



## II. R 프로그래밍



- 조건문, 반복문
- 함수정의 및 활용
- 사용자의 입력 받기 및 메뉴 생성



# 흐름제어(조건문)

## ➤ if

문법	예
<pre>if (cond) {   cond이 참 일때 실행할 문장 } else {   cond이 거짓일 때 실행 할 문장 }</pre>	<pre>&gt; if (x&gt;=5) { + print("greater than 5") + }else{ + print("less than 5") + }</pre>

## ➤ ifelse

문법	예
<pre>ifelse (   test, # 참,거짓을 저장한 객체   yes,  # test가 참일때 선택할 값   no    # test가 거짓일때 선택할 값 )</pre>	<pre>&gt; x=c(1,2,3,4,5) &gt; ifelse(x%%2 ==0,"even","odd") [1] "odd" "even" "odd" "even" "odd"</pre>



# 흐름제어(조건문)

➤ switch: 다중 조건문

switch(기준, 수행문)  
수행문:  
값 = 수행문

```
1 x=c(1:10)
2 x
3
4 switch(x[2],|
5     "1"=print("one"),
6     "2"=print("two"),
7     "3"=print("three"),
8     print("something else"))
9
```

4:13 (Top Level) ⇅

**Console** **Terminal** x

C:/Users/KSL/Desktop/course/R/Prog/chap1/ ➡

```
> x=c(1:10)
> x
[1] 1 2 3 4 5 6 7 8 9 10
> switch(x[2],
+     "1"=print("one"),
+     "2"=print("two"),
+     "3"=print("three"),
+     print("something else"))
[1] "two"
> |
```



# 흐름제어(반복)

➤ for, while, repeat

문법	예
<pre>for (i in data) {   i를 사용한 문장 }</pre>	data에 들어있는 각각의 값을 변수 i에 할당하면서 각각에 대해 블록 안의 문장을 수행한다.
<pre>while (cond) {   조건이 참 일때 수행할 문장 }</pre>	조건 cond가 참일 때 블록 안의 문장을 수행한다.
<pre>repeat {   반복해서 수행할 문장 }</pre>	블록 안의 문장을 반복해서 수행한다.

# 흐름제어(반복)



➤ for

```
> for (i in 1:10){ print(i)}
```

A screenshot of an R Console window. The window has a title bar that says "R Console" and standard Windows window controls (minimize, maximize, close). The console contains several lines of R code and their output. The code includes comments in Korean explaining the purpose of each step, such as setting a variable, using a for loop to calculate a sum, and confirming the result. The output shows the sum of 1 to 10 is 55, and the sum of 1 to 5 squared is 225.

```
R Console
> # R에서 지원하는 반복 기능에 대해서 알아보자
>
> # 반복 기능 요약
>
> # for (반복) 명령문
> # while (조건) 명령문
> # repeat 명령문
>
> #####
> # for문의 사용 예
>
> # for문의 기본적인 사용 예(1)
>
> sum1 <- 0                # 변수를 설정한다
> for( i in seq(1, 10, by=1)) sum1 <- sum1+i # 1,2,3..10을 차례대로 넣고, 이값을 더한다
> sum1                     # 총합을 확인한다
[1] 55
>
> # for문의 기본적인 사용 예(2)
>
> sum1 <- 0                # 변수를 설정한다
> for( i in 1:5) {         # i 는 1, 2, 3, 4, 5를 차례로 대입한다
+   for ( j in 1:5)        # j 는 1, 2, 3, 4, 5를 차례로 대입한다
+     sum1 <-sum1+i*j      # 1*1,1*2, 1*3,1*4,1*5,2*1,2*2 .... 5*5의 값을 모두 더한다
+ }
>
> sum1                     # 모두 더한 값을 확인한다
[1] 225
>
> #####
```

# 흐름제어(반복)



➤ while

```
> i = 1
> while (i<=10) {
+ print(i)
+ i = i+1
+ }
```

```
> sum2 <- 0          # 변수를 설정한다
> i <-1
>
> while ( i <=5 ) {   # i 는 1, 2, 3, 4, 5를 차례로 대입한다
+   j <- 1;
+   while(j <= 5 ) {  # j 는 1, 2, 3, 4, 5를 차례로 대입한다
+     sum2 <-sum2+ i*j; # 1*1,1*2, 1*3,1*4,1*5,2*1,2*2 .... 5*5의 값을 모두 더한다
+     j <- j+1;
+   }
+   i <- i+1;
+ }
>
> sum2               # 모두 더한 값을 확인한다
[1] 225
```



