PROGRAMMING FUNDAMENTALS

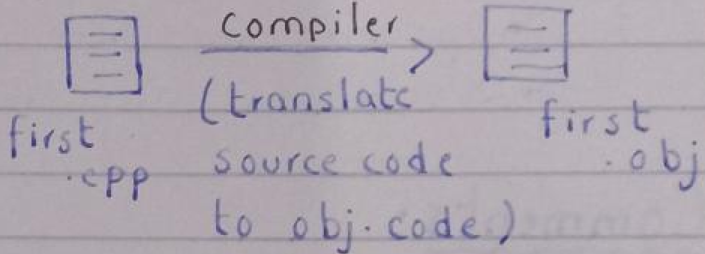
Let's explore technology together to live in the future

Programming Language

TYPES
- Low → Binary (0,1)
- Middle → Assembly (Mnemonics)
- High → C++, Java, Python.

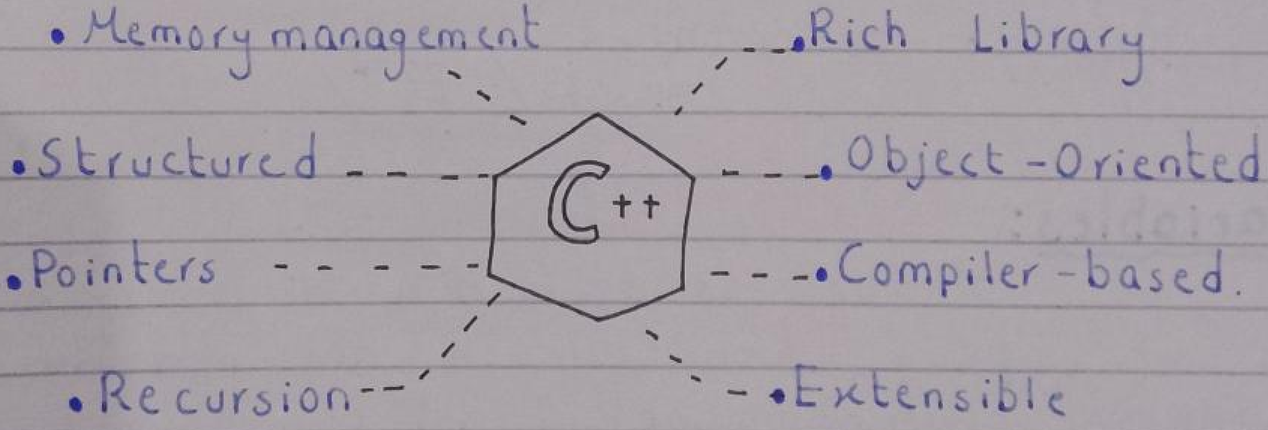
A Computer Program:

C → C++



• Object oriented programming.

Features of C++ Language:



=>Identifiers are defined as
that we declare in a program, in order to name a value, variable, function, array etc.
e.g int x = 10;

Rules
- → First charachter should be an alphabet.
- → Succeding charachters can be digit/letter
- → Special charachter not allowed except (_)
- → Should not be keywords.

Comments:

C++ Program comments are user-written statements that are included in C++ code for explanation purpose, they help reading source code.
=>Compiler ignore the comments.

Singleline '//' ; Multi line comments /* ≡ */

Variables:

Variable is defined as the reserved memory space which stores a value of defined datatype, value of variable is not constant, it allows changes. "Named Memory Location".

Rules
- → cannot start with number, digit.
- → can start with alphabet, symbol.
- → can't have spaces.

Each variable in C++ has a type, determines
- the size and layout of variable's memory.
- the range of values that can be stored within that memory

A simple variable definition consist of:
- a type specifier, followed by
- a list of one or more variable names separated by commas and ends with a semicolon.

e.g
```
int num1;
float num2, num3;
```

Variable decloration:

1) Assigning memory location / datatype to variable name in source code.

Data Type:
- numeric - integer type => int
- decimal - floating point => float
- alphabet / symbols - charachter => char
- true / false - Boolian => bool.

2) Assigning datatype and name.

$$\begin{tikzcd} \text{Data type} \arrow[d] & \text{variable name} \arrow[d] \\ \text{int} & x \end{tikzcd}$$

- Compiler will assign a memory location named as 'x' with '4 bytes'.

Variable Initialization:

Assigning a value to variable at the time of decleration is called initialization.

Methods → Datatype var.name = value;
→ Datatype var.name(value);
→ Datatype var.name {value};

Size of Operator:

=> Size of operator gives the size in number of bytes of any datatype or variable.

Syntax:

Size of (operand); $\text{operand} \begin{cases} \rightarrow \text{variable} \\ \rightarrow \text{constant} \end{cases}$

Compound Assignment Statement:

=> An assignment statement that assigns a value to many variables.

e.g A = B = 10;   • assign value 10 to A and B.

Compound Assignment Operators:

=> Compound assigment operators combine assignment operator with arithmetic operators.

variable to assign a value ↲   ↱ can be arithmetic op.

variable operator = expression;

↳ can be constant, variable, arithmetic expression.

