

**CCA-101:**  
**Fundamentals of**  
**IT &**  
**Programming**

**Assignment - 2**

**Q1 . What is the difference between Machine Language and High Level Language?**

**Ans.**

**Machine language, or machine code,**

is the only language that is directly understood by the computer, and it does not need to be translated. All instructions use binary notation and are written as a string of 1s and 0s. A program instruction in machine language may look something like this:

1. 10010101100101001111101010011011100101

**High-Level Language**

is a programming language that uses English and mathematical symbols, like +, -, % and many others, in its instructions. When using the term 'programming languages,' most people are actually referring to high-level languages. High-level languages are the languages most often used by programmers to write programs. Examples of high-level languages are C++, Fortran, Java and Python.

To get a flavor of what a high-level language actually looks like, consider an ATM machine where someone wants to make a withdrawal of \$100. This amount needs to be compared to the account balance to make sure there are enough funds. The instruction in a high-level computer language would look something like this:

1. x = 100  
2. if balance x:  
3. print 'Insufficient balance'  
4. else:  
5. print 'Please take your money'

This is not exactly how real people communicate, but it is much easier to follow than a series of 1s and 0s in binary code.

There are a number of advantages to high-level languages.

The **first advantage** is that high-level languages are much closer to the logic of a human language.

The **second advantage** is that the code of most high-level languages is portable and the same code can run on different hardware

**Q2. Discuss about different data types of C programming Language.**

**Ans. Data types in C Language**

Ad by Valueimpression

Data types specify how we enter data into our programs and what type of data we enter. C language has some predefined set of data types to handle various kinds of data that we can use in our program. These datatypes have different storage capacities.

**C language supports 2 different type of data types:**

**1. Primary data types:**

These are fundamental data types in C namely integer(`int`), floating point(`float`), character(`char`) and `void`.

**2. Derived data types:**

Derived data types are nothing but primary datatypes but a little twisted or grouped together

like [array](#), [structure](#), [union](#) and [pointers](#). These are discussed in details later.

Data type determines the type of data a [variable](#) will hold. If a variable `x` is declared as `int`. it means x can hold only integer values. Every variable which is used in the program must be declared as what data-type it is.

# Integer Type

Type	Size(bytes)	Range
int or signed int	2	-32,768 to 32767
unsigned int	2	0 to 65535
short int or signed short int	1	-128 to 127
unsigned short int	1	0 to 255
long int or signed long int	4	-2,147,483,648 to 2,147,483,647
unsigned long int	4	0 to 4,294,967,295

Integers are used to store whole numbers.

**Size and range of Integer type on 16-bit machine:**

Floating point type

Floating types are used to store real numbers.

Type	Size(bytes)	Range
Float	4	3.4E-38 to 3.4E+38
double	8	1.7E-308 to 1.7E+308
long double	10	3.4E-4932 to 1.1E+4932

**Size and range of Integer type on 16-bit machine**

## Character type

Character types are used to store characters value.

**Size and range of Integer type on 16-bit machine**

Type	Size(bytes)	Range
char or signed char	1	-128 to 127
unsigned char	1	0 to 255

## void type

`void` type means no value. This is usually used to specify the type of functions which returns nothing. We will get acquainted to this datatype as we start learning more advanced topics in C language, like [functions](#), pointers etc.

**Q3. Find the output of the following expressions**

**A)  $X=20/5*2+30-5$**

**B)  $Y=30 - (40/10+6) +10$**

**C)  $Z= 40*2/10-2+10$**

**Ans.**

**A)  $X=20/5*2+30-5$**

**$X=4*2+30-5$**

**$X=8+30-5$**

**$X=38-5$**

**$X=33$**

**B)  $Y=30 - (40/10+6) +10$**

$$\mathbf{C) Z = 40 * 2 / 10 - 2 + 10}$$

$$\mathbf{Z = 40 * 0.2 - 2 + 10}$$

$$\mathbf{Z = 8 - 2 + 10}$$

$$\mathbf{Z = 6 + 10}$$

$$\mathbf{Z = 16}$$

**Q4. Describe the syntax of the following statements**

**A) If - else statement**

**B) for loop**

**C) while loop**

**D) do-while loop**

**Ans.**

**(A) If - else statement**

**if Statement**

The syntax of the `if` statement in C programming is:

```
if (test expression)
{
    // statements to be executed if the test expression is true
}
```

**How if statement works?**

The `if` statement evaluates the test expression inside the parenthesis `()`.