# 흐름제어(사용자 정의 제어)

## ➤ break, next

문법	예
break	반복문을 종료
next	현재 수행 중인 반복문 블록의 수행을 중단하고 다음 반복을 시작

```
> i=1
> repeat{
+ print(i)
+ i=i+1
+ if(i>10)break
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```





# 흐름제어(반복)

## ➤ Repeat + break

```
R Console
> #####
> # repeat 문의 사용 예
>
> # repeat 문의 기본적인 사용 예(1)
>
> sum3 <- 0          # 변수를 설정한다
> i <- 1
>
> repeat {           # 1,2,3... 10을 더한다
+   sum3 <- sum3+i;
+   i <- i+1;
+   if(i>10) break   # i가 11이면 루프를 탈출한다
+ }
>
> sum3               # 결과를 확인한다
[1] 55
> i                  # i의 값이 11임을 확인한다
[1] 11
>
> # repeat 문의 기본적인 사용 예(2)
>
> sum3 <- 0          # 변수를 설정한다
> i <- 1
>
> repeat {
+   if(i > 5) break;  # i는 1,2,3,4,5이다
+   j <- i;
+   repeat {
+     if(j > 5) break; # j는 1,2,3,4,5이다
+     sum3 <- sum3+i*j; # 1*1, 1*2, 1*3, 1*4, 1*5, 2*2, 2*3, 2*4, 2*5, 3*3, 3*4, 3*5, 4*4, 4*5, 5*5
+     j <- j+1
+   }
+   i <- i+1;
+ }
>
> sum3               # 결과를 확인한다
[1] 140
>
>
```

# 실습



➤ 1부터 30까지 숫자 중에서 짝수만 출력하시오.

```
> i=0
> while ( i<=30) {
+ i=i+1
+ if(i%%2 == 0) print(i)
+ }
[1] 2
[1] 4
[1] 6
[1] 8
[1] 10
[1] 12
[1] 14
[1] 16
[1] 18
[1] 20
[1] 22
[1] 24
[1] 26
[1] 28
[1] 30
```

```
> i=0
> while(i<=30) {
+ i=i+1
+ if(i%%2 != 0) next
+ print(i)
+ }
[1] 2
[1] 4
[1] 6
[1] 8
[1] 10
[1] 12
[1] 14
[1] 16
[1] 18
[1] 20
[1] 22
[1] 24
[1] 26
[1] 28
[1] 30
```



# 함수의 정의

## ➤ 기본 정의

### 문법

```
function_name = function(인자, 인자, ... ) {  
  함수 본문  
  return(반환 값) #반환값이 없는 경우 생략함  
}
```

## ➤ 예: 피보나치 수열, $f(n) = f(n-1) + f(n-2)$ ( $n > 2$ )

```
> fibo = function(n) {  
+   if (n==1 || n==2) {  
+     return(1)  
+   }  
+   return(fibo(n-1)+fibo(n-2))  
+ }
```

← 함수 정의

```
> fibo(1)  
[1] 1  
> fibo(2)  
[1] 1  
> fibo(3)  
[1] 2  
> fibo(4)  
[1] 3
```

← 함수 호출



## 함수: 예

➤ calculator(피연산자, 연산자)

```
10 calculator <- function(x, type) {  
11     switch(type,  
12         mean = mean(x),  
13         sum = sum(x),  
14         print("unexpected type"))  
15 }  
16 x <- c(1:10)  
17 calculator(x, "mean")  
18 calculator(x, "sum")  
19 calculator(x, "sd")  
20
```

20:1 (Top Level) ↕

Console

Terminal x

C:/Users/KSL/Desktop/course/R/Prog/chap1/ ➡

```
> calculator <- function(x, type) {  
+   switch(type,  
+       mean = mean(x),  
+       sum = sum(x),  
+       print("unexpected type"))  
+ }  
> x <- c(1:10)  
> calculator(x, "mean")  
[1] 5.5  
> calculator(x, "sum")  
[1] 55  
> calculator(x, "sd")  
[1] "unexpected type"
```



# 함수 정의 및 사용

## ➤ 인자 전달 방식: 위치, 이름

```
> f = function(x,y) { #두개의 인자를 가짐
+ print(x)
+ print(y)
+ }
>
> f(1,2) #위치에 맞추어 인자 전달
[1] 1
[1] 2
>
> f(y=2, x=1) #이름을 지정하여 인자 전달
[1] 1
[1] 2
```

## ➤ 가변 길이 인자: "..."

