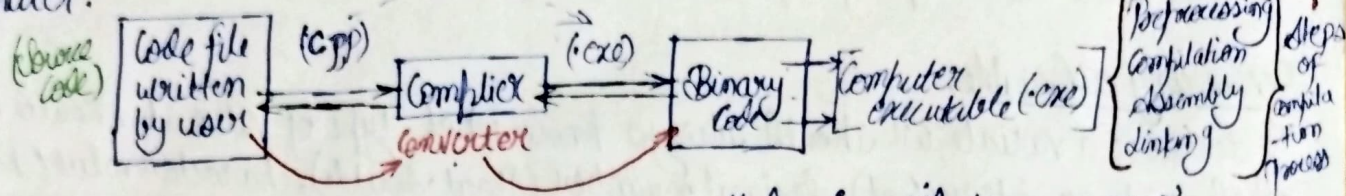
● Code Basics (Getting started)

A user writes a code to perform some task to computer, but computer cannot understand the language, computer can only understand binary, so there is a compiler in our system that converts our code into binary format which is readable by computer.



Compiler also tells us about errors that are called as 'compile time errors' and there are one more error type called as 'run time errors' which cannot catch by compiler.

ex [We give a+b in place of a*b] [compiler will compile it but user don't want it.]

C++ code begins with the inclusion of header files. There are many header files available in the C++ programming.

Header files ⇒ The names of program elements such as variables, functions, classes and so on must be declared before they can be used.

For ex, we can't just write x=42 without first declaring variable 'x' as: `int x=42;`

This declaration tells the compiler whether the element is an int, a double, a float, a function or a class. Similarly, header files allows us to put declaration in one location and then import them whenever we need them.

* To declare a header file, we use #include directive in every .cpp file.
  {• # operator is known as Macros.}

Syntax:

```
#include<iostream>
using namespace std;
int main(){
}
return 0;
```

Starting point of function

Scope of this function

* It shows or tells to OS, the successful execution of program (Symbol of successful execution)

- iostream stands for Input/Output stream, meaning this header file is necessary if we want to take input through the user or print output to the screen.
  header file contains :-
  - cin : used to take input
  - cout : used to take output.
- namespace defines which I/O form is to be used.
- semicolon (;) is used for terminating a C++ statement

- main() function: (int main()) → int: This is the return type of the function.
- main: This is the portion of any C++ code inside which all commands are written and gets executed. →
- { }: all the code written inside the curly braces is said to be in one block, also known as scope of a particular function.

Declaring a Variable:

To declare a variable, we should always know what type of value it should hold. whether it's an integer (int), decimal number (float, double), character value (char). In general variable is declared as:

Datatype variablename = value

- Variable is a container that contain data value of any type.

Datatypes → It defines ~~user~~ about the data. It is for memory management.

(type of data & storage or size of data)

(i) bool – 1 byte – Boolean values (true (1) or false (0))
(ii) char – 1 byte – Character value (including special characters)
(iii) short – 2 byte – Contains integer value but with smaller size
(iv) int – 4 byte – Integer value
(v) long – 8 byte – Contains integer with large size
(vi) float – 4 byte
(vii) double – 8 byte
(vi) & (vii) – contains decimal number

- datatype defines two things
  - (i) storage
  - (ii) tells what a user giving, its type

int age;
(Declaration of variable)

ex. int a = 5;
(definition & initialization)

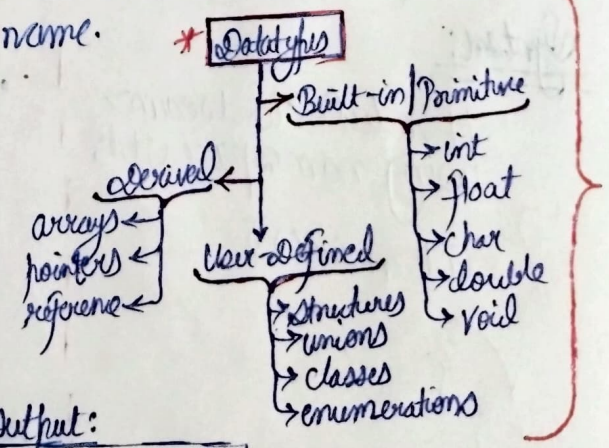
a = 101;
(manipulation) (updation)

- String → It is a type of variable but it contains collection of characters or words

int a = "Hello World";

Rules for variables names:

- Can't begin with a number.
- Spaces and special character except underscore (_) are not allowed.
- C++ keywords must not be used as a variable name.
- C++ is case sensitive.



