Experiment No: 08

AIM: Using R tools to debugging a code while using trace back (),debug (),browser(),trace(), recover()

Objectives:

- ➔ Students will be able to implement a Program Using R tools to debugging a code
- ➔ Students will learn trace back (),debug(),browser(),trace(), recover() functions in R

Software Requirements: RStudio, VSCode

Theory:

Debugging code is an essential skill in programming, especially when working with complex algorithms or large codebases. R provides several tools that can aid in the debugging process, such as trace back(), debug(), browser(), trace(), and recover(). These tools offer different functionalities to help programmers identify and resolve issues within their code effectively.

The trace back() function in R provides a traceback of the function calls leading to an error, allowing programmers to pinpoint where the problem occurred in the code execution flow. Debug() is another useful tool that enables interactive debugging by pausing code execution at specified breakpoints, allowing programmers to inspect variables and step through code line by line using the browser() function. Additionally, trace() can be used to insert debugging statements into specific functions or expressions, providing insight into their behavior during runtime. Finally, the recover() function allows programmers to enter a debugging environment when an error occurs, providing an opportunity to inspect variables and the call stack to diagnose and fix issues efficiently. Overall, leveraging these debugging tools in R can significantly enhance the development process by streamlining the identification and resolution of code errors.

Implementation (Code and Output):

```r
# Define a function and make errors intentionally
divide_by_zero <- function(x, y) {
  z <- x / y
  return(z)
}

# Set up debugging using trace back() and debug()
trace_back_error <- function() {
  result <- divide_by_zero(10, 0)
  return(result)
}

# Call the function that triggers an error
trace_back_error()

# Use browser() and trace() for interactive debugging
trace_example <- function(x) {
  y <- x + 5
  browser()
  z <- y * 2
  return(z)
}

# Call the function with trace() enabled
trace_example(10)

# Use recover() to enter a debugging environment upon error
recover_example <- function() {
  a <- c(1, 2, 3)
  b <- a[5]  # Trigger an error by accessing out-of-bounds index
  return(b)
}

# Call the function that triggers an error and enter the debugging environment
recover_example()
```

```
Browse[1]> divide_by_zero
function(x, y) {
   z <- x / y
   return(z)
}
<bytecode: 0x0000021c7852f338>
Browse[1]> divide_by_zero(2,5)
[1] 0.4
Browse[1]> divide_by_zero(10,0)
[1] Inf
Browse[1]>
```

Conclusion: using trace back(), debug(), browser(), trace(), and recover() functions in R showcased the utility of these debugging tools in identifying and resolving code issues effectively. The trace back() function provided a clear traceback of function calls leading to errors, aiding in pinpointing the source of issues. Debug() and browser() facilitated interactive debugging by pausing code execution at breakpoints, allowing programmers to inspect variables and step through code line by line. Additionally, trace() enabled the insertion of debugging statements into specific functions or expressions for runtime analysis.