```
> f = function(...) {
+ args = list(...)
+ for (a in args) {
+ print(a)
+ }
+ }
>
> f('3','4')
[1] "3"
[1] "4"
> f('1','2','3')
[1] "1"
[1] "2"
[1] "3"
```



# 중첩함수

---

- 함수 안에 또 다른 함수(중첩함수)를 정의하여 사용할 수 있음
- 함수 안에서 반복되는 동작을 한 함수로 만들고 이를 호출하여 코드를 간결하게 표현할 수 있음

➤ 예

```
> f = function(x,y) {  
+   print(x)  
+   g = function(y) {  
+     print(y)  
+   }  
+   g(y)  
+ }  
>  
> f(1,2)  
[1] 1  
[1] 2
```



# 함수의 영역 규칙(scope)

- 정적 영역 규칙 사용: 변수가 정의된 블록 내부에서만 접근 가능

```
> n=1
> f = function() {
+   print(n)
+ }
>
> f()
[1] 1
```

```
> n = 100
> f = function() {
+   n = 1
+   print(n)
+ }
> f()
[1] 1
```

```
> f=function(){
+   a=1
+   g = function() {
+     a = 2
+     print(a)
+   }
+   g()
+   print(a)
+ }
>
> f()
[1] 2
[1] 1
```

```
> b=0
> f=function() {
+   a=1
+   g = function() {
+     a<-2
+     b<-2
+     print(a)
+     print(b)
+   }
+   g()
+   print(a)
+   print(b)
+ }
>
> f()
[1] 2
[1] 2
[1] 2
[1] 2
```



# 함수 저장 및 활용



➤ Save: 저장

➤ Load: 불러오기

```
R Console
> # 프로그램을 제작하다 보면, 특정 기능을 함수로 만들어서 파일로 저장하다가
> # 실행 중에 읽어들이어서 사용해야 하는 경우가 생긴다.
> myF <- function(x){      # 함수를 선언한다
+   return (x*x)
+ }
> myF(2)                    # 함수를 사용한다
[1] 4
> save(myF, file="myF.Rdata") # 설정된 디렉토리로 함수를 저장한다(이름:myF$)
> ls()                      # myF이 변수로 선언되어 있음을 확인한다
[1] "myF"
> rm("myF")                  # myF이 제거되었음을 확인한다
> myF(3)                      # 당연히 에러 발생
Error: could not find function "myF"
> load("myF.Rdata")          # 함수를 읽어 들여서 사용할 수 있게 한다
> myF(4)                      # 읽어들이인 함수를 사용한다
[1] 16
> ls()                        # myF 함수가 있음을 확인한다.
[1] "myF"
> # 여러개의 함수를 하나의 파일에 저장해서, 실행 중에 불러올 수 있다
> rm("myF")
> myF <- function(x) { return (x*x) } # 함수 선언
> myF2 <- function(x) { return (x) }  # 함수 선언
> save(myF, myF2, file="myFF.Rdata") # 여러 함수를 하나의 파일에 저장
> rm("myF", "myF2")           # 선언된 함수를 지운다
> ls()
character(0)
> load("myFF.Rdata")          # 하나의 파일에서 여러개의 함수를 동시에 읽는다
> ls()                         # 함수가 시스템에 올라온 것을 확인한다
[1] "myF" "myF2"
>
```





## 참고: R 프로그래밍 연산자

연산자	기능	연산자	기능
-	뺄셈	+	덧셈
!	부정	~	=
?	도움말	:	공차 1의 등차수열
*	곱셈	/	나눗셈
^	거듭제곱	%%	나머지
/%	정수나눗셈	%%*	행렬곱
%0%	외적	%x%	크로네커곱
==	같다	!=	같지않다
>=	크거나 같다(이상)	>	크다
<=	작거나 같다(이하)	<	작다(미만)
&, &&	그리고	,	또는
<-	대입	<<-	영속 대입

## 참고: R 프로그래밍 인덱스

---



기호	기능
<code>x[i]</code>	i 번째 요소에 접근한다
<code>x[(i)]</code>	i 번째 리스트의 요소에 접근한다
<code>x\$a</code>	x에서 a를 추출한다
<code>x[i, j]</code>	i 행 j 열의 요소에 접근한다
<code>x\$"a"</code>	x에서 a를 추출한다

# 실습: Queue 자료 구조 만들기



Queue 자료 구조를 사용하려고 한다.

다음과 같은 Queue 자료 구조 조작을 위한 함수를 생성하시오.

- enqueue: queue의 맨 뒤에 데이터를 추가한다
- dequeue: queue의 맨 앞에 있는 데이터를 가져온다. 가져온 데이터는 줄에서 빠진다.
- size: queue의 길이, 즉 자료 구조 내에 저장된 데이터의 수를 반환한다.

