Portfolio 7

Sam Bowyer

Performance and Bugs

In this portfolio we will discuss some general techniques for improving and analysing `R` code within RStudio, however, these techniques will be widely applicable to other programming languages and IDEs.

Debugging

Bugs are inevitable during development of non-trivial code. Luckily IDEs usually come with tools that can help speed up the identification and correction of bugs.

Errors in syntax are often picked up by a spell-check or similar mechanism in the editor, and if not, you will hopefully receive an informative error message when trying to run your program. Logical errors, however, cannot usually be detected automatically, meaning that you will have to follow the logic of your program in order to detect the problem yourself (ideally you can produce a 'minimal working example' to begin zeroing in on the part of the code that actually produces the bug). You can of course do this by inserting lots of 'print' statements, allowing you to see the state of variables at different points during runtime, and often this will be sufficient to fix your problem, though a more detailed runtime analysis can be achieved through the IDE's built-in *debugger*.

In RStudio, given the following buggy code we can first examine the error message and then also see the call stack that led to the error by typing `traceback()`.

```
f1 <- function(arr1, arr2, i){
  if (arr1[i] == arr2[i]){
    cat("Match at index", i, "\n")
  }
}

f2 <- function(arr1, arr2){
  for(i in 1:4){
    f1(arr1, arr2, i)
  }
}
f2(1:4, c(1,3,2))
```

```
## Match at index 1

## Error in if (arr1[i] == arr2[i]) {: missing value where TRUE/FALSE needed

traceback()  # This would be typed in the terminal afterwards to produce the following output:

6: f1(arr1, arr2, i) at .active-rstudio-document#9
5: f2(1:4, c(1, 3, 2)) at .active-rstudio-document#12
4: eval(ei, envir)
3: eval(ei, envir)
2: withVisible(eval(ei, envir))
1: source("~/.active-rstudio-document")
```

In this case we see that the error occurs inside the function "f1", to which we can add a print statement to check the value of i at which the error occurs:

```
f1 <- function(arr1, arr2, i){
  print(i)
  if (arr1[i] == arr2[i]){
    cat("Match at index", i, "\n")
  }
}

f2(1:4, c(1,3,2))
```

```
## [1] 1
## Match at index 1
## [1] 2
## [1] 3
## [1] 4
```

```
## Error in if (arr1[i] == arr2[i]) {: missing value where TRUE/FALSE needed
```

Now that we know the error occurs when `i` is 4, we can run `browser()` to open a debugger at this point, which will allow us to move through the program step by step and investigate the value of any variable at each step:

```
f1 <- function(arr1, arr2, i){
  if(i==4){browser()}

  if (arr1[i] == arr2[i]){
    cat("Match at index", i, "\n")
  }
}

f2(1:4, c(1,3,2))
```

```
## Match at index 1
## Called from: f1(arr1, arr2, i)
## debug at <text>#4: if (arr1[i] == arr2[i]) {
##     cat("Match at index", i, "\n")
## }
```

```
## Error in if (arr1[i] == arr2[i]) {: missing value where TRUE/FALSE needed
```

The usage of the debugger can be seen using the `help` command:

Profiling

Performance