What is C++

- C++ is known to be a very powerful computer programming language.
- C++ is a general-purpose, case-sensitive, object oriented programming language.
- High-level program language / Mid-level.
- Bjarne Stroustroup starting in 1979 / 1980 at Bell Labs.
- Portable program language

Usages of C++

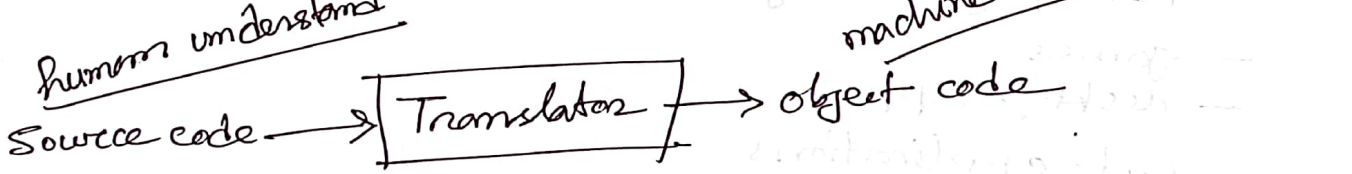
- develop game engines
- games
- desktop apps
- art applications
- music players
- write device drivers
- software

A program is a set of instructions for a computer to execute specific tasks.

A programming language is a formal language that specifies a set of instructions for a computer to execute specific tasks.

Types of programming language

- procedural - FORTRAN, COBOL, ALGOL, BASIC, Pascal, C
- Functional - Haskell, Lisp, clojure, Scala, F#
- object oriented - C++, C#, Java, JavaScript, Python, R
- Scripting - JavaScript, PHP, Perl, R, Python, Ruby, HTML



Translator/ programming language processor

Translator
- Compiler : source code into machine code (c, c++) (execute at a time)
- Interpreter : directly executes without convert machine (python, php, JS) (line by line)
- Assembler : Assembly language into machine

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

int main()
{
  cout << "gk";