```
q = c()
q_size = 0
enqueue = function(data) {
:
:
}
dequeue = function() {
:
:}
size = function() {
:
}
```

2	3
---	---

```
> enqueue(1)
> enqueue(2)
> enqueue(3)
> print(size())
[1] 3
> print(dequeue())
[1] 1
> print(size())
[1] 2
```



# Improvement

Problem: 모듈화 되어 있지 않기 때문에 외부에서 변경 가능

```
1 q=c()  
2 q_size=0  
3  
4 enqueue = function(data) {  
5   q <- c(q,data)  
6   q_size <- q_size+1  
7 }  
8  
9 dequeue = function() {  
10  first=q[1]  
11  q <- q[-1]  
12  q_size <- q_size -1  
13  return(first)  
14 }  
15  
16 size = function() {  
17   return(q_size)  
18 }  
19
```

```
1 queue = function() {  
2   q=c()  
3   q_size=0  
4  
5   enqueue = function(data) {  
6     q <- c(q,data)  
7     q_size <- q_size+1  
8   }  
9  
10  dequeue = function() {  
11    first=q[1]  
12    q <- q[-1]  
13    q_size <- q_size -1  
14    return(first)  
15  }  
16  
17  size = function() {  
18    return(q_size)  
19  }  
20  
21  return(list(enqueue=enqueue, dequeue=dequeue, size=size))  
22 }
```

q\_size = 100 영향 받음

```
> q=queue()  
> q$enqueue(1)  
> q$enqueue(2)  
> q$size()  
[1] 2  
> q$dequeue()  
[1] 1  
> q$dequeue()  
[1] 2  
> q$size()  
[1] 0
```

q\_size = 100 => 영향 없음



# 참고: 사용자 입력

## ➤ 한 문자를 입력 받는 경우

```
1 fun = function() {  
2   answer = readline("y/n을 입력하세요: ")  
3   if (substr(answer,1,1) == "n")  
4     cat("n을 입력받았습니다.")  
5   else  
6     cat("y를 입력받았습니다.")  
7 }
```

7:2 (Top Level) ↕

**Console** **Terminal** x

C:/Users/KSL/Desktop/course/R/Prog/chap1/ ↗

> fun()  
y/n을 입력하세요: y  
y를 입력받았습니다.

## ➤ 한 문장을 입력 받는 경우

```
9 fun2 = function() {  
10   x = readline("문장을 입력하세요: ")  
11   unlist(strsplit(x, " "))  
12 }
```

12:2 (Top Level) ↕

**Console** **Terminal** x

C:/Users/KSL/Desktop/course/R/Prog/chap1/ ↗

> fun2()  
문장을 입력하세요: I am Tom  
[1] "I" "am" "Tom"



## 참고: 메뉴 생성

```
1 #Menu
2
3 funMenu = function() {
4   answer = menu(c("오렌지", "포도", "딸기"))
5   if (answer == 1) {
6     cat("your input is 오렌지")
7   } else if (answer == 2) {
8     cat("your input is 포도")
9   } else {
10    cat("your input is 딸기")
11  }
12
13 }
```

13:2 (Top Level) ↕

Console

Terminal x

C:/Users/KSL/Desktop/course/R/Prog/chap1/ ↗

> funMenu()

1: 오렌지

2: 포도

3: 딸기

선택: 1

your input is 오렌지

> |



## 참고: 조건에 맞는 데이터의 위치 찾기

- 자신이 원하는 데이터가 vector, matrix, data frame안에 어디에 위치하고 있는지 알고 싶을 때 사용
- 함수

함수	기능
which()	찾고자 하는 값의 index를 알아냄
which.max()	벡터 안에서 최대값의 index를 알아냄
which.min()	벡터 안에서 최소값의 index를 알아냄





# 실습: 조건에 맞는 데이터의 위치 찾기