Example :

```
int main ()
{
  int a=10;
  cout << "Value of a:"<<a<< endl,
   a += 5
  cout << "Value of a after a+=5 :"<<a<<endl;
   a -= 5
  cout << "Value of a after a-=5 :"<<a<<endl;
   a* = 2
  cout << "Value of a after a*=2:"<<a<<endl;
   a/=2
  cout << "Value of a after a/=2 :"<<a<<endl;
   a %=2
  cout <<"Value of a after a% =2:"<<a<<end

}
```

Limits.h header file :

```
#include <iostream>
#include <limits.h>
using namespace std;
int main ()
{
   cout<<"Minimum Range of Charachter Data
          Type:"<<CHAR-MIN<<endl;

   cout<<"Maximum Range of Charachter Data
          Type:"<<CHAR-MAX <<endl;
}
```

Expression:

A statement that evaluates to a value is called expression.

- Gives a single value.
- Consist of
  - operator is a symbol that performs some operation.
  - operand is value on which operator performs operation
- Can be constant, variable.

e.g A + B ; m/n ;

Operators:

Operators are symbols that are used to perform different operations

Types of operators:

- Arithmetic Operators +, -, *, / and %
- Relational Operators >, <, ==, >=, <=, !=
- Logical Operators && , || and !
- Assignment Operators =
- Increment and Decrement ++ , --
- Compound Assignment Operators
  +=, -=, *=, /= and %=

Compiler language

```
int main ()
{
  int a=10;
  int b=0;
   b = ++a
  cout <<"b";
```

1- increment
a = a+1
2- assignment
b = a

```
int main ()
{
 int a=10;
 int b=0;
  b=a++
 cout <<"b";
```

1- assigment
b=a
2- increment
a=a+1

Arithmetic Expression:

A type of expression that consists of constants, variables and arithmatic operators.

Example:

Suppose we have two variables A and B where A=10 , B=5.

```
#include <iostream>
using namespace std;
 int main()
{
   int a=10;
   int b =5;
   cout << "a+b=" <<a+b <<endl;
   cout <<" a-b=" <<a-b <<endl;
   cout <<"a /b=" << a/b <<endl;
   cout <<"a % b =" <<a %.b <<endl;
    return 0;
  }
```

Operator Precedence:

The order in which different types of operators in an expression are evaluated is known as operator precedence.

Operator	Precedence
!	Highest
*, /, %	
+, -	
<, <=, >, >=	
==, !=	
&&	
||	
+=, -=, *=, /=, %=	↓
=	Lowest

Increment Operators:

It is (++) a unary operator, works with single variable. A++ ; is equivalent to A = A + 1 ;

Prefix and Postfix Increments:

- No difference if 'alone' in statement :
  A++; and ++A; identical results.
- The value of expression (that uses ++/--) depends on the position of operator.

Expressions: If ++ is after variable as in a++, then increment takes place after expression is evaluated => Uses current value, increment.

$$\begin{tikzcd} & x = a \text{ value of expression is 'a'.} \\ x = a++ \arrow[ur] \arrow[r] & a = a+1 \text{ value of 'a' incremented by 1.} \end{tikzcd}$$

e.g

```
int n=2;
int result = 2*(n++);
cout << "result is" << result << endl;
```

Output:

```
result is 4
```

Expressions: If ++ is before variable as in ++a; increment takes place before expression is evaluated;
=> Increment variable first, then uses new value.

$$\begin{tikzcd} & a=a+1 \text{ value of a incremented by 1.} \\ x = ++a \arrow[ur] \arrow[r] & x = a \text{ value of expression is a after inc.} \end{tikzcd}$$

e.g

```
int n = 2;
int result = 2*(++n);
cout << "result is" << result << endl;
```

Output:

```
result is 6.
```

TYPE CASTING

The process of converting datatype of a value during execution.

1) Implicit type casting:

Conversion from smaller to bigger datatype. It is performed automatically by compiler.

- Arithmetic operations are performed between operands of same type, if don't same type, C++ will automatically convert one to be the type of other.

RULES

1) When using = operator, the type of expression on right will be converted to type of variable on left.

```
int A = 5 + 12.75;        // A = 17
```

2) If a real value is assigned to an integer variable, it is truncated (chopped off after decimal point, not rounded off).

```
int A = 12.9;        // A = 12
```

3) If an integer value is assigned to a real variable, it is promoted (conversio to a higher type) to real (decimal added)

```
double x = 7;        // x = 7.0
```

e.g

```
int main ()
{ int x = 20;
  char y = 'a';   //y implicity converted to int. ASC

  x = x+y;               // value of 'a' is 97.
  cout << "x = " << x ;
```

Explicit Type Casting: Conversion from bigger to smaller datatype. It is performed by programmer.

Syntax:
- static_cast <DataType> (Expression)
- (type) Expression

type: indicates datatype to which operand is to be converted.

Expression: indicates constant, variable, expression whose datatype is to be converted.

```
1) int A = 5 + static_cast <int>(12.75);
2) int A = 5 + int (12.75);
```

```
int main ()
{
  float a, b;
  int c;
  a = 10.3;
  b = 5.2;
              // c=(int)a % (int) b; (old style)
  c = static-cast <int>(a) % static-cast <int>(b);
  cout <<"Result is "<< c;
}
```

A constant expression is an expr
whose value cannot change and that can be evaluated at compile time.

```
#include <iostream>
#define PI 3.1415
int main ()
{
    float r= 32.12;
    cout << " Area of circle = "<< PI *(r*r);

}
```

=> Decleration of constant at compile time:

```
#include <iostream>
#define PI 3.1415
int main()
{

    float r=32.12;
    const float PI =3.1415;

    cout <<"Area of circle = " << PI*(r*r);

}
```

Input:

Cin >> → extraction operator
↓
Console input