R-Programming Structures:

R is a block structured language in the manner of the ALGOL descentant family such as C, C++, phython, Perl and so on.

Here the Basic structure of R as a programing language in which we review the more details on loops and functions and other control statements. mai

Many scripting languages we do not declare variables in R. R has a richer scoping structure

Control statements:

Control statements in R very similar to those of the ALGOL descent family.

Here we will look at loop and if-else. statements.

loops:-

Here we define the for loop function in that function the following line should have been instantly recognized by python programmers.

Syntax: for (Declared variable in Intilized variable)

for(n in x)

for any next it means that their will be one iteration of the loop. for each component of vector x with n taking on the varlles of those components.

In the 1st iteration n=x[1] in the 2nd iteration n=x[2] and so on

For example we use this structure to Output the Square of every element in the vector

Eg:- For:
```
> x ← c(5,6,7)
> for (n in x)
> print (n^2)
```
O/p: 25 36 49

Eg:-
```
> i ← 1
> while (i≤10)
> i=i+4
> Print (i)
```
O/p:- [1] 13

```
eg:- > a <- 1
     > while (a<10)
     > { print(a)
     > a = a+1
     }
```

o/p: [1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9

17/05/2022

```
eg:- > i <- 1
     > while (i<=10)
     > print(i)
     > i <- i+4
```

o/p :- 1 5 9

```
eg:- > i <- 1
     > while (TRUE)
     > i <- i+4
     > if (i>10)
     break.
     > i
```

o/p : [1] 13

[while(TRUE) = repeat]

```
eg:- > i <- 1
     > repeat
     > i <- i+4
     > if (i>10)
     break
     > i
```

o/p : [1] 13

In loopings with while and repeat is also available with this while and repeat functions are complete with break. A statement that causes control to leve loop.

→ In the 1st example the variable i took on the values 1, 5, 9 and 13 as the loop went through its iterations In that last case the condition $i \leq 10$ fail so, the break took hold and we left the loop

Note:- repeat has no boolean exist condition we must use break or something like return.

Next function:-
The Next statement in R programming language is useful when we want to skip the current iteration of a loop without ed termination it.

1Eg:- >i ← 1
>while (i<=10)
>if (i==5)
>next
>print (i)
>i=i+1

```
i←1
while (i<=10)
{
if (i==5)
{
next
}
print (i)
i=i+1
}
```

O/P: 1 2 3 4 6 7 8 9 10

2 Eg:- >x ← 1:10
>for (val in x)
{
if (val==3)
{
next
}
print (val)
}

O/P:- 1 2 4 5 6 7 8 9 10

3eg:- >v ← LETTERS [1:6]
for (i in v)
{
if (i=="D")
{
next
}
print (i)
}

O/P: A B C E F

Scan() function:

19/05/2022

Here we use scan() read the no.of items and displays the items. here we can use in any looping function
i.e for, if, while etc.:

File-1: 1
2
3
4
5
6

> for (fn in c ("File 1", "File 2"))
print (scan (fn))
Read 6 items
[1] 1 2 3 4 5 6

File-2:
5
12
13

Read 3 items
[1] 5 12 13

looping over non vector sets:

R does not directly support iteration over non vector sets But their are a capple of indirect & easy ways to ecomplish it

lapply:

The iterations of the loop are independent of each other, those thus allowing to be performed in any order

```
> file 1 ← c (1, 2, 3, 4, 5, 6)
> file 2 ← c (5, 12, 13)
> lapply (c ("file 1", "file 2"), sum)
```

o/p:- 6 14 16 4 5 6

get() function :-

As the name implise in R programing language is used to return an object with the name specified argument to the function. this is another method of print the values of the object i.e Just like a print function

this function can also be used to copy one object to another

Syntax: get ("object")

object are nothing but vector, array, list etc...

```
eg:- x1 ← c ("a", "b", "c")
     x2 ← c (1, 2, 3)
     get ("x2")    o/p:- 1 2 3
x4 ← get ("x1")
Print (x4)         o/p:- "a" "b" "c"
list 1 ← list (x1, x2)
get ("list 1")     o/p:
```

[1]	$x1	$x2
	a	1
	b	2
	c	3

If_else

Syntax: v ← if (condition)
exp 1
else
exp 2

Eg:- > x ← 2
> y ← if (x == 2)
> x o/p: 2
> else
> x+1
> y o/p: TRUE

The result of y depending on weather condition is TRUE or FALSE we can use this fact to complie our code with out taking x value

```
> y ← if (x == 2)
  y ← x
  else
  y ← x+1
```

```
if (x == 2)
y ← x
else
y ← x+1
```

here x is not mentioned so output gets error!