  return 0;
}
```

means we can use names for objects and variables from the standard library.

Escape sequence / Backslash

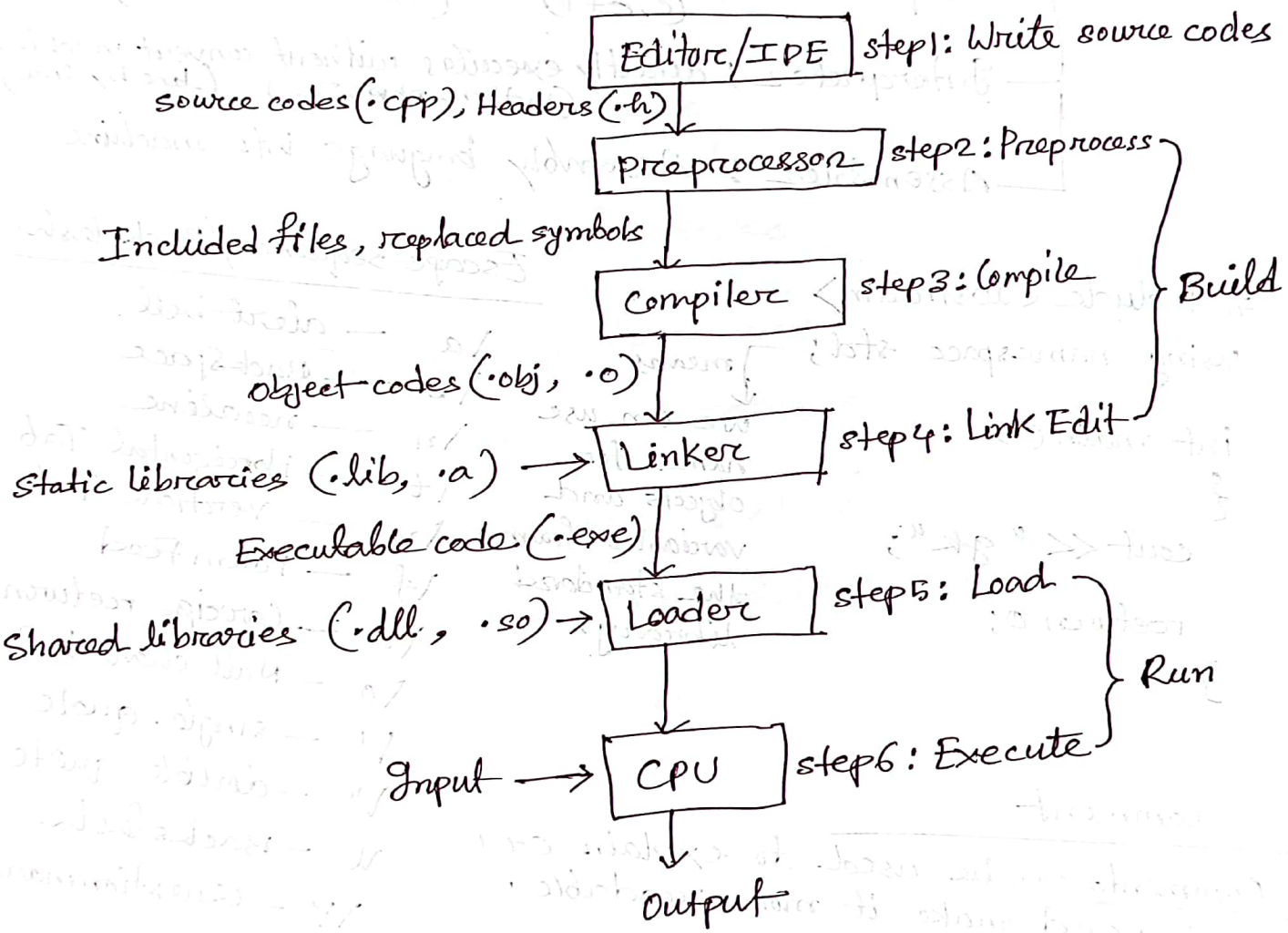
- \a — alert bell
- \b — BackSpace
- \n — newline
- \t — Horizontal Tab
- \v — vertical Tab
- \f — Form Feed
- \r — Carriage return
- \0 — Null character
- \' — single quote
- \" — double quote
- \\ — Backslash
- \? — Questionmark

Comment

Comments can be used to explain C++ codes and make it more readable.

