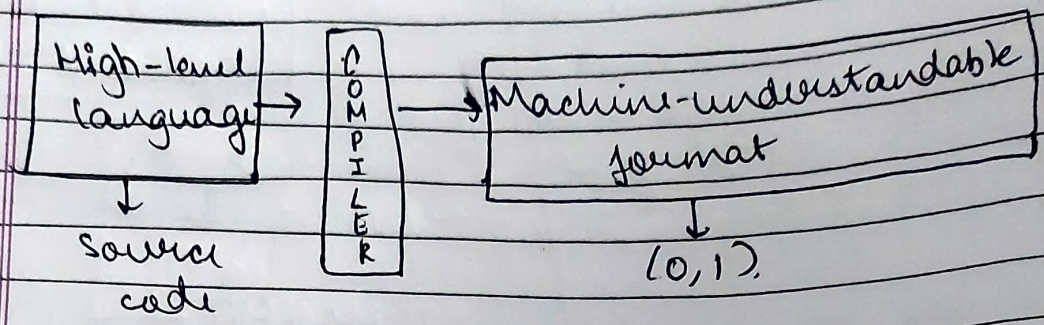
Lecture-3
Create your first program

* Compilation Process-



* Programming Languages-

- A language using which, we can ~~construct~~ instruct the computer to carry out real life tasks and computations is called as programming language. It acts as a language in which we could easily express our thoughts to the machine.

- It has a fixed set of rules according to which programs could be written in it. These programs are then converted into a machine understandable language through compiler

- Every language ka compiler has its own compiler/interpreter.
- Once a program is compiled and linked, its executable is created and computer can run our program now.

* IDE (Integrated Development Environment)-

- Code Editor
- It provides environment to write the code and also provides various features

* Create your first program-

```
int main(){
}
```

1. int main() {- it is a starting point of execution of a code

```
int main(){
    cout<<"Namaste";
}
```

2. cout - we can print anything in console or output using this functionality

```
#include<iostream>
int main(){
    cout<<"Namaste";
}
```

3. #include<iostream>- Through this we import a file in which implementation of our keywords for input and output are stored such as cin and cout.

```
#include<iostream>
using namespace std;

int main(){
    cout<<"Namaste";
}
```

4. Namespace - It is a region or part or area of a code on basis which implementation of input/output differs. In this scope of identifiers is defined.

5. std → standard nameplace.

whatever code is written within these brackets belongs to int main() function

```
#include<iostream>
using namespace std;

int main(){
    cout<<"Namaste";
    cout<<endl;
}
```

6. endl - when we print it, we end the ongoing line and bring the cursor to next line. We can also use '\n' for the same function

* Taking input-

1. Comment - If we want some part of code not to be executed, we can simply add "//" in front of a single line and "/*" in beginning of code, "*/" in end of code for multiline.

2. For taking input, first we need to select a variable which should be valid in which we can store the address of the input and also specify its data type, then we can simply take input using cin as

```
int a;
cin >> a;
```

NOTE: "<<" is known as an insertion operator

Single inverted commas denote character

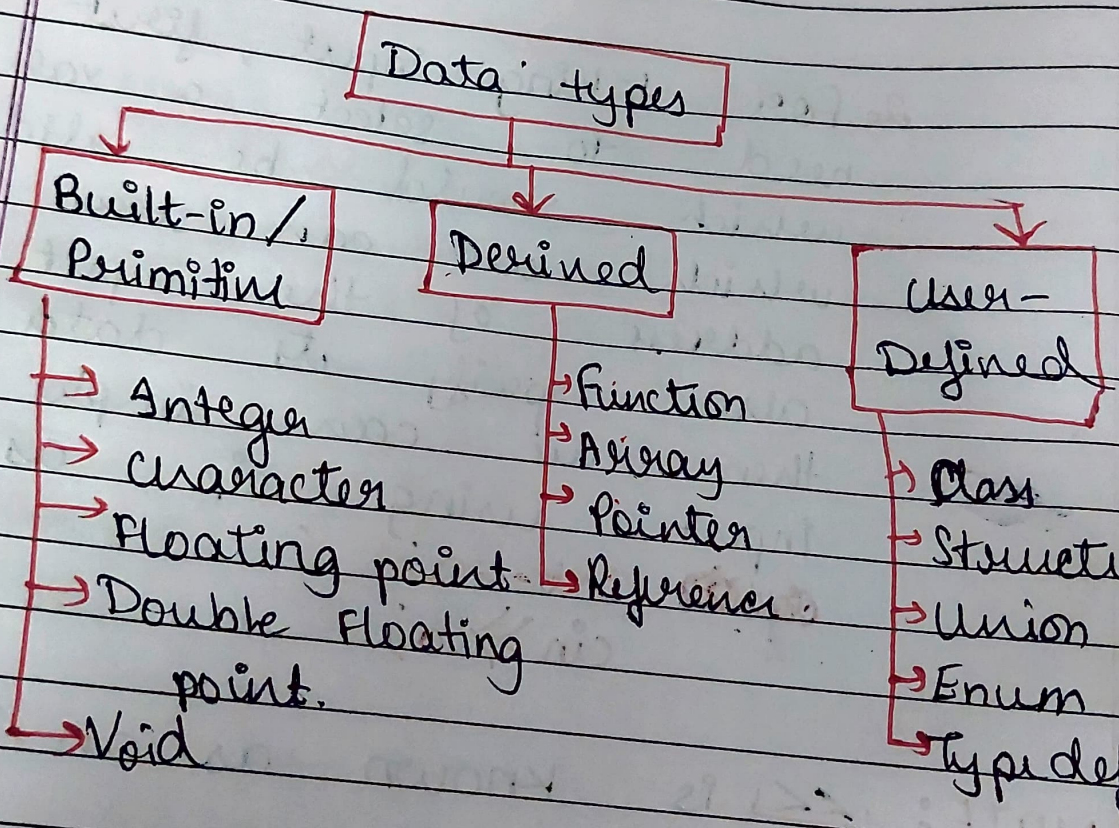
Double inverted commas denote string.

