LOOP

Computers are very good at performing repetitive tasks very quickly. In this section we will learn how to make computer repeat actions either a specified number of times or until some stopping condition is met.

while Loops (Condition-Controlled Loops)

- Both while loops and do-while loops (see below) are ***condition-controlled***, meaning that they continue to loop until some condition is met.
- Both while and do-while loops alternate between performing actions and testing for the stopping condition.
- While loops check for the stopping condition first, and may not execute the body of the loop at all if the condition is initially false.
- Syntax:

```
while( condition )
    body;
```

where the body can be either a single statement or a block of statements within { curly braces }.

- Example:

```
int i = 0;
while( i < 5 )
    printf( "i = %d\n", i++ );
printf( "After loop, i = %d\n", i );
```

do-while Loops

- do-while loops are exactly like while loops, except that the test is performed at the end of the loop rather than the beginning.
- This guarantees that the loop will be performed at least once, which is useful for checking user input among other things (see example below.)
- Syntax:

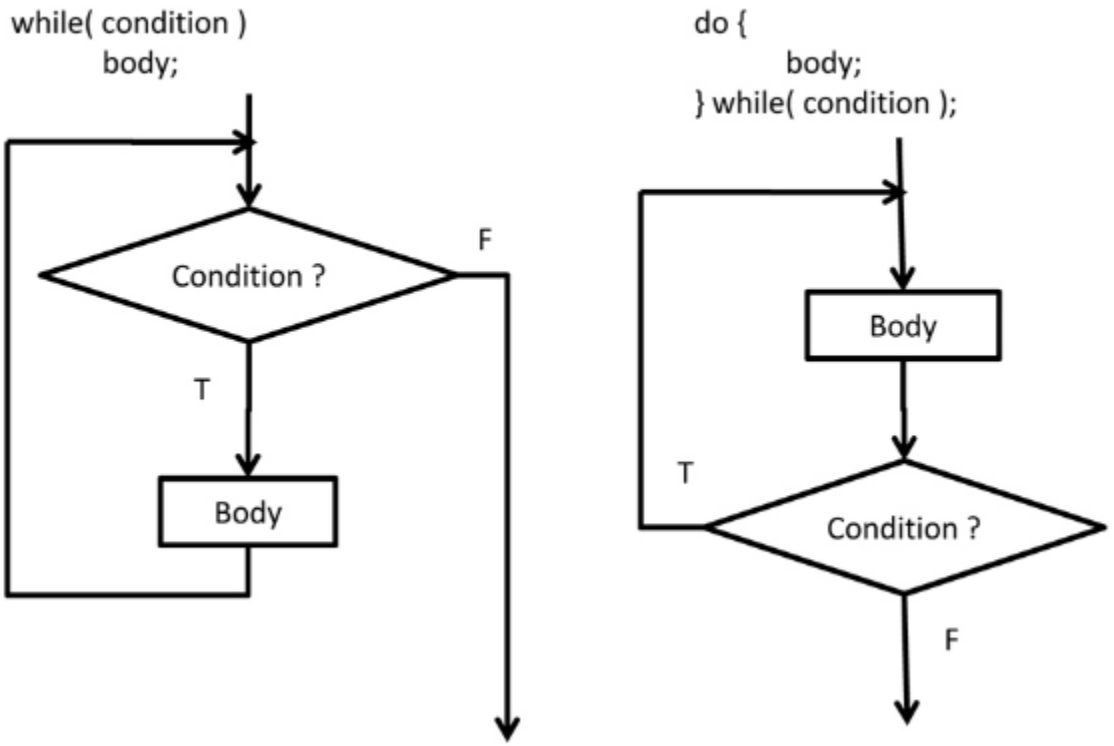
```
do {
    body;
} while( condition );
```

- In theory the body can be either a single statement or a block of statements within { curly braces }, but in practice the curly braces are almost always used with do-whiles.
- Example:

```
int month;
do {
    printf( "Please enter the month of your birth > " );
    scanf( "%d", &month );
} while ( month < 1 || month > 12 );
```

- Note that the above example could be improved by reporting to the user what the problem is if month is not in the range 1 to 12, and that it could also be done using a while loop if month were initialized to a value that ensures entering the loop.
- The following diagram shows the difference between while and do-while loops. Note that once you enter the loop, the operation is identical from that point forward:

While versus Do-While Loops



Empty Loops

- A common error is to place an extra semi-colon at the end of the while or for statement, producing an empty loop body between the closing parenthesis and the semi-colon, such as:

```
int i;
for( i = 0; i < 5; i++ ); // Error on this line causes empty loop
   printf( "i = %d\n", i );  // This line is AFTER the loop, not inside it.
```

- or:

```
int i = 0;
while( i < 5 );  // Error - empty loop on this line
   printf( "i = %d\n", i++ ); // Again, this line is AFTER the loop.
```

- In the case of the while loop shown above, it is not only empty but also infinite.
- There are some very rare circumstances in which a programmer will deliberately write an empty loop, most of which are beyond the scope of this course. (This is know as a ***busy loop.***)
  In this case, the semicolon should be placed on a line by itself, and clearly commented to indicate that the empty loop is deliberate and not an oversight:

```
while( ( error = someFunction( ) ) != 0 )
   ;  // Empty loop - Does nothing forever, until someFunction returns a zero
printf( "error = %d\n", error ); // After the loop.  error must be zero to get here.
```

WRP to Print Table of any Given Number

Given a number n as input, we need to print its table.