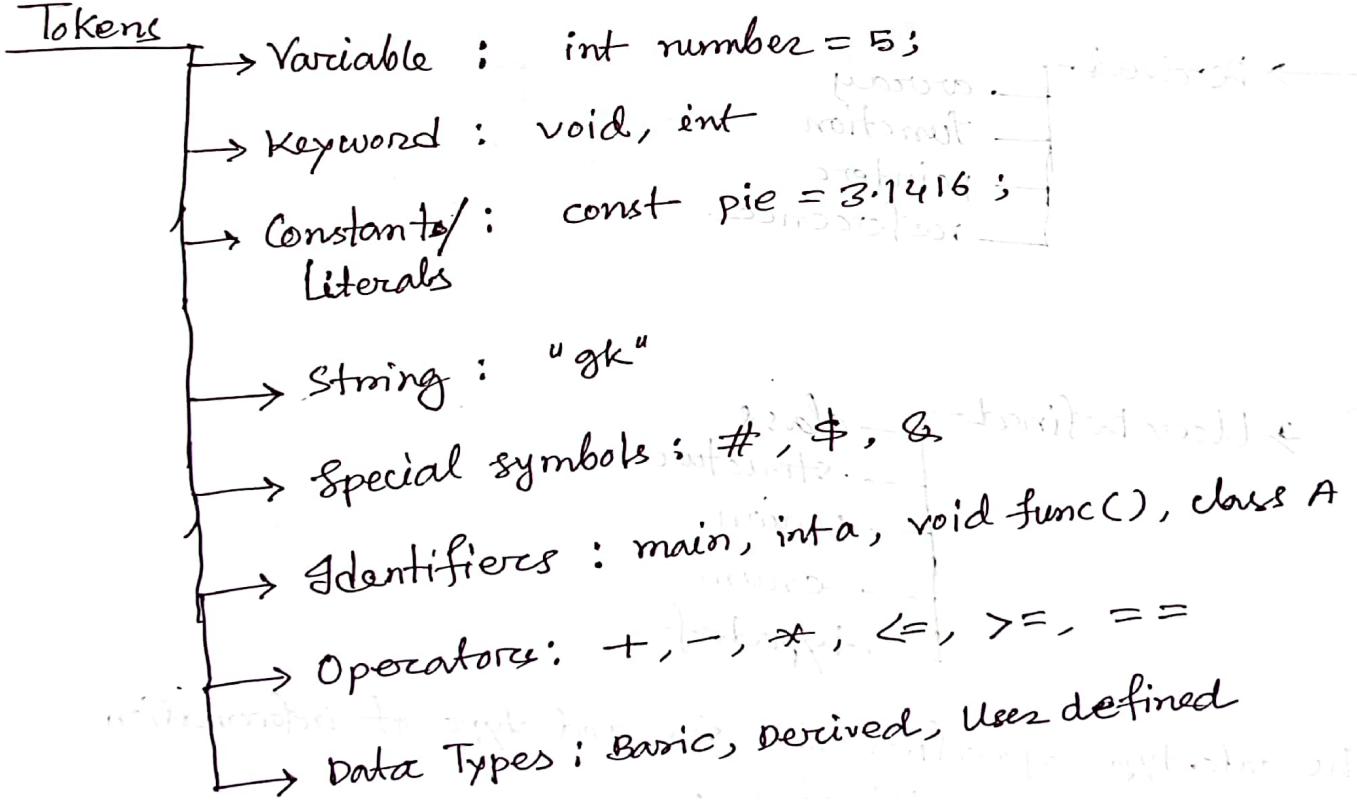
Types
- single line : //
- multiple line : /* ..... */

Compilation & Execution process of C++ program

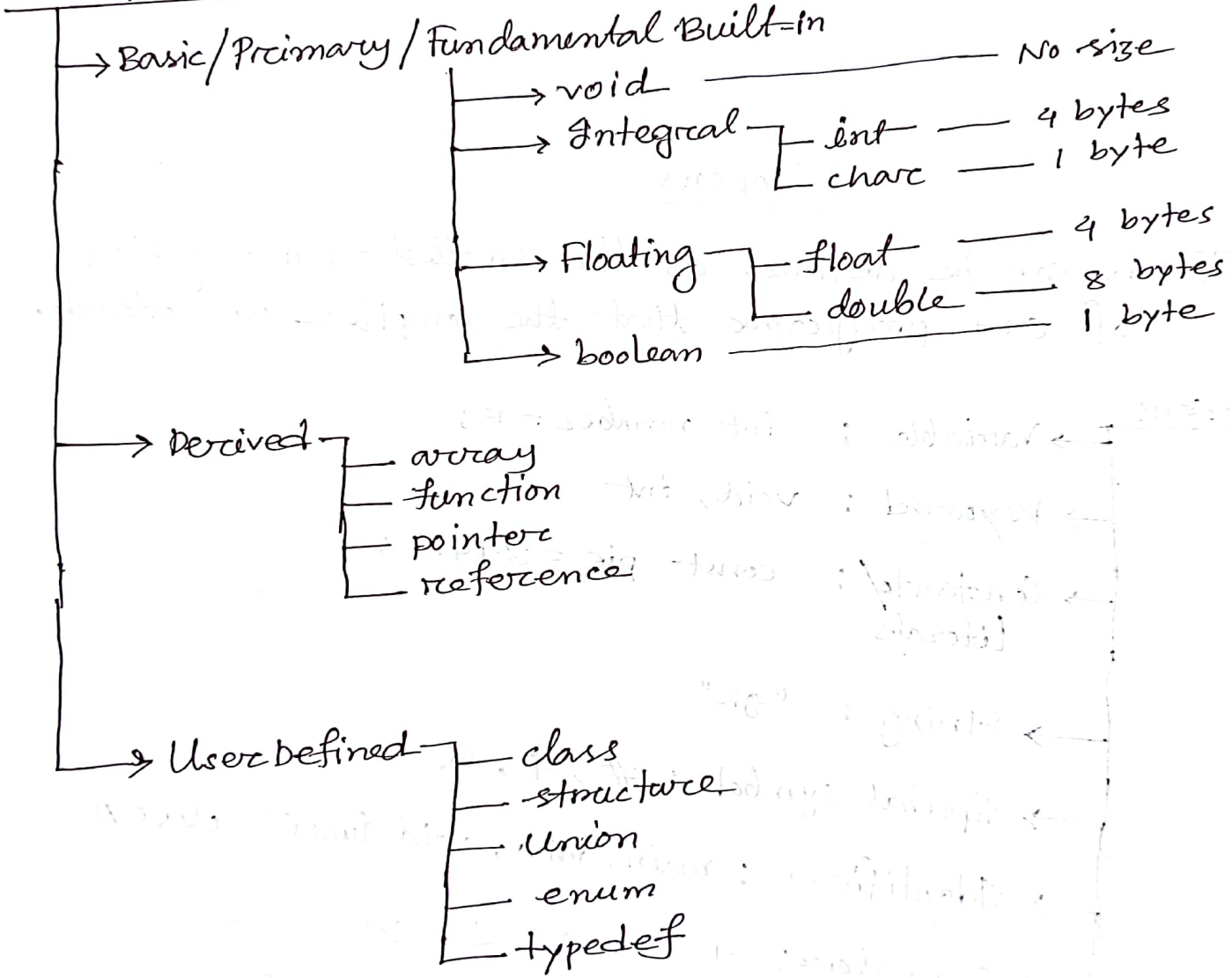


Tokens

Tokens can be defined as the smallest building block of C++ programs that the compiler understands.



Data Types in C++



The data type specifies the size and type of information the variable will store.

Variable

Variable is the name of memory location where we can store data/value.

Syntax