Shortcut for comment out a line is ctrl+/.

* Variables and Datatypes-

It is a memory location with some name in order to store some form of data and retrieve whenever required.

Datatypes are used to tell the variables the type of data they can store



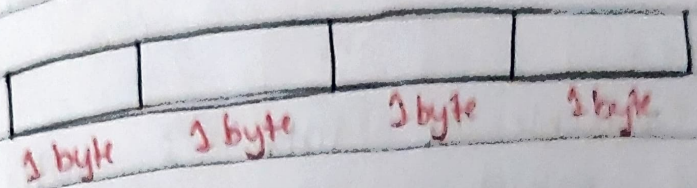
Example → int sum = 12

datatype ← int

sum → variable

12 → value

* int → Takes upto 4 byte which means 32 bits. (4x8).



It can take only 2 bits, that are 0 or 1.
So, we can have 2^{32} combinations for int data type

* char → Takes upto 1 byte which means 8 bits and 2^8 combinations are possible. Char stores character in form of bits. We convert character to its ASCII value and which it is further converted in 0s and 1s

* Bool or Boolean → It has only T (true) and F (false). T is represented by 1 and F is represented by 0. It takes 1 byte which means 8 bits. ~~because~~ Or 1 bit is sufficient for bool but 1 byte is minimum memory to be taken, therefore it takes 1 byte

* float - It takes 4 bytes which means 32 bits.

* double - It takes 8 bytes which means 64 bits. It gives more precised value since it takes more memory than float.

* Variable Naming Convention

Rules are-

1. Variable name must start with a letter or underscore character.
2. A variable name cannot be started with a digit.
3. A variable name can only contain alpha-numeric and underscores. (a-z, A-Z, 0-9, _)

* size of (datatype) - It gives the size of given data type in byte.

Eg: size of (int) = 4 bytes

* How data is stored?

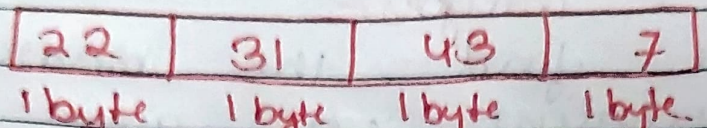
1. Positive Integers:-
We find the binary equivalent of given character in the memory and the right most bits of the leftover ~~not~~ bits are 0.

NOTE: +ve number → first bit always 0.
-ve number → first bit always 1.

2. Negative integers-
We find the 2's complement and store it in memory.
For finding the 2's complement we have following two steps-
1. Find 1's complement- just write the opposite digits such as 1's complement for 0101100 will be 1010011.
2. Then just add 1 to the 1's complement.

* Steps for storing negative integer-
1. Ignore the negative sign
2. Binary equivalent.
3. 2's complement.

* Interesting Problem-



How much byte or how much bits we will read will depend on datatype defined.

Ex, datatype defined is int then we will read 4 bytes

* Signed vs Unsigned data-

signed → +ve, -ve or 0.
unsigned → +ve or 0.

Generic for n bits.

1. signed → $(-2^{n-1} \rightarrow 2^{n-1}-1)$ range
2. unsigned → $(0 \rightarrow 2^{n}-1)$ range

* Type Casting -

* It refers to the conversion of one data type to another in a program.
* It can be done in two ways
  - automatically by compiler
  - manually by programmer
* It is also known as type conversion.

* Implicit type casting or Implicit type conversion -

1. It is known as the automatic type casting
2. It automatically converts one data type to another without any external intervention such as programmer through compiler.
3. All data type is automatically converted to largest data type without losing any information
4. It can only apply in a program if both variables are compatible with each other.

* Explicit type casting or Explicit type conversion-

1. Also known as manual type casting
2. Data type is one data type is converted to other manually by programmer according to the requirement.
3. Does not require checking the compatibility of the variables.
4. Upgrade or downgrade of data type from one variable to another can be done.

* Operators-

1. Arithmatic.

% → Modulo
+ → Addition
− → Subtraction
* → Multiplication
/ → Division

NOTE: int = int, float = float, int = float
int int float

Bigger data type (greater memory) will dominate.

2. Relational

>, <, >=, <=, !=, == Equal to

!= not equal to

It will give boolean output, i.e., 0 or 1.

3. Assignment

Assigning a value to variable (=).

4. Logical

&&, ||, !

And Or Not

used in case of multiple conditions

&& → If any one condition is false, then ~~every~~ output is false.
Only true when all the conditions are true

|| → True if any one condition is true.
False only when all conditions are false.

5. Bitwise