 1
 return(inc(x-1))
}
v <- 2
r <- inc(v)
print(v)
print(r)
```

Output:

```
> inc <- function(x){
+     if(x == 0) return(0)
+     x <- x + 1
+     return(inc(x-1))
+ }
> v <- 2
> r <- inc(v)
Error in `<current-expression>` : node stack overflow
Error during wrapup: node stack overflow
Error: no more error handlers available (recursive errors?); invoking 'abort' restart
> print(v)
[1] 2
> print(r)
Error: object 'r' not found
```

11)

## R Programming Structures

Q1. Analyze the following R program:

```
---------------------------------
result <- 0
for(i in 1:5){
 if(i %% 2 == 0){
  result <- result + i
 } else {
  result <- result - i
 }
}
print(result)
```

Output:

```
> result <- 0
> for(i in 1:5){
+     if(i %% 2 == 0){
+         result <- result + i
+     } else {
+         result <- result - i
+     }
+ }
> print(result)
[1] -3
>
```