```
type variable_name = value;
int number = 10;
char myletter = 'G';
float cgpa = 3.88;
double pie = 3.1416;
string myName = "gk";
bool myBoolean = true;
```

Multiple Variable

```
int a=b=c=10;
int a,b,c;
a=b=c=10;
```

Rules

- Names can contain letters, digits and underscores (_)
- Begin with a letter or an (_) or not number
- case sensitive
- Not allowed white space or special symbols like !, #, %.
- Reserved words/keywords cannot be used as variables.

Keyword

Keywords is nothing but reserved word whose meaning already defined on the compiler.

example — int, char, float, static, class, struct ------.

≈ 48 keywords in c++

Constant / Literals

Constant means fixed value which does not change in runtime.

- used const keyword
- constant can be of any datatype
- we can change the value of constant forcely using pointer.

example

```
const int number =10;
const double PI = 3.1416;
const int magic_number = 42;
```

string (details in next)

The string type is used to store a sequence of characters.

```
string names = "gfk";
```

- string must be surrounded by double quotes.

Special symbols

special characters/escape sequence

Identifiers

Identifier refers a name used to identify a variable, function, class, module or any other user-defined item.

- Keyword cannot be used as identifier.

```
int a;
void funcc)
class A
```

- short names (a, b)
- descriptive names (age, sum) ✓

Operator

Operator is a symbol that is used to perform mathematical and logical task.

Types

- Arithmetic : +, −, *, /, %
- Relational : >, <, >=, <=, ==, !=
- Logical : &&, ||, !
- Assignment : =, +=, -=, *=, /=
- Ternary : ? : (if-else)
- Bitwise : &, |, ^, ~, <<, >>

strings

string concatenation

```
string firstName = "Gholam";
string lastName = "Kaderye";
string fullName = firstName + lastName;
string full_Name = firstName + " " + lastName;
cout << fullName;
cout << full_Name;
```

Append

```
string fullName = firstName.append(lastName)
```

Adding Numbers and Strings

```
string number_1 = "10";
string number_2 = "20";
string number = number_1 + number_2;
```

string length

