This time the simple useless function will generate a long list of IF clauses that will in fact return the resul of a mathematical operation between two numbers.

407 lines (404 sloc) 22.4 KB

```
r <- "calc <- function(a,b,oper){ if(a==1 & b==1 & oper==\"+\"){print(\"Result is 2\")}
 if(a==2 & b==1 & oper==\"+\"){print(\"Result is 3\")}
 if(a==3 & b==1 & oper==\"+\"){print(\"Result is 4\")}
 if(a==4 & b==1 & oper==\"+\"){print(\"Result is 5\")}
 if(a==5 & b==1 & oper==\"+\"){print(\"Result is 6\")}
 if(a==6 & b==1 & oper==\"+\"){print(\"Result is 7\")}
 if(a==7 & b==1 & oper==\"+\"){print(\"Result is 8\")}
 if(a==8 & b==1 & oper==\"+\"){print(\"Result is 9\")}
 if(a==9 & b==1 & oper==\"+\"){print(\"Result is 10\")}
 if(a==10 & b==1 & oper==\"+\"){print(\"Result is 11\")}
 if(a==1 & b==2 & oper==\"+\"){print(\"Result is 3\")}
 if(a==2 & b==2 & oper==\"+\"){print(\"Result is 4\")}
 if(a==3 & b==2 & oper==\"+\"){print(\"Result is 5\")}
 if(a==4 & b==2 & oper==\"+\"){print(\"Result is 6\")}
 if(a==5 & b==2 & oper==\"+\"){print(\"Result is 7\")}
 if(a==6 & b==2 & oper==\"+\"){print(\"Result is 8\")}
 if(a==7 & b==2 & oper==\"+\"){print(\"Result is 9\")}
 if(a==8 & b==2 & oper==\"+\"){print(\"Result is 10\")}
 if(a==9 & b==2 & oper==\"+\"){print(\"Result is 11\")}
 if(a==10 & b==2 & oper==\"+\"){print(\"Result is 12\")}
 if(a==1 & b==3 & oper==\"+\"){print(\"Result is 4\")}
```

This is not by no means any type of automatic generation of code, not a script, that will write itself.

The concept is fairly simple. Create a function that will return the results of a mathematical operation: addition, multiplication, division, subtraction, for two positive integers (whole number). Between 1 and 10. That's it 🙂

```
# Basic concept
calc <- function(a,b,oper){
  if(a==1 & b==1 & oper=="+"){print("Result is 2")}
  if(a==1 & b==1 & oper=="-"){print("Result is 0")}
  if(a==1 & b==1 & oper=="*"){print("Result is 1")}
  if(a==1 & b==1 & oper=="/"){print("Result is 1")}
  }

calc(1,1,"-")
calc(1,1,"+")
calc(1,1,"*")
calc(1,1,"/")
```

I have played a little bit with combinations of 4 operations, and two times 10 numbers, yielding 400 combinations. And the function that generates the final calculate function is the following:

```
# set all combinations
df <- data.frame(merge(merge(c(1:10), c(1:10), by=NULL), c("+","-","/","*"),
by=NULL))
colnames(df) <- c("numberA", "numberB", "oper")
f <- "calc <- function(a,b,oper){"
for (i in 1:nrow(df)){
```

```
  res <- paste0(as.character(df$numberA[i]) , df$oper[i],
as.character(df$numberB[i]))
  rr <- eval(parse(text=res))
  f1 <- paste0(' if(a==',as.character(df$numberA[i]), ' & b==',
as.character(df$numberB[i]), ' & oper==', '"',as.character(df$oper[i]),'"' ,
               '){print("Result is ', as.character(rr),'")}', '\n\r' ,
collapse=NULL)
  f <<- paste0(f, f1, collapse = NULL)
  if(i==nrow(df)){
    f <<- paste0(f, "}", collapse = NULL)
    eval(parse(text=f))
    }
}

calc(4,5,"/")
```

Once the calc function for all 400 combinations is created, feel free to use it.

For some comparison, let's crunch some numbers. I wanted to see how long does it take if the function get's longer. The code for measuring the execution time:

```
start.time <- Sys.time()
calc(20,3,"+")
end.time <- Sys.time()
end.time - start.time
```

And the results for adding more positive integers is at least to say interesting, if not useless.

Number of integers	Number of combinations	First Execution time (sec)	Average next execution time (sec)	Size of function
10	400	5,6s	0.0012	2.3Mb
20	1600	11,5s	0.0022	9.3Mb
25	2500	28,4s	0.0027	14.5Mb
50	10000	717s	0.0081	58Mb
100	40000	/	/	232.1Mb

Hitting first 100 positive integers, the function increased to 230 Mb, taking space in memory. And based on the exponential growth of the size of the function, my laptop would hit "out of memory" boundary by th integer 1000. This makes you appreciate he usefulness of CPU architecture 🙂