C and R programming: if-else
In C, the general formats for these are,

```c
if( condition 1 ) {
    statement1;
}
else if( condition 2 ) {
    statement2;
}
else {
    statement3;
}
```

An example would be:

```c
#include <stdio.h>

main()
{
    int i;
    printf("Please choose an integer.");
    scanf("%d", &i);

    /* Here is a simple example.*/
    if ( i%2 == 1) {
        printf("It is an odd number.\n");
    }
    else {
        printf("It is an even number.\n");
    }

    /*Here is a complicated one*/
    if (i > 10) {
        printf("The number is larger than 10.\n");
    } else if (i > 5) {
        printf("The number is larger than 5.\n");
    } else {
        printf("The number is smaller or equal to 5.\n");
    }
}
```

Note: In C, true is equivalent to any nonzero value, and false is equivalent to 0. Please see the following example,
While in R, we have

```
if( condition 1) {
    statement1
} else if(condition2) {
    statement2
} else if(condition3) {
    statement3
}
```

An example would be:

```
cat("Please choose an integer.
"

i <- scan(file = ",", what = integer(0), nmax =1)

# Here is a simple example.
if ( i%%2 == 1) {
    cat("It is an odd number.
"
} else {
    cat("It is an even number.
"
}

# Here is a complicated one
if (i > 10) {
    cat("The number is larger than 10.
"
} else if (i > 5) {
    cat("The number is larger than 5.
"
} else {
    cat("The number is smaller or equal to 5.
"
}
```

Note: 1. to execute an .R file. Use command ‘source’ in R.

2. In R code, make sure ‘}’ and “else” are in the same line.
C and R programming: for/while-loop

for in C:

```
for (initialization_expression; loop_condition; increment_expression) {
    statements
}
```

*initialization_expression* is executed before execution of the loop starts. This is typically used to initialize a counter for the number of loop iterations. You can initialize a counter for the loop in this part.

The execution of the loop continues until the *loop_condition is false*. This expression is checked at the beginning of each loop iteration.

The *increment_expression* is usually used to increment the loop counter. This is executed at the end of each loop iteration.

Example:

```c
#include <stdio.h>

main()
{
    int i;
    for(i = 1; i <= 10; i++) {
        printf("%d
",i);
    }
}
```

for in R:

```
for (variable in sequence) {
    statements
}
```

Example:

```r
for(i in 1:10) {
    cat(i,"\n")
}
```

while in C and R:

```
while (condition) {
    statements;
}
```
Examples:

```c
#include <stdio.h>

int a[3] = {2,1,3};
```

and

```c
i <- 1;
while(i <= 10) {
    cat(i,"
    i <- i + 1;
}
```

**Array in C**

We can declare an array by specify its data type, name and the number of elements the array holds between square brackets immediately following the array name. For example,

```c
int a[3];
```

To initialize an array, you provide initializing values which are enclosed within curly braces in the declaration and placed following an equals sign after the array name. Here is an example of initializing an integer array.

```c
int a[3] = {2,1,3};
```

Each array element occupies consecutive memory locations and array name is a pointer that points to the first element. (We will come back later.)
Note: The first element in an array is a[0];

An array with more than one index value is called a multidimensional array. An example would be

```c
int matrix[3][3] = {11, 12, 13, 21, 22, 23, 32, 31, 33};
```

Array is very effective when we combine it with loop. For example, we want to compute the first 10 values of the Fibonacci sequence. Each number in the sequence is the sum of the previous two numbers.

```c
#include <stdio.h>
main()
{
    int i, fib[10];
    fib[0] = 0, fib[1] = 1;
    for(i = 2; i < 10; i++) {
        fib[i] = fib[i-1] + fib[i-2];
    }
    for(i = 0; i < 10; i++) {
        printf("%d  ", fib[i]);
    }
    printf("\n");
}
```

**Array in R:**

In R, usually we use a column vector to express an array. For example,
To create a matrix, we can use statement ‘matrix’. For example

\[
A <- c(1,2,3)
\]

\[
a <- c(11,12,13,21,22,23,31,32,33)
ma <- matrix(a, ncol = 3, nrow =3, byrow = TRUE)
\]

**Function in C:**
A function in C has a *name* that you call it by, and a list of zero or more arguments or parameters that you hand to it for it to act on or to direct its work; it has a *body* containing the actual instructions (statements) for carrying out the task the function is supposed to perform; and it may give you back a *return value*, of a particular type. An example would be

```c
#include <stdio.h>
double multbytwo(double x); /* prototype the function*/

main()
{
    double a, res;
    printf("Please give a number\n");
    scanf("%lf", &a);
    res = multbytwo(a);
    printf("The result is %lf\n", res);
}
double multbytwo(double a) {
    double res;
    return a * 2;
}
```

**Note:** 1. You can define your own function before main function. For example

```c
#include <stdio.h>

double multbytwo(double a) {
    double res;
    return a * 2;
}

main()
{
    double a, res;
    printf("Please give a number\n");
    scanf("%lf", &a);
    res = multbytwo(a);
    printf("The result is %lf\n", res);
}
```
2. The main function is the function that will be "called" first when our program starts running.
3. Use 'void' if you don’t have any input or return value.

**Function in R:**

In R functions are objects and can be defined using the following syntax,

```r
name <- function(arg1, arg2, ...) expr1
```

An example of function would be

```r
multbytwo <- function(a) {
  res <- a * 2
  res
}
cat("Please give a number\n")
i <- scan(file = "", what = double(0), nmax =1)
res = multbytwo(i);
cat("The result is",res,"\n");
```