```
# R Programming
```

```
# which
```

```
score = c(76, 84, 69, 50, 95, 60, 82, 71, 88, 84)
```

```
which(score==69)
```

```
which(score >= 85)
```

```
max(score)
```

```
which.max(score)
```

```
min(score)
```

```
which.min(score)
```

```
idx = which(score <=60) #성적이 60 이하인 학생의 위치
```

```
score[idx] = 60
```

```
idx = which(score >=80) #성적이 80 이상인 학생의 위치
```

```
score.high = score[idx]
```

```
score.high
```

The screenshot shows the RStudio environment. The script editor contains the following code:

```
1 # R Programming
2 # which
3
4 score = c(76, 84, 69, 50, 95, 60, 82, 71, 88, 84)
5 which(score==69)
6 which(score >= 85)
7 max(score)
8 which.max(score)
9 min(score)
10 which.min(score)
11
```

The console shows the execution of the code:

```
R 4.1.2 ~ /R/prog/
>
[1] 4
[1] 8 9
[1] 95
[1] 8
[1] 60
[1] 5
```



# 실습: 조건에 맞는 데이터의 위치 찾기

```
# R Programming
```

```
# which
```

```
score = c(76, 84, 69, 50, 95, 60, 82, 71, 88, 84)
```

```
which(score==69)
```

```
which(score>= 85)
```

```
max(score)
```

```
which.max(score)
```

```
min(score)
```

```
which.min(score)
```

```
#성적인 60점 미만인 경우 60점으로 상향 조정
```

```
idx = which(score <=60)
```

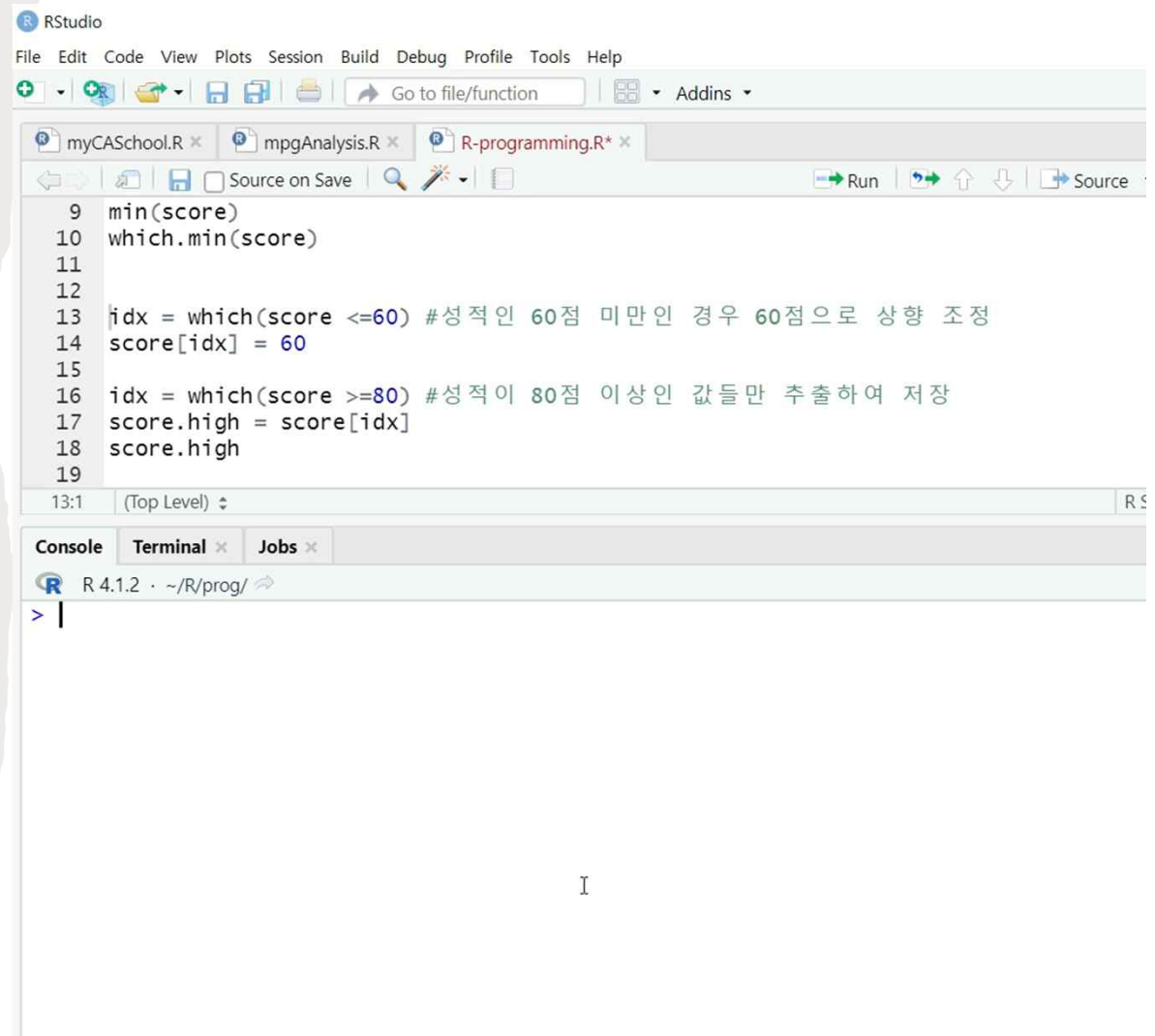
```
score[idx] = 60
```