Arithmetic and Boolean operators and values: 20/5/2022

Basic R Operators

Operation	Description		
x+y	Addition	x%%y	Integer division
x-y	Subtraction	x==y	Test for equality
x*y	multiplication	x<=y	Test for less than or = to
x/y	division	x>=y	Test for greater than or equal to
x^y	Exponentiation	x&&y	Boolean AND for scalars
x%%y	modular arithmetic	x||y	Boolean OR for Scalars
		x&y	Boolean AND for Scalars Vectors (vector x,y,result)
		x|y	Boolean OR for vectors (vector x,y result)
		!x	Boolean negation

Example:-

```
> x
[1] TRUE  FALSE  TRUE
> y
[1] TRUE  TRUE  FALSE
```

```
> x+y
[1] TRUE FALSE FALSE
> x[1] && y[1]
[1] TRUE
> x&&y # looks at just the first elements
       of each vector.
[1] TRUE
> if(x[1] && y[1]) print ("both TRUE")
[1] "both TRUE"
> if (x&&y) print ("both TRUE")
[1] "both TRUE"
```

warning massage:

In if (x&y) print ("both TRUE")

the condition has length > 1 and only the first element will be used.

In R programming has no scalar types with scalars being treated as one element vectors

In the above table there are different boolean operators for the scalar and vector cases.

In the above example the central point is that in evaluating an if, we need a single boolean not a vector of booleans, Hence the warning seen in the proceeding example as well as the need for having both the & and && operators

⇒ The boolean values TRUE and FALSE can be abbreviated as T and F these values changes to one and zero in arithemetic expression

Eg:

```
> 1<2
TRUE
> (1<2) * (3<4)
[1] 1
> (1<2) * (3<4) * (5<1)
[1] 0
```

> (1<2) == TRUE'

> (1<2) == 1

In the 2nd compilation (statement) for instance the comparision (1<2) returns TRUE and (3<4) yields TRUE as well both values are treated as 1 so the product is 1

R functions look similar to those of C, JAVA, python and so on. However they have much more of a functional Programming flavours which has directly implications for the R programmers

Default values for Arguments 23/05/2022

In R programming function can have default values of its parameters it means while defining a function, its parameters can be set values. It makes function more generic

```
Eg1:- function_name <- function (a, b=10)
      {
        Print (a+b)
      }
      function_name(10)  o/p: 20
      function_name (30, 40)  o/p: 70
```

the consle prints on 70 because it over olds the 20.

```
Eg2: > n <- function (a, b, c="Hai", d="TKR")
     {
       Print (10,20)
     }
     n (a, b, c, d)
```

o/p:- 10 20 "Hai" "TKR"

The terminology named arguments that because R uses lazy evolution. It does not evolute an expression until or unless it needs. to the named argument may not actually be used

Return values:-

The Return values of a function can be any R object. the return value is often a list, it could even be another functio we can transmit a value back to the caller by explicitly calling return. The value of the last executed statement will be return by default.

Eg:- Return value

```
Odd count
> function (x)
 {
> k ← 0
>for (n in x)
  {
 > if (n%.%.2 ==1)
 > k ← k+1
 }
> return (k) (or) > k
  }
```

this functions returns the count of odd numbers in the argument.

- we could simplify the code by eliminating the call to return() to do this we evalute the expression to be returned k in the last statement.

Eg2 :- odd count

```
> function (x)
{
> k ← 0
> for (n in x) {
> if (n%.%2 ==1)
> k ← k+1
 }
> k
 }
```

Eg3 :- odd count

```
> function (x)
 {
> k ← 0
> for (n%.%.2 ==1) {
> k ← k+1 > if (
 }
> for (n in x) {
> if (n%.%.2 ==1) {
> k ← k+1)
 }
}
```

In the eg3 the last executed statement here is the Call to for() which returns the value NULL that means there would be no return value at all.