First Program:

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

int main(){
    cout<<"Hello World";
}
```

Output:

Hello World

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

int main() {
    cout << "Hello" << endl;
    cout << "Hello World" << "\n";
}
```

Output:

```
Hello
Hello World
```

Line seperator ⇒ for seperating different lines in C++, we use endl or "\n".

Some Special characters:

- \t : tab character
- \b : backspace character
- \" : treats this symbol as a part of string rather than as string terminator
- \\ : includes a \ in string

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

int main() {
    cout << "career" << "labs";
}
```

Output:

```
careerlabs
```

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

int main() {
    int a = 10;
    int b = 20;
    int c = a + b;
    cout << c << endl;
}
```

Output:

```
30
```

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

int main() {
    int a = 20;
    float f = 2.5;
    size = sizeof(f);
    cout << size << endl;
}
```

[sizeof gives the size of any data]

Output:

```
2
```

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

int main() {
    int P, R, T, S;
    P = 20, R = 30, T = 10;
    S = (P * R * T) / 100
    cout << S << endl;
}
```

Output:

```
60
```

---

*
```
bool a = true
cout << a
```

Output : 1

• Compiler assigns a garbage value to an uninitialized local variable in C++.

Taking input:

To take input from user, we use 'cin' statement

```
#include<iostream>
using namespace std;
int main(){
    int a, b, sum;
    cin>>a>>b;
    sum=a+b;
    cout<<sum<<endl;
}
```

Input:

```
2 50
```

Output:

```
52
```

```
#include<iostream>
using namespace std;
int main() {
    int a;
    double d;
    char c;
    cin>>a>>d>>c;
    cout<<a<<d<<c;
}
```

Input:

```
2 10.1 D
```

Output:

```
210.1D
```

Operators in C++:

The symbols that are used in C++, to perform an operation called as operators.

(i) Arithmetic Operator ⇒ These are used in mathematical operations.

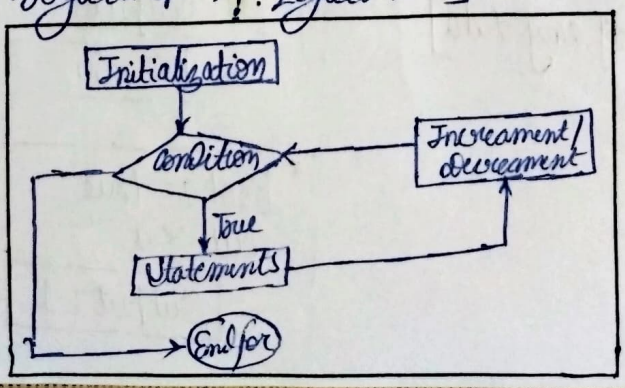
[+: add two operands ; −; *; /; %]

(ii) Relational Operator ⇒ These specify the relation b/w two variables by comparing them.

[==; !=; >; <; <=; >=] → (gives a boolean value)

(iii) Logical Operator ⇒ C++ supports 3 types of logical. The result of these operator is a boolean value. i.e. True (Bitwise '1') or False (Bitwise '0')

[&& : Logical AND ; || : Logical OR ; ! : Logical NOT]



(Arithmetic Operator)

```
#include <iostream>
using namespace std;
int main() {
    int f;
    cout << "Enter the fah Value" << endl;
    cin >> f;
    int c = (5.0/9) * (f-32);
    cout << c << endl;
    cout << (5.0/9) * (f-32) << endl;
    cout << (5/9) * (f-32) << endl;
}
```

→ [5.0 because int/int = int, then it gives 0]

→ [here we can can see so-thats why, above we gave a decimal no. either it is numerator or denominator]

```
Enter the fah Value
100
37
37.7778
0
```

(Relational Operator)

```
#include <iostream>
using namespace std;
int main() { int a, b;
cout << "Enter the values" << endl;
cin >> a >> b;

bool isEqual = (a == b);
bool isAGreater = (a > b);
bool isAlesser = (a < b);
cout << "Are They Equal" << isEqual << endl;
cout << "Is A Larger" << isAGreater << endl;
cout << "Is A lesser" << isAlesser << endl;
}
```

[print True if both are True]

```
bool third = isEqual && isAGreater;
bool fourth = isEqual || isAGreater;
cout << "Not Equal" << !isEqual << endl;
```

[|| → print True if any one condition is true]

