Loop

Loops are n-iterative control structure. They have:

1. Condition
2. Code block

If the condition is – (a) TRUE: The code block runs repeatedly.

(b) FALSE: The code block stops and loop breaks.

Two types of loops:

1. Entry controlled:

   This type of loop checks the condition before the execution of the code block i.e. , they check if the condition is true while entering the loop.

   a. **for loop**
   b. **while loop**

2. Exit controlled:

   This type of loop checks the condition after the execution of the code block i.e. they check if the condition is true while exiting the loop.

   a. **do while loop**

for loop

- It is an entry controlled n-iterative control structure.
- Its syntax includes initialization, condition and increment/decrement in the same line.
- Syntax:

```
for(int i=0;i<n;i++){
    //code block
}
```

  Here *i* is called the **Loop variable**. Its value is initialized with 0 and the loop will continue till value of *i* is less n. After every iteration, the value of *i* increments by 1.

- Like *if*, when the code block contains only statement, then { } is not necessary.
- *for* loop can also be written without providing initialisation, condition or increment/decrement.

  For eg:

```
//for loop without initialisation
for( ;i<n;i++){
    //code block
}
//for loop without condition and initialisation
for( ; ;i++){
    //code block
}
```

```
    //infinite loop using for loop
    for(;;){
      //code block
    }
```

Example of for loop:

```
int main(){
    int n;
    printf("Enter a number: ");
    scanf("%d",&n);
    for(int i=1;i<=n;i++){
        printf("%d ",i);
    }
    return 0;
}
```

```
Enter a number: 10
1 2 3 4 5 6 7 8 9 10
Enter a number: 5
1 2 3 4 5
```

Here, the value of loop variable *i* is initialised by 1 and the loop continues until the value of *i* is less than or equal to the value of n (Input value). In every iteration of the loop, it prints the value of *i* followed by a space.

while loop

- It is also an entry controlled n-iterative control structure.
- Its syntax only includes the condition, initialisation and increment/decrement must be done in different line.
- Syntax:

```
while(condition){
        //code block
}
```

  Here, *condition* block includes loop condition till which the loop needs to be continued.
- An infinite while loop can be written by providing a valid *true* condition. For example:

```
while(1){
        printf("This is an infinite loop.\n");
}
```

  Here, writing while(1) means while(1!=0) and that condition is always true. That means, the message "*This is an infinite loop.*" will continue to show until we manually stop the program.

Example of while loop:

```
//print all the power of 2 less than 100
int main(){
    int i=1;
    while(i<100){
        printf("%d ",i);
        i = i * 2;
    }
    return 0;
}
```

do while loop

- It is an exit controlled n-iterative control structure.
- This loop checks the condition after executing the code block first. That means even if the condition is false, the code block will run at least one time.
- Syntax:

```
do{
    //code block

}
while(condition);
```

  Here, *condition* block includes loop condition till which the loop needs to be continued. An important observation here is that there is a semicolon (;) just after while(condition) which is different from while loop syntax.
- This loop can be used as a user operated loop i.e., we can use this loop to repeat certain thing until the user wants to continue. For example:

```
int n;
do{
      printf("Enter a number (Negative number to stop) : ");
      scanf("%d",&n);
      printf("%d\n",n);
}
while(n>0);
```

  In this example, the loop repeatedly asks for an input and continues to repeat until a negative number is entered. An important thing to note here is even if we have not initialised the value of *n,* the loop will start to work.

Example of do while loop:

```
int main(){
    char ch;
    int n;
    do{
        printf("\nEnter a number to find its square: ");
        scanf("%d",&n);
        printf("Square of %d is %d\n",n,n*n);
        printf("Do you want to continue? (Y/N): ");
        ch = getche();
    }
    while(ch=='y'||ch=='Y');
    return 0;
}
```

In this example, we have created a square calculator program which continues to operate until user wants. Here, *getche()* is a function under *conio.h* header file which takes one character stroke and returns it.

When to use for loop:

1. When number of iteration (number of times loop works) is known/fixed.
2. Value of iterator (loop variable) is required/useful.
3. Mostly used in Arithmetic progression related problems. For example: counting from 1 to n, printing even numbers between 1 and 100, etc.

When to use while loop:

1. When number of iteration is not known/fixed.
2. Value of iterator is not required, or if no iterator is required.
3. Mostly used in Geometric progression related problems. For example: printing power of 2, printing digits of a given number, etc.

When to do while loop:

1. This loop has mostly same use as while loop but only difference is that the while loop is entry controlled whereas do while loop is exit controlled.
2. Used in a program where user control is required (As mentioned in the above examples).

NOTE: We have mentioned the most usual cases when these loops are used. Although any loop can do the work same as the other loop, that means we can write same program using any of the three loops. The reason that we have provided the conditions when to use which loop, is only for better program readability and understanding.