24/05/2022

Returning Complex Objects (8.4)

The return objects can value can be any R object we can return complex objects

Eg:
```
g
function()
{
t ← function (x)
return (x^2)
return (t)
}
g()
function (x)
return (x^2)
```

In the above example if function have multiple return values place them in a list or other container.

Functions are objects:

R functions are first class objects that means they can be used for the most part just like other objects

In R a function is an object so the R interpreter is able to pass control to the function along with arguments that may be necessary for the function to acomplish the action.

Syntax of function creation:

```
q ← function (x)
  {
   return (x+1)
  }
```

Function is a built in R function whose job is to create functions

- Their are two arguments to function the first is the formal argument list for the function we are creating i.e x the
- Second is the Body of the function i.e the single statement return (x+1) the 2nd argument must be of class expression. so, the point is that the right hand side creates a function object i.e is "g"

```
Eg :=> f1 ← function (a,b)
        return (a+b)
     > f2 ← function (a,b)
        return (a-b)
     > f ← f1
     > f(3,2)
     [1] 5
```

```
>f ← f2
 f(3,2)
 [1] 1
```

(Nested function (or) function within a function)

```
>g ← function (h,a,b)
   h(a,b)
>g (f1, 3,2)
   [1] 5
>g (f2, 3,2)
   [1] 1
```

In the above example functions are objects we can also assign them use them as arguments to other functions i.e h this process we can call it as a function call. Nested within another function or function call with in a function.

28/05/2022

Environment scope issues:

Environment can be thought of a collection of objects "functions, variables". An environment is create we when we first sign up the R ~~Interpreter~~ studio that means Rstudio. Any variable we define is now in this environment A function formaly refered to as a closure in the R documentation consist not only of its arguments and body but also of its environment in this environments works in R is essential for writing effective R functions

Eg:-

```
a ← 2
b ← 5
f ← function(x)
 x ← 0
 w ← 12
 f ← function (y)
 {
  d ← 8
  h ← function ()
  {
   return (h())
  }
  Environment (f)
```

o/p: 0

o/p: R_GlobalEnv

```
a ← 2
b ← 5
f ← function
x ← 0
ls() → it gives output
       for how many
       variables are present
```

o/p: a b f

Here the function f is created to' at the top level i.e. at the interpreter command prompt and thus has the top level Environment. which in R Output is refered to as R_GlobalEnv. But which we refere to in R code as .GlobalEnv

In your R program if we run any program as a batch file. i.e is consider top level

```
ls():
    a←2
    b←5
f←function(x)
    n←0
    ls()
o/p: "a" "b" "f"
```

```
Eg2:-  w←12
f←function(y)
{
d←8
h←function()
{
return(ls())
}
Environment(f)
ls()
o/p:- "w" "d"
```

```
ls(str()
f: function(y)
w: num 12
```

In the above function ls() list the object of an environment. If we call it at the top level we get the top level environment.

The top level environment here includes the variable "w". which is actually used in "f" function

```
> environment()
<environment : R_GlobalEnv>
> Global Env
< environment: R.GlobalEnv>
```

In the above two commands we can used to know the R environment

Scope Issues :-

If we were working with the c language we would say that. the variable "w" in the previous example is global to function. while "d" is local to f. things are Smilar in R. but R is more hairarecal Compared to C

```
> outer_fun <- function()
{
  b <- 20   } gobal variable.
  inner_fun <- function()
  {
    c <- 30   } local variable
  }
  a <- 10   } gobal variables
}
```

scope 1, scope 2

so, here a, b are the gobal variables and c is local variable.