Output:

```
Enter the values
45 445
Are They Equal 0
Is A Larger 0
Is Alesser 1
```

```
#include<iostream>
using namespace std;
int main(){
double a=6/4;
int b=6/4;
double c=a+b;
cout<<c;
}
```

Output

2

```
#include<iostream>
using namespace std;
int main(){
    double a=55.5;
    int b=55;
    a=a%10;
    b=b%10;
    cout<<a<<" "<<b;
}
```

Syntax error

[% can't used with double or float]

```
#include<iostream>
using namespace std;
int main(){
int var1=5;
int var2=6;
cout<<(var1>var2);
}
```

0

How is Data stored:

[** 1. In memory '1 byte' is the smallest "addressable space" 2. That is why bool also take 1 byte instead it can use 1 bit]

- For integers ⇒

The most commonly used is a signed 32-bit integer type. When you stored an integer, its corresponding binary value is stored. There is seperate way of storing +ve and -ve numbers. for +ve numbers the integral value is simply converted into binary value while for -ve number their 2's complement form is stored.

for -ve numbers ⇒

Computers use 2's complement in representing signed integers because,

(i) There is only one representation for the number zero in 2's complement, instead of two representation in sign-magnitude and 1's complement.

(ii) +ve and -ve numbers can be located together in addition and substraction

Substraction can be carried out using the addition logic.

for example: int i = -4

Steps to calculate Two's complement of -4 as follows:

(i) Take binary equivalent of +ve value (4 in this case)

32 bit system (1 cell = 1 byte)

0000 0000 0000 0000 0000 0000 0000 0100

(ii) Then write's 1's complement of binary representation by inverting the bits

1111 1111 1111 1111 1111 1111 1111 1011

(iii) Find 2's compliment by adding 1 to the corresponding 1's complement

1111 1111 1111 1111 1111 1111 1111 1011
+ 0000 0000 0000 0000 0000 0000 0000 0001
1111 1111 1111 1111 1111 1111 1111 1100

Thus, this is representation of -4.

{ 1 bit → 0/1 → 2 ④ {0/1} ⑤
2 bit → 4
{ 1 byte → 8 bits → 2^8 (0–255) (256) }
{ 32 bits → 2^{32} (0–$2^{32}-1$) { numbers can two store here } }

{ int
→ signed : Range -2^{n-1} – $2^{n-1}-1$
→ unsigned : $1 - 2^n - 1$ }

- **For float and double variables:**

In C++, any value declared with a decimal point is by-default of type Double. If we want to assign a float value, then we must use 'f' or 'F' literal to specify that the current value is "float".

for example:

```
float var = 10.4f   // float value
Double val = 10.4   // double value
```

2.35 → Binary format

10.——

100.1101

1.00110 1 * 2^2 → exponent

Mantesa

- **For characters values:**

Every character has a unique integer value, which is called as ASCII value. As we know system only understand binary language and thus every thing has to be stored in the form of binaries. So, for every character there is a corresponding integer code - ASCII code and binary equivalent of this code is actually stored in memory when we try to store a character.

When we add int to char, we are basically adding two numbers one corresponding to int and other to ASCII code for the character.

```
#include<iostream>
using namespace std;
int main(){
    cout<<'a'+1;
}
```

98

* [ASCII of 'a' = 97]

```
#include<iostream>
using namespace std;
int main(){
    int i='C';
    cout<<i<<endl;
}
```

99

```
#include<iostream>
using namespace std;
int main(){
    char c=74;
    cout<<c;
    return 0;
}
```

J

```
#include<iostream>
using namespace std;
int main(){
    int a=10;
    char ch='a';
    ch=ch+a;
    cout<<ch<<endl;
}
```

k

```
#include<iostream>
using namespace std;
int main(){
    unsigned int a=-123;
    cout<<a<<endl;
}
```

4294967173

Conditional Operator (?:) / Ternary operator ⇒

Expression1 ? Expression2 : Expression3

↑ It is the condition to be evaluated, if the condition is True, then we will execute and return the result of Exp2, otherwise Exp3.

Pointer Operator ⇒

(a) & ⇒ It refers to the address (memory location) in which the operand is stored.

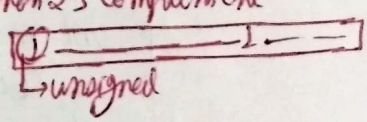
(b) * ⇒ It is a pointer operator

```
int main(){
    double d=53.69887;
    cout<<d<<endl;                    //53.6989
    cout<<setprecision(7)<<d<<endl;   //53.69887
}
```

* Here, when you o/p a double using cout, it often formatted to a certain no. of decimal places by default. This formatting is controlled by the precision setting of the o/p ostream. By default cout typically displays up to a certain no. of decimal places.
  To control the precision of floating point no. displayed by cout, you can use 'setprecision' manipulator from <iomanip> header.

