CCA-101: Fundamentals of IT & Programming

Assignment - 2

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

Ans. Difficulty Level : Medium

- LastBoth **High level language** and **low level language** are the programming languages's types.

The main difference between **high level language** and **low level language** is that, Programmers can easily understand or interpret or compile the high level language in comparison of machine. On the other hand, Machine can easily understand the low level language in comparison of human beings.
Examples of high level languages are C, C++, Java, Python, etc.
Let's see the difference between high level and low level languages:

S.NO	High Level Language	Low Level Language
1.	It is programmer friendly language.	It is a machine friendly language.
2.	High level language is less memory efficient.	Low level language is high memory efficient.
3.	It is easy to understand.	It is tough to understand.
4.	It is simple to debug.	It is complex to debug comparatively.
5.	It is simple to maintain.	It is complex to maintain comparatively.
6.	It is portable.	It is non-portable.
7.	It can run on any platform.	It is machine-dependent.
8.	It needs compiler or interpreter for translation.	It needs assembler for translation.
9.	It is used widely for programming.	It is not commonly used now-a-days in programming.

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

Data types in C programming language

One of the most important concept in programming is the **variable**. The variable can be seen as the "place" to store "things" as: numerical values, characters, text strings, memory addresses, etc. There are two main concepts regarding variables. The fist concept is the **declaration** of the variable, which basically means setting its **data type**. The second concept is the **definition** of the variable, which means setting its **content**. In the C programming language every variable used in the source code needs to be declared by setting its **data type**. By assigning a certain data type to a variable we define several aspects linked to the variable:

- the memory size to be allocated to store the content of the variable
- the types of operations that can be performed on the variable
- the restrictions which are applied in terms of operations

By the end of this tutorial the reader will know:

- what is the significance of a data type
- how to declare and define a variable
- which are the properties of the standard data types
- what is integer overflow

In the C programming language a variable is **declared** as:

```
unsigned int uiVar;
```

where:

`uiVar` – is the name of the variable
`unsigned` – keyword which defines that our variable is always positive (without the sign "-")
`int` – keyword which defines that our variable is an integer and has 4 bytes of memory allocated (for a 32-bit compiler)

The **definition** of the variable can be done in another line, but only after the declaration:

```
uiVar = 25;
```

Variables can be **declared and defined** in the same instruction:

```
unsigned int uiVar = 25;
```

The main data types in C programming language are:

- **integer (fixed-point)**
- **floating-point**
- **void**

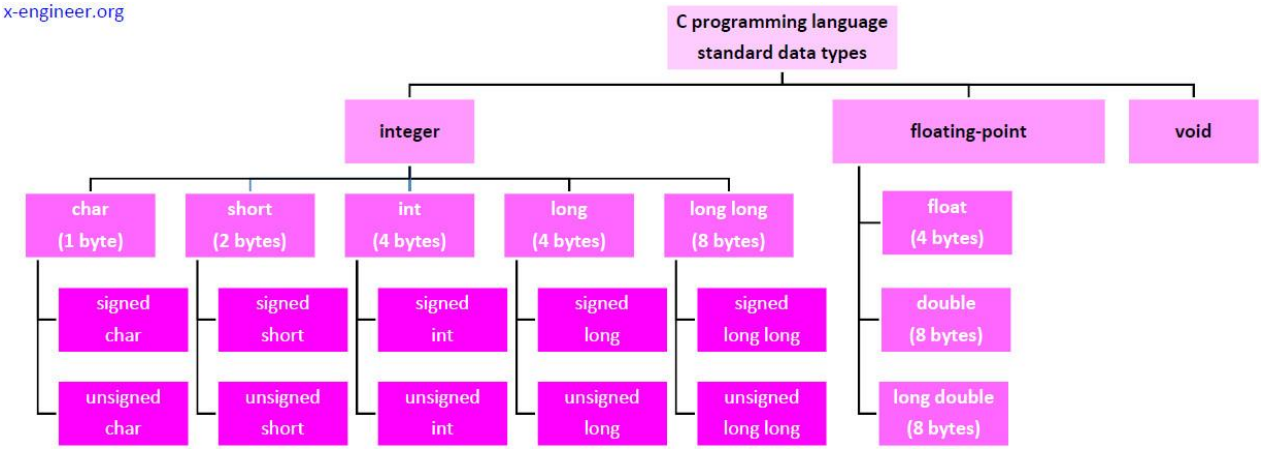


Image: C programming language – standard data types

Note: Depending on the type of the compiler (16-bit or 32-bit), the size and range of `int`, `short` and `long` are different.
The **void** type has no value and unknown size. It's mainly used for function definition as return type or argument of the function.