- If the test expression is evaluated to true, statements inside the body of `if` are executed.
- If the test expression is evaluated to false, statements inside the body of `if` are not executed.

#### Expression is true.

```
int test = 5;

if (test < 10)
{
    // codes
}

// codes after if
```



#### Expression is false.

```
int test = 5;

if (test > 10)
{
    // codes
}

// codes after if
```



To learn more about when test expression is evaluated to true (non-zero value) and false (0), check [relational](#) and [logical operators](#).

### Example 1: if statement

```
// Program to display a number if it is negative
```

```
#include <stdio.h>
int main() {
    int number;

    printf("Enter an integer: ");
    scanf("%d", &number);

    // true if number is less than 0
    if (number < 0) {
        printf("You entered %d.\n", number);
    }

    printf("The if statement is easy.");
}
```



```
return 0;
}
```

### Output 1

```
Enter an integer: -2
You entered -2.
The if statement is easy.
```

When the user enters -2, the test expression `number<0` is evaluated to true. Hence, `You entered -2` is displayed on the screen.

### Output 2

```
Enter an integer: 5
The if statement is easy.
```

When the user enters 5, the test expression `number<0` is evaluated to false and the statement inside the body of `if` is not executed

### else Statement

The `if` statement may have an optional `else` block. The syntax of the `if..else` statement is:

```
if (test expression) {
    // statements to be executed if the test expression is true
}
else {
    // statements to be executed if the test expression is false
}
```

## How if...else statement works?

If the test expression is evaluated to true,

- statements inside the body of `if` are executed.
- statements inside the body of `else` are skipped from execution.

If the test expression is evaluated to false,

- statements inside the body of `else` are executed
- statements inside the body of `if` are skipped from execution.

### Expression is true.

```
int test = 5;

if (test < 10)
{
    // body of if
}
else
{
    // body of else
}
```

### Expression is false.

```
int test = 5;

if (test > 10)
{
    // body of if
}
else
{
    // body of else
}
```

## Example 2: if...else statement

```
// Check whether an integer is odd or even
```

```
#include <stdio.h>
int main() {
    int number;
    printf("Enter an integer: ");
    scanf("%d", &number);

    // True if the remainder is 0
    if (number%2 == 0) {
```

```
    printf("%d is an even integer.",number);
}
else {
    printf("%d is an odd integer.",number);
}

return 0;
}
```

### Output

```
Enter an integer: 7
7 is an odd integer.
```

When the user enters 7, the test expression `number%2==0` is evaluated to false. Hence, the statement inside the body of `else` is executed.

## (B) for loop

# The For Loop

The **for** loop has the following syntax:

```
for (statement 1; statement 2; statement 3) {
    // code block to be executed
}
```

**Statement 1** is executed (one time) before the execution of the code block.

**Statement 2** defines the condition for executing the code block.

**Statement 3** is executed (every time) after the code block has been executed.

### Example

```
for (i = 0; i < 5; i++) {
    text += "The number is " + i + "<br>";
}
```

```
}
```

From the example above, you can read:

Statement 1 sets a variable before the loop starts (var i = 0).

Statement 2 defines the condition for the loop to run (i must be less than 5).

Statement 3 increases a value (i++) each time the code block in the loop has been executed.

## Statement 1

Normally you will use statement 1 to initialize the variable used in the loop (i = 0).

This is not always the case, JavaScript doesn't care. Statement 1 is optional.