```
#성적이 80점 이상인 값들만 추출하여 저장
```

```
idx = which(score >=80)
```

```
score.high = score[idx]
```

```
score.high
```



The screenshot shows the RStudio IDE with three open files: myCASchool.R, mpgAnalysis.R, and R-programming.R\*. The active file is R-programming.R\*, which contains the following R code:

```
9 min(score)
10 which.min(score)
11
12
13 idx = which(score <=60) #성적인 60점 미만인 경우 60점으로 상향 조정
14 score[idx] = 60
15
16 idx = which(score >=80) #성적이 80점 이상인 값들만 추출하여 저장
17 score.high = score[idx]
18 score.high
19
```

The console window at the bottom shows the R prompt > |, indicating that the code has been executed.



## 실습: R 함수 작성

(중간고사 기출)

vector형 변수를 입력 받아서, 해당 vector에 있는 결측치(na)를 vector에 속해 있는 원소들의 평균값으로 대체하는 함수를 작성하시오.

(예)

```
> x = c(7,12,9,15,NA,8,14,2,9,NA,8 )
> replacena(x)
[1] 7.000000 12.000000 9.000000 15.000000 9.333333 8.000000 14.000000 2.000000
[9] 9.000000 9.333333 8.000000
```





## 실습: R 함수 작성

```
replacena=function(x){  
  x=ifelse(is.na(x),mean(x,na.rm=T),x)  
  return(x)  
}
```

```
x=c(7,12,9,15,NA,8,14,2,9,NA,8)  
replacena(x)
```

# 실습: Fibonacci 수열 리스트 생성 함수



- 주어진 개수만큼 Fibonacci 수열을 list로 생성하는 함수(fibolist)를 작성하시오
- Fibonacci 수열: 1, 1, 2, 3, 5, 8, 13, ...

```
> fibo = function(n) {  
+ if (n==1 || n==2) {  
+   return(1)  
+ }  
+ return(fibo(n-1)+fibo(n-2))  
+ }  
> fibo(1)  
[1] 1  
> fibo(2)  
[1] 1  
> fibo(3)  
[1] 2  
> fibo(4)  
[1] 3  
.
```



```
> fibolist(2)  
Error in fibolist(2) : The size should be greater than 2  
> fibolist(5)  
[1] 1 1 2 3 5  
> fibolist(20)  
[1]      1      1      2      3      5      8     13     21     34     55     89    144    233    377    610  
[16]  987 1597 2584 4181 6765
```



## 실습: Tips



```
1 ▾ fibolist=function(size) {  
2 ▾   if(size <=2) {  
3     stop("The size should be greater than 2")  
4   }  
5   num1=1  
6   num2=1  
7   fibonacci=c(num1,num2)  
8   count=2  
9 ▾   repeat {  
10     count=count+1  
11     oldnum2=num2  
12     num2=num1+num2  
13     fibonacci=c(fibonacci,num2)  
14     num1=oldnum2  
15     if(count>=size) break  
16   }  
17   print(fibonacci)  
18 }
```

# Homework#5주차



stack 자료 구조를 구성하는 다음과 같은 함수를 작성하시오.

- push(data)
- pop()
- size()
- empty()
- full()

가정: Stack의 최대 크기는 20



# Homework#5주차



stack 자료 구조를 구성하는 다음과 같은 함수를 작성하시오.

- push(data)
- pop()
- size()
- empty()
- full()

가정: Stack의 최대 크기는 20