30/05/2022

we have h() being local to f() just liked d. In such a situation scope to be hierarchial in R setup d which is local to f() is in turn Global to h() The some is true for y as argument are consider locals in R. the hirarchel nature of scope implies that since w is global to f. it is gobal to h as well we do use a within h()

Eg:-
```
w <- 12
f <- function (y).
{
  d <- 8
  h <- function()
  {
    return (h())
  }
  return (h())
}
```

O/P:-
```
f(2)
h <- function
{
return (d* (w+y))
}
```

The R interpreter founds that there was no local variables of that name, so it ascended to the next higher level in this case the top level where it founds a variables 'w' with value 12. Keep in mind h() is local to f(). it invisible at the top level it displays error: Object 'h' not found

Eg:-
```
f ← function(y)
{
d ← 8
h ← function()
{
return (d * (w+y))
}
return (h())
}
```
o/p:- 112

```
> f ← function(y)
{
d ← 2
return (h())
}
> h ← function()
{
return (d * (w+y))
}
```
o/p:- error: d, w, y values are not there

In the above examples environments created by inheritence that means inner most environment is used first A reference to d which h() would refers to h()'s d not in f()'s compare the both example the example 2 does not work as d is no larger in the environment h because h() is defined at the top level thus an error is generated

Eg:-
```
f ← function(y)
{
d ← 8
return (h(d,y))
}
> h ← function (dee, yyy)
{
return (dee * (w+yyy))
}
> f(2)
```
⇒
```
> f ← function (y, ftn)
{
d ← 8
print (environment (ftn))
return (ftn (d, y))
}
> h ← function (dee, yyy)
{
return (dee * (w+yyy))
}
> w ← 12
> f (3, h)
```
when f() executed the formal argument ftn wear matched by the actual argument h() ftn has h environment

More on ls()

31/05/2022

ls() function is from with in a function returns the name of the current local variables with the environment arguments it will point the names of the local of any frame in the calling function

Eg:-
```
f <- function (y)
{
  d<-8
  return (h(d,y))
}
h<-function (dee, yyy)
  Print(ls())
Print (ls(envir = parent.frame (n=1)))
 return (dee * (w+yyy))
}
>f(2)
```

o/p:- [1] "dee" "yyy"
[1] "d" "y"
112

In the above example parent.frame(), the argument "n" specifies how many frames to go up in the call chain (or) call function. here we were in the executing of h() which had been called from f. so, specifing n=1 gives us f()'s frame and thus we get the locals

Function have no side effects :-

function programming physically is that functions do not change non-local variables that is generally their are no side effects the code in the function. has read access to its non-local variables but it doesnot have write access to them. in the previous examples code than appear to re assign those values but the action with will effect only copies. not the variables themselves. lets see the one example by adding some more go to our previous example.

```
>w ← 13
 f ← function (y)
  {
   a1 ← 8
   w ← w+1
   y ← y-2
  Print (w)
h ← function ()
 {
  return (h())
  Print (w)
 }
  t ← 4
   f(t)
```

O/P: [1] 13

```
   > w
     12
   > t
     4
```

Here w at the top level didnot change even though it appear to change with in f(). Only a local copy of w with in f() changed Similarly the top level variable 't' didn't Change even though its associated formal argument 'y' did Change

Note :- Reference to the local w actually go to the same memory location as the global one. until the value of the local changes. In that case a new memory location is used

Extendend Example: A function to display the contents of a call frame

The values of the local variables in our current function we want to know through our board in a debugging Settings using Single step in that means to know the values of the locals in the parent function i.e the one from which the current function was called here we will develop code to display this values their by further demonstrating access to the environment hairarechy

```
Eg: f ← function()
    {
     a ← 1
    return (g(a)+a) o/p:- 2
    }
    g ← function (aa)
    { b ← 2
    aab ← h(aa+b) o/p:- 3
      return (aab)
    }
    h ← function (aaa)
     { c ← 3
      return (aaa+c) o/p:- 4
```

g(1)+1
g(a)
g(aa)

when we calls f() it inturns calls g() which then calls h() we want to know the values of the local variables of the current function. the variables aa, aab, b and while we are in g() we also wish to know the values of the locals in f() at the time of the call to g(). as well as the values of the global variables.

09/06/2022.

No pointers in R

R does not have variables corresponding to pointers or references like 'c' language this can make programming more dejecult in some cases

for example you can't write a function directly changes its arguments like

```
Eg:- > x = [13, 5, 12]   (array)
     > x.sort()
     > x
o/p:- 5  12  13
```

The current version of R programming has an experimental feature called reference classes which may reduce the dejicully hence in the above example the value of x, the argument to sort() changed

```
eg2 :- > x ← c(13, 5, 12)
       > sort(x)
                          > x
o/p:- 5  12  13            13  5  12
```

Eg:- > x ← c (13, 5, 12)
> x ← sort(x)
> x
5 12 13

In the example 2 the argument to sort does not change if we do want x to change in this R code the solution is to re assign the arguments.