You can initiate many values in statement 1 (separated by comma):

### Example

```
for (i = 0, len = cars.length, text = ""; i < len; i++) {  
  text += cars[i] + "<br>";  
}
```

And you can omit statement 1 (like when your values are set before the loop starts):

### Example

```
var i = 2;  
var len = cars.length;  
var text = "";  
for (; i < len; i++) {
```

```
text += cars[i] + "<br>";  
}
```

## Statement 2

Often statement 2 is used to evaluate the condition of the initial variable.

This is not always the case, JavaScript doesn't care. Statement 2 is also optional.

If statement 2 returns true, the loop will start over again, if it returns false, the loop will end.

If you omit statement 2, you must provide a **break** inside the loop. Otherwise the loop will never end. This will crash your browser. Read about breaks in a later chapter of this tutorial.

## Statement 3

Often statement 3 increments the value of the initial variable.

This is not always the case, JavaScript doesn't care, and statement 3 is optional.

Statement 3 can do anything like negative increment (i--), positive increment (i = i + 15), or anything else.

Statement 3 can also be omitted (like when you increment your values inside the loop):

Example

```
var i = 0;  
var len = cars.length;  
for (; i < len; ) {  
  text += cars[i] + "<br>";  
  i++;  
}
```

## (C) while loop

The **while** loop loops through a block of code as long as a specified condition is true.

Syntax

```
while (condition) {  
  
}
```

Example

In the following example, the code in the loop will run, over and over again, as long as a variable (i) is less than 10:

Example

```
while (i < 10) {  
  text += "The number is " + i;  
  i++;  
}
```

The Do/While Loop

The **do/while** loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax

```
do {  
  // code block to be executed  
}  
while (condition);
```

## Example

The example below uses a **do/while** loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

### Example

```
do {  
  text += "The number is " + i;  
  i++;  
}  
while (i < 10);
```

Do not forget to increase the variable used in the condition, otherwise the loop will never end!

## Comparing For and While

If you have read the previous chapter, about the for loop, you will discover that a while loop is much the same as a for loop, with statement 1 and statement 3 omitted.

The loop in this example uses a **for** loop to collect the car names from the cars array:

### Example

```
var cars = ["BMW", "Volvo", "Saab", "Ford"];  
var i = 0;  
var text = "";  
  
for (;cars[i];) {  
  text += cars[i] + "<br>";  
  i++;  
}
```

The loop in this example uses a **while** loop to collect the car names from the cars array:

#### Example

```
var cars = ["BMW", "Volvo", "Saab", "Ford"];
var i = 0;
var text = "";

while (cars[i]) {
  text += cars[i] + "<br>";
  i++;
}
```

Exercise:

Create a loop that runs as long as **i** is less than 10.

```
var i = 0;
 (i  10) {
  console.log(i);
  i++;
}
```

## **D) do-while loop**

### **The Do/While Loop**

The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax

```
do {
```



```
}  
while (condition);
```

### Example

The example below uses a do/while loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

#### Example

```
do {  
  text += "The number is " + i;  
  i++;  
}  
while (i < 10);
```

Do not forget to increase the variable used in the condition, otherwise the loop will never end!

### Comparing For and While

If you have read the previous chapter, about the for loop, you will discover that a while loop is much the same as a for loop, with statement 1 and statement 3 omitted.

The loop in this example uses a for loop to collect the car names from the cars array:

#### Example

```
var cars = ["BMW", "Volvo", "Saab", "Ford"];  
var i = 0;  
var text = "";  
  
for (;cars[i];) {  
  text += cars[i] + "<br>";  
}
```

```
i++;  
}
```

The loop in this example uses a while loop to collect the car names from the cars array:

#### Example

```
var cars = ["BMW", "Volvo", "Saab", "Ford"];  
var i = 0;  
var text = "";  
  
while (cars[i]) {  
  text += cars[i] + "<br>";  
  i++;  
}
```

#### Test Yourself With Exercises

Exercise:

Create a loop that runs as long as i is less than 10.

```
var i = 0;  
 (i  10) {  
  console.log(i);  
  i++  
}
```