Examples :

Input : 5
Output : 5 * 1 = 5
5 * 2 = 10

5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50

Input : 8
Output : 8 * 1 = 8
8 * 2 = 16
8 * 3 = 24
8 * 4 = 32
8 * 5 = 40
8 * 6 = 48
8 * 7 = 56
8 * 8 = 64
8 * 9 = 72
8 * 10 = 80
8 * 11 = 88
8 * 12 = 96

Example 1: Display Multiplication table up to 10

- C++
- Java
- Python
- C#
- PHP
- Javascript

Output :

5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50

This program above computes the multiplication table up to 10 only.
The program below is the modification of above program in which the user is also asked to entered the range up to which multiplication table should be displayed.

Example 2: Display multiplication table up to a given range

- C++
- Java
- Python
- C#
- PHP
- Javascript

```
// CPP program to print table over a range.
#include <iostream>
using namespace std;
int main()
{
   int n = 8;   // Change here to change input number
  int range = 12; // Change here to change result.
   for (int i = 1; i <= range; ++i)
      cout << n << " * " << i << " = "
         << n * i << endl;
   return 0;
}
```

Output:

```
8 * 1 = 8
8 * 2 = 16
8 * 3 = 24
8 * 4 = 32
8 * 5 = 40
8 * 6 = 48
8 * 7 = 56
8 * 8 = 64
8 * 9 = 72
8 * 10 = 80
8 * 11 = 88
8 * 12 = 96
```

WRP to Print Pyramid

In this example, you will learn to print half pyramids, inverted pyramids, full pyramids, inverted full pyramids, Pascal's triangle, and Floyd's triangle in C Programming.

Example : Half Pyramid of Numbers

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

C Program

```
#include <stdio.h>
```

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

WRP to Find Given Number is Prime or Not

A prime number is a number that is divisible only by two numbers itself and one. The factor of a number is a number that can divide it.

The list of the first ten prime numbers is 2,3,5,7,11,13,17,23,29,31.

A number that is not prime is a composite number. A composite number is a number that can be divided by more than two numbers.

Elser then prime and composite there is 1 which is neither Prime nor composite because it can be divided only by itself.

How to check if a number is prime or composite to check if a number is prime there are two conditions that should be checked

1) It should be a whole number greater than 1.

2) it should have only two factors i.e one and the number itself.

If these two conditions are satisfied, then we can say a number is a prime number.

In our program, we will check dividing the number by each number smaller than that number. If any number smaller than the given number divides it then it is not Prime number. Otherwise, it is a prime number.

Let's take an example of two numbers and check whether they are prime or not using this process.

Input − Number1 − 42
Output − 42 is not a prime number
Logic − We will divide 42 by every number greater than 1 and smaller than 42. So,

42/2 = 21 i.e. 42 is divisible by 2, this means 42 is not a prime number because it is divisible by another number.

Input − Number2 − 7
Output − 7 is a prime number
Logic − We will divide seven by every number greater than 1 and smaller than 7. So,

7 is not divisible by 2, so the code will check for the next number i.e. 3

7 is not divisible by 3, so the code will check for the next number i.e. 4

7 is not divisible by 4, so the code will check for the next number i.e. 5

7 is not divisible by 5, so the code will check for the next number i.e. 6

7 is not divisible by 6, This means that 7 is divisible by only 1 and 7 this means 7 is a prime number.

look at the above logic what does the number would be 1000 plus or 100000 Plus then the program would take that many iterations for the for a loop this method would take a lot of computation time. So to reduce the number of iterations they must be a better way.

An optimised solution to this is run the loop only halfway. this means if the number is 77 the loop will run only till 38. This will reduce the number of iterations required so we will use this algorithm to create our program.

Example

```
#include <stdio.h>
int main() {
  int num = 33, flag = 0;
  for(int i=2 ; i < num/2 ; i++) {
    if(num%i == 0) {
      printf("%d is not a prime number", num);
      flag = 1;
      break;
    }
  }
```

```
  if(flag == 0) {
    printf("%d is a prime number", num);
  }
}
```

Output

```
33 is a prime number
```