```
string txt = "abcdefghijklmnopqrstuvwxyz";
cout << "The length is: " << txt.length();
cout << "The length is: " << txt.size();
```

Access strings

```
string names = "Kaderye";
cout << names[0];                    // output K
cout << names[names.length()-1];     // output e
names[0] = 'D';
cout << names;                       // output Daderye
cout << names.at(0);                 // output first character
```

Special characters

```
string txt = " I am \" Golam Kadoriye \" ";
string txt_1 = " It\'s abright ";
string txt_2 = " The character \\ is called backslash.";
```

User Input strings

```
string firstName;
cout<< " Type your first name : ";
cin >> firstName; //or getline(cin, firstName)
cout<< " Your first name is : " <<firstName;
```

Omitting Namespace

```
#include <iostream>
#include <string>

int main() {
    std:: string names = " gk ";
    std:: cout << names;

    return 0;
}
```

Formatting

- showpoint
- noshowpoint
- fixed
- setprecision ( )
- setw ( )

Math

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

int main (){
    cout<< "The maximum number is: "<< max(5,15)<<endl;
    cout<< "The minimum number is: "<< min(5,15)<<endl;
    cout<< "The square root is: "<< sqrt(64) << endl;
    cout<< "The round value is: "<< round(3.7) << endl;
    cout<< "The logarithm value is: "<< log(2) << endl;

    return 0;

}
```

Conditional Statement

- if
- if-else
- nested if-else
- else-if
- switch

Any meaningful expression is called a statement.

ex

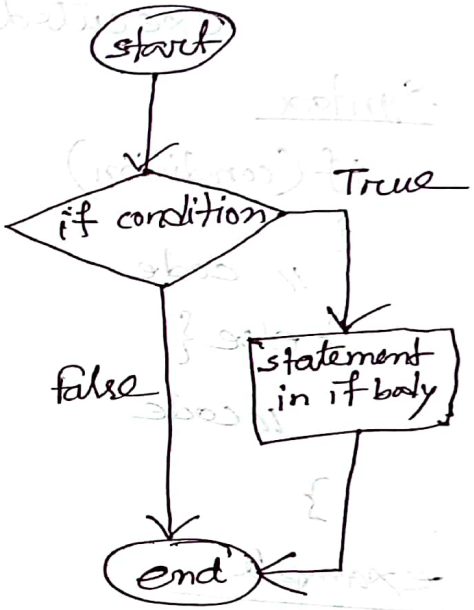
```
x=15;
cout << "gfk";
```

Control statement

- Selection control statement or condition
  - if
  - if-else
  - nested if-else
  - else-if
  - switch
- Iteration/looping
  - while
  - do-while
  - for
  - nested for
- Jump
  - break
  - continue
  - return
  - exit
  - goto

if statement

If statement test conditions, if condition is true then if block code will be executed otherwise no action taken.