Writing upstairs :>

In any variable higher in the environment hairarechy then the level at which your writing statement exists. we can use the stepper "super assignment operator" or the Assign function. because we cannot directly write access to variables at the higher level why are the standard '←' operator is not possible.

Writing to non-locals with the super Assignment operator.

```
Eg:- two ← function (x)
     {
     u <<— 2*u
     z ← 2*z
     }
     > x ← 1
     > z <<— 3
     > u
```

Error : "Object "u" not found

```
> two(x)
> x   [1] 1
> z   [3] 3
> u   [2]
```

1. x: was the actual argument two() in the above example it retains the value 1 after the call. this is because its value 1 was copied to the formal argument u. which is treated as a local variable with in the function. Thus when u changed, x did not change with it

z: The two z values are entirely unrelated to eachother one is top level and other is local to two(). The change in the local variable has no effect on the global variable. Of course having two variables with the same name is probably not good programming

u: The u value didnot even exist as a top level variable prior to two(). Hence the not found error massage. It was created as a top level variable by the super assignment operator to function → two(). As confirmed after the call

```
eg>:. >f <-function()
      {
    fn <<- function()
      {
      x <<- x+1
    }
    x <<- 3 [supper assignment]
     fn()
    return(x)
    }
    >x
    f(n)
    [1] 4
    >x
Error : object 'x' is not found
```

o/p:4

10/06/2022

Here inc is defined within function f, when inc() is executed when R interpreter sees a 'super assignment to x it starts going of the hierarchy at the first top level the environment with fn function & it doesnot find an x and so that x is the one that return() to not x at the top level

writing to non-locals with assign:-

Assign a value to name in environment assign() to write to upper level variables in R current versions

Syntax:-

assign (variable, value)

Example :-

```
> assign (a, 10)
> Print (a)
[1] 10
```

Eg:-

```
> two <- function (a) {
> assign ("a", a^2, pos.== Global Env)
 z <- z^z
}
> two (x)
> x
 [1]
> a
 [2]
```

In the above example we replaced the super assignment operator with a call to assign() function. that call instructs R to assign the value z^2 to a variable. A further up the call stage. specifically in the top level environment

Paste() :-

Here we are using paste() to create new variables combining the prefix x- with the remaining index!

Eg:-

```
for (i in 1:3)
{
assign (paste ("x-", i));
}
```

O/P:

```
x-1
[1] 1
x-2
 [1] 2
x-3
 [1] 3
```

13/06/2022

Extended Example:

Discreate Event stimulation in R :-

DES is widely used in businesses, industry and government sectors. the term discreate refers to the fact that the state of the system changes only in discreate quantities rather than changing continuously

eg:- ATM machine

this example the state of our system at a time 't' to be the no of people in queue at the time. the state changes only by +1 when someone arrives, or by -1 when a person finishes a an ATM transaction.

General to DES operation is maintainence of the event list which is simply a list of scheduled events. this is general DES term so the word list here does not refere to the R Data type. Instead of that we will represent the event list by a data frame.

Ex:- cus 1 arrives at time 23.12
2 arrives at time 25.88
3 arrives at time 25.97
1 finisher services at time 26.02

So. In the above data frame with the 1st row containing the earlrest schedule the 2nd containing the 2nd earliest and so on

In this example if the customer arrival occure when the queue is empty. that customer servies will belog begin one event triggers setting upto another

The person will be finished which is another event that must be added to event list

One of the oldest approch to writing DES code is the event oriented approch. consider the ATM situation at time '0' (zero) the queue is empty the stimulation code randomly generates the code of the 1st arrival at the 2.3 at this point the event list is simply (2.3, arrival) . this event is pulled of the list stimulated time is updated to 2.3