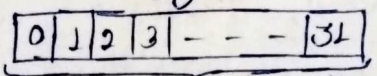
* We cannot cin true or false in boolean values or variables instead of this we have to give 0 or 1

Firstly
It will consider -123 as 123 then it will store it will convert in form of binary take, 1's compliment and then 2's compliment



{ int → 4 byte → 32 bits
1 byte → 8 bits }

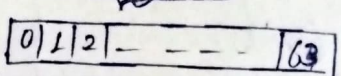
32 bit system



32 bits can handle by CPU at a single time

Vs

64 bit system



64 bits can be handled by CPU at a single

{ 1 bit ← 0/1 } (2¹) Combinations

0/1

0/1 0/1 ← 2 bits } (2²)

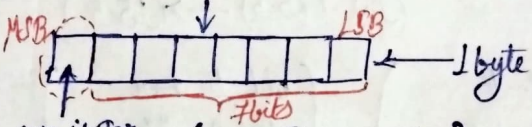
0/1 0/1 0/1 ← 3 bits } (2³)

[8 bits → (2⁸) → 256 → 0-255 (store values)]

[(32 bits) → (2³²) → (0-(2³²-1))]

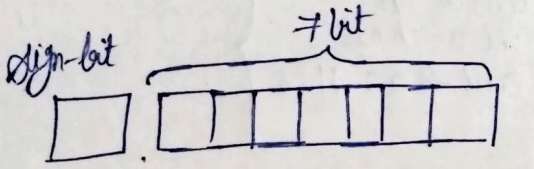
{ n bits → (2ⁿ-1) ← values can store }

(then are only +ve values)
(then what about -ve values)



use it for sign bit → { 0 ← for +ve value
1 ← for -ve value }

{ 1 bit ← for sign bit
7 bit ← (2⁷-1) → 128 → (0-127) }



(-ve) → (0-(-128))

{ 1 1111111 ← (-127)
1 0000001 ← (-1)
1 0000000 ← (-128)
sign bit }

(+ve) → (0-127)

{ 00000000 ← 0
0 1111111 ← 127
sign bit }

{-128 - 0 - 127}

1 bit → {-2⁷ - 0 - 2⁷-1}

(256 values)

* { 4 byte
(2³²) → -2³²⁻¹ - 0 - 2³²⁻¹ - 1
{-2³¹ - 0 - 2³¹-1} } for signed

* { 4 bit
2³² → 0 - 2³²-1 } for unsigned
*(no sign bit in consideration)

---

* cin.get() ← takes 1 character as i/p { char first, char last; }

first = cin.get(); ← J
cin.ignore(256, ' '); ← {ignore 256 characters or stops unless a space comes}
last = cin.get(); ← R
cout << first << last << endl; ← JR

{ >> Jai Roy ← consoule }
pick by cin

* cin.ignore(5, ' ')
* n custom delimeter
no. of characters to ignore (or) stops as it gets the same character

short → 2 bytes
(16 bits)
{Range: -32768 → 32767}

```
short a=32767;
cout << a; //32767
a=32768;
cout<<a; //-32768
a=32769;
cout<<a; //-32767
```

Not in the range of short

-32768 ... 32767 (+1) → reaches here in cyclic way

Typecasting :-

(1) Implicit Type Casting :- • Compiler automatically converts one data type to another during an operation

• This happens when you perform operations involving variable of different data types, and the compiler promotes one type to a larger type to maintain precision.

ex: int + float → float (because can store more (.5))
(10) + (5.5) → (15.5)

```
int to float
a=10; b=5.5;
float c=a+b;
cout<<c; //15.5
```

```
int to char
int a=97;
char ch=a;
cout<<ch; //a
```

```
char to int
char ch='A';
int a=ch+1;
cout<<a; //66
```

(2) Explicit Type Casting :- • Manual Type Conversion

• Allow you to specify the desired data type during an assignment.
• You use the casting operator (), containing the target data type.

```
ex: int num1=10;
    float num2=5.5;
    float result = num1 + (int)num2; //10 + 5
    cout<< result; //15
```

```
double to int
double pi=3.148;
int a=(int)pi;
cout<<a; //3
```

```
char to int
char ch='A'
int a=(int)ch;
cout<<a; //65;
```