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

Divide the numbers

$x=205\cdot2+30-5$

$x=4\cdot2+30-5$

2

Multiply the numbers

$x=4\cdot2+30-5$

$x=8+30-5$

3

Add the numbers

$x=8+30-5$

$x=33$

Solution
$x=33$

Q4. Describe the syntax of the following statements

a) If – else statement b) for loop c) while loop d) do-while loop

Ans.aSyntax

The **if** statement specifies a block of code to be executed if a condition is true:

```
if (condition) {
  // block of code to be executed if the condition is true
}
```

The **else** statement specifies a block of code to be executed if the condition is false:

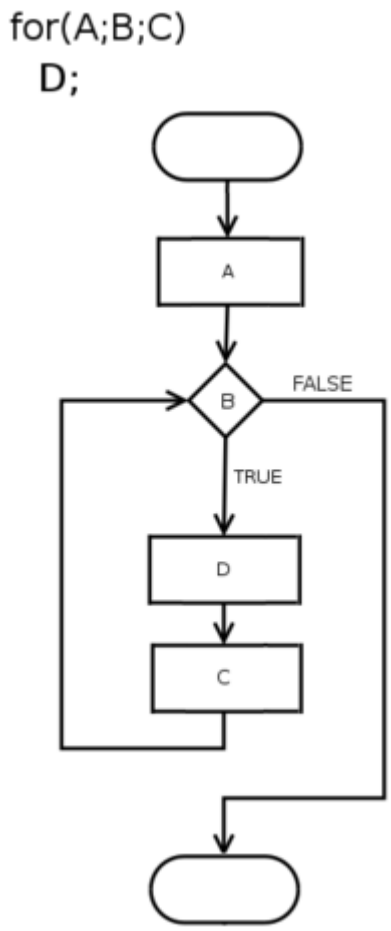
```
if (condition) {
  // block of code to be executed if the condition is true
} else {
  // block of code to be executed if the condition is false
}
```

The **else if** statement specifies a new condition if the first condition is false:

```
if (condition1) {
  // block of code to be executed if condition1 is true
} else if (condition2) {
  // block of code to be executed if the condition1 is false and
condition2 is true
} else {
  // block of code to be executed if the condition1 is false and
condition2 is false
}
```

Ans.bFor loop

From Wikipedia, the free encyclopedia
Jump to navigationJump to search



For loop flow diagram

Loop constructs

- Do while loop
- While loop
- For loop
- Foreach loop
- Infinite loop
- Control flow

- v
- t
- e

In computer science, a **for-loop** (or simply **for loop**) is a control flow statement for specifying iteration, which allows code to be executed repeatedly. Various keywords are used to specify this statement: descendants of ALGOL use "for", while descendants of Fortran use "do". There are other possibilities, for example COBOL which uses "PERFORM VARYING".

A for-loop has two parts: a header specifying the iteration, and a body which is executed once per iteration. The header often declares an explicit loop counter or loop variable, which allows the body to know which iteration is being executed. For-loops are typically used when the number of iterations is known before entering the loop. For-loops can be thought of as shorthands for while-loops which increment and test a loop variable.

The name *for-loop* comes from the word for, which is used as the d in makeyworny programming languages to introduce a for-loop. The term in English dates to ALGOL 58 and was popularized in the influential later ALGOL 60; it is the direct translation of the earlier German **für**, used in *Superplan* (1949–1951) by Heinz Rutishauser, who also was involved in defining ALGOL 58 and ALGOL 60. The loop body is executed "for" the given values of the loop variable, though this is more explicit in the ALGOL version of the statement, in which a list of possible values and/or increments can be specified.

In FORTRAN and PL/I, the keyword *DO* is used for the same thing and it is called a **do-loop**; this is different from a do-while loop.

Ans.c The While Loop

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

Syntax

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

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:

Ans.d How while loop works?

- The `while` loop evaluates the test expression inside the parenthesis `()`.
- If the test expression is true, statements inside the body of `while` loop are executed. Then, the test expression is evaluated again.
- The process goes on until the test expression is evaluated to false.
- If the test expression is false, the loop terminates (ends).

To learn more about test expression (when the test expression is evaluated to true and false), check out relational and logical operators.

Q5. Find the output of the following program segments

Ans.