. The queue for the ATM is empty so we start the service by randomly generating the service time say it is 1.2 time units. then the completion of service will occur at stimulated time 2.3 + 1.2 = 3.5 ,,

library functions using in DES :-

- Scheduled event → Schedevent():
  Insert a newly created event into the event list
- getnext event(): pulles the earliest event of the event list
- dosimulation → dosim()
  include the core loop of the simulations repeatedly calls get next event to get earliest of the pending events, updates the current simulated time.
- Sim$ curtime :
  to reflect the occurances the event and calls the application specific function. reactevent() to process this newly occured event.
- React event(): It performs the event when it will occurs
- Initglbls(): It is used to initialize global variables.
- Printoslts() → print results.
  Prints the application specific results of specific events

When should we use global variables? 14/06/2022

Variables that are created outside of function are known as global variables.

- Global variables can be used by every programmer, But inside and outside of a function.
- The term global variable will be used to include any variable located higher in the environment then the level of the given code of interest in R library super assignment operator is used in many functions. So, treaded code and GPU code which are used for writing fast programs to make havey use of global variables, which provides the main avenue of communication b/w parallel actors

(debuging, maintance and error detection used)

Eg-1:
txt ← "RProgramming"
my_fun ← function()
{
Paste (txt, "language")
}
my_function()
o/p: Rprogramming language.

Eg-2:
> txt ← "programming language"
> my_fun ← function()
{
txt ← "alphabet"
Paste ("R is", txt)
}
> my_function()
o/p: R is alphabet
> txt
o/p: Programming language

So the above two examples are cleaner code is usually easier to write, debug and maintain for this reason avoiding un interepted errors or uncleared errors and simplifing the code that we choose to use global

Closures :=

R closure consisting of a functions arguments and body together with its environment at the time of the call. The fact that the environment is included in explioted in a type of programming that uses a feature also known as closure.

A closure consisting of a function that steps up a local variable and then creates another function that access the variables

Eg:
> counter ← function()
{
ctr ← 0
f ← function()
{
ctr ← ctr+1
}
Cat ("This count currently has value", ctr, "\n")
}
o/p:- 1
> c1 ← counter()
c2 ← counter()
c1 ← function()
{
ctr ← ctr+1
}
Cat ("this count currently has value", ctr, "\n")
}

counter()
> c2 ← c1()
{
ctr ← ctr1 + 1
}
Cat ("This count currently has value", ctr, "\n")
}
}

In the above example c_1 function and c_2 functions series as completly independent counters where we invoke each of them a few times.

Recursion :-

12/06/2022

A recursion function call itself the idea is this.

1. To solve a problem type x by writing a recursive function:
- Break the orginal problem of type x into one or more smaller problems of type x
- within f() call f() on each of the smaller problems
- within f() piece together the result of it point to solve the orginal problem.

Quick sort implementation :-

In quick sort Algorithm used to sort a vector of number from smallest to largest

Ex:- 5, 4, 12, 13, 3, 8, 88

- First compare every thing to the first element 5
- To form two sub vectors one consisting of the elements less than 5 and other consisting of the gender than or = to 5
- i.e sub vector 1 is 4 3 & sub vector 2 is 12 13 8 88
- we than call the function on the sub vector 1 it returns 3, 4 and sub vector 2 returning 8 12 13 88 then we string goes together with the file i.e 3, 4, 5, 8, 12, 13, 88

Source code in R :-

```
qs <- function(x)
{ if (length(x) <= 1)
  return(x)
  PNot <- x[1]
  therest <- x[-1]
  sv1 <- therest [therest < pNot]
 sv2 <- therest [therest >= pNot]
  sv1 <- qs(sv1)
  sv2 <- qs(sv2)
return (c(sv1, pNot, sv2))
}
```

from the above source code if (length(x) <= 1) return(x) without the function would keep calling itself.

repeatedly an empty vector executing forever. So, recursion certainly is an elegate way be solve many problems (99)
But recursion has two potential drawbacks.
- It's fairly abstract i.e recursion is really just the inverse of proof by mathematical induction; but many programmer find it tough.
- Recursion is very lavish in its use of memory which may be an issue in R. If apply to large problems

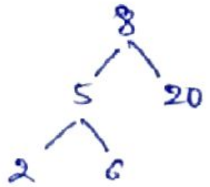
Binary Search Tree (Extended Example) :- 20/6/2022

The library function in R programming to create a new tree use "TsTr" In other programming languages like C fortune we use Data structures are common in both computer science and Statistics

But in rpart library for a recursive portioning approch to regression and classification is very popular.

Trees can be implemented in R itself and it performs it is not an issue

eg:-



We know we very well binary search tree pondple 1st value is goo root. the value at leftly if any less than or equal to that of the root (parent) while the value at the rightly if any greater than of the parent (root)

A matrix in R is a time consuming activity we alocate new space for a trees martix several rows at a time instead of row by row the no. of rows allocated each time will be given the variable INC (increment) as it is common with tree travels we implement our algorithm with recursion

```
Eg:- > x <- new tree(8,3)
> x
$mat
     [,1] [,2] [,3]
[1,]   NA   NA    8
[2,]   NA   NA   NA
[3,]   NA   NA   NA

$nxt
[1] 2

$inc (increment)
[1] 3
```

```
>n<-ins(1,n,5)
>n
$mat
     [,1] [,2] [,3]
[1,]    2   NA    8
[2,]   NA   NA    5
[3,]   NA   NA   NA

$nxt
[1] 3
$inc
[1] 4
```

```
>n<-ins(1,n,6)
>n
$mat
     [,1] [,2] [,3]
[1,]    2   NA    8
[2,]   NA    3    5
[3,]   NA   NA    6

$nxt
[1] 4
$inc
[1] 5
```

```
>n<-ins(1,n,2)
>n
$mat
     [,1] [,2] [,3]
[1,]    2   NA    8
[2,]    4    3    5
[3,]   NA   NA    6
[4,]   NA   NA    2
[5,]   NA   NA   NA
[6,]   NA   NA   NA

$nxt
[1] 6
$inc
[1] 7
```

O/P:

```
>n<-(1,n,20)
>n
$mat
     [,1] [,2] [,3]
[1,]    2    5    8
[2,]    4    3    5
[3,]   NA   NA    6
[4,]   NA   NA    2
[5,]   NA   NA   20

$nxt
[1] 7
$inc
[1] 8
```

The first command containing newtree(8,3) that means that creates a new tree assigned to n storing the no. 8. the argument 3 specifies that we allocate storeage room 3 rows at a time. the result already shown

The next command ins(1,n,5) to insert a second value 5 into the tree n. the argument 1 specifies the root. that means insert 5 in the subtree of n whose root is in row 1. So the result already shown

The element 2 means (in the above result) that the left link out of the node containing 8 is mean to point to row2 where our new element 5 is stored.

The session continuous that when our intial alotment of 3 rows is full ins() allocates three new rows for a total of 6 rows.

22/06/2022

Replacement functions :-

In R programming the replacement function named as "names"

Eg:-1 > x ← c (1,2,3)
> Replace (x) o/p:- NULL
> names (x) o/p:- NULL

eg2:- > x ← c (1,2,3)
> names (x) ← (c("a", "b", "c"))
> names (x) o/p:- "a" "b" "c"
1 2 3

eg3:- > x ← c (1,2,3)
> names (x) ← c("a","b","z")
> Replace (y, names(x), "x", "y", "z")
> Replace (y)
o/p:- x y z
1 2 3

In the above examples we will use the names and replace functions in the R console

What's consider a Replacement functions :-

Any assignment statement in which the left side is not just a identifier is consider as a replacement function.

Eg:- > g(u) ← v
> u ← "g ← "(u, value = v) } same executes

Note :- the replacement function has one or more arguments than the orgmal function g(), a named argument value for reasons replaced. same.

Eg2: > x ← c (8, 88, 5, 12, 13)
> x 8, 88, 5, 12, 13
> x[3] o/p:- 5
> "[" (x,3) o/p:- 5

in the above example we will use the function "[(" if for reading vecto elements. the complicated call of a function "[←"() is used to write.

Eg3:- > "p ←" (x, 2:3 value = 99:100)
> x o/p:- 8 99 100 13

Is simply performing what happen in the colsone when execute this x[2,3] → 99:100. we can easily verfiy what occuring like

Eg 4:- > x ← c (8, 88, 5, 12, 13)
> x [2:3] ← 99:100
> x
o/p: 8 99 100 12 13