Syntax

```
if (condition)
{
  // block of codes
}
```

example

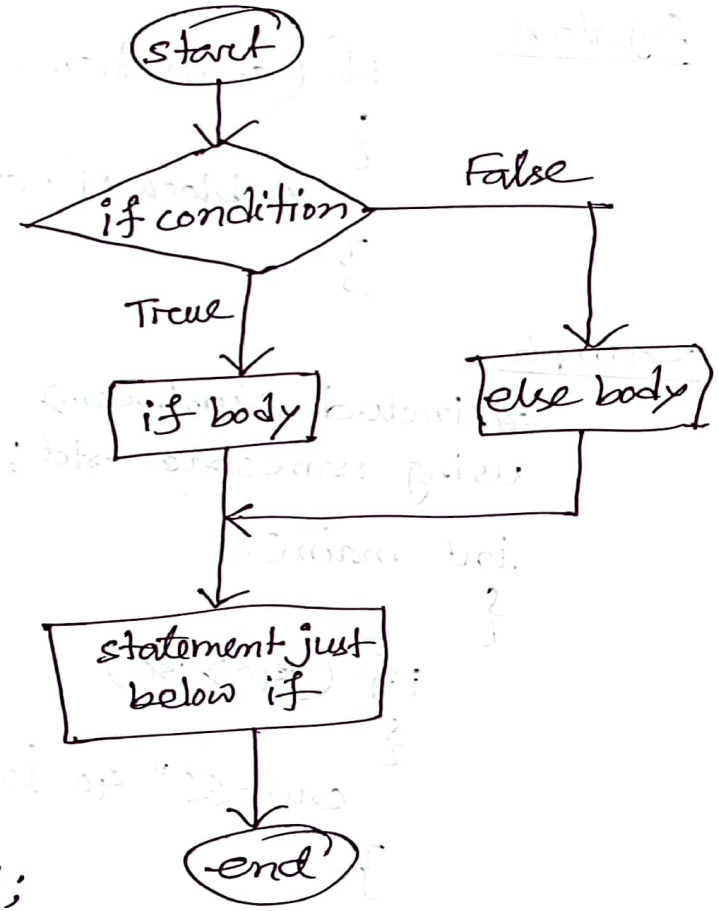
```
#include <iostream>
using namespace std;
int main()
{
   if (40>30)
   {
     cout<<" 40 is greater than 30 ";
   }
  return 0;
}
```

if_else statement

It is used to execute two statements for a single condition, if given condition is true, if block executed otherwise else block will be executed.

Syntax

```
if(condition)
{
  // code
}else{
   // code
}
```



example :

```
#include<iostream>
using namespace std;
int main(){
    int age=25;
    if(25>=18){
      cout<<"You are eligible";
    }else{
      cout<<"You are not eligible";
    }
return 0 }
```

```
#include <iostream>
using namespace std;
int main()
{
  int number;
  cout<<"Enter a number: ";
  cin >> number;
  if(number % 2 == 0)
  {
    cout<< number << " is even";
  }else{
    cout<<number<<" is odd";
  }
return 0;
}
```

```
#include <iostream>
using namespace std;
int main()
{
  int pin;
  cout<<"Enter pin code: ";
  cin >> pin;
  if (pin == 4732)
  {
    cout<<"Correct pin\n";
  }else{
    cout<<"Wrong pin\n";
  }
return 0;
}
```

```
int time = 25;
string result = (time < 18) ? "Good day" : "Good evening";
cout<< result;
```

nested if-else statement

nested if-else is used when we consider if-else inside the if-else inside the if-else and so on.

Syntax

```
if(condition)
{
   if (condition)
   {
      // code;
   }else{
      // code
   }
}else{
    //code
}
```

nested if-else

```
#include<iostream>
using namespace std;
int main()
{
  int number=6;
  if(number%2==0)
  {
    if(number%3==0)
    {
      cout<<"Divisible by 2 and 3";
    }else{
      cout<<"Divisible by 2 but not 3";
    }
  }else{
    cout<<"Not divisible by 2";
  }
  return 0;
}
```

```
#include <iostream>
using namespace std;
int main()
{
   int a=10, b=8, c=6;
   if(a>b)
   {
      if(a>c)
      {
      cout<<"Largest is: "<<a;
      }else{
        cout<<"Largest is: "<<c;
      }
   }else{
        if(b>c)
        {
          cout<<"Largest is: "<<b;
        }else{
          cout<<"Largest is: "<<c;
        }
   }
   return 0;
}
```

else_if statement

else_if ladder statement is used when we have multiple conditions.

Syntax

```
if (condition-1)
{
  // code
} else if (cond^n 2){
  // code
} else if (cond^n 3){
  // code
} else{
   //code
}
```

```
#include <iostream>
using namespace std;
int main() {
   int a, b, ch, add, sub, mul, div;
   cout<< "Enter two numbers: ";
   cin >> a >> b;
   cout<< "Enter choice: ";
   cin >> ch;
   if (ch==1){
      add = a+b;
      cout<< add;
    } else if (ch==2){
      sub = a-b;
      cout<< sub;
    } else if (ch==3){
      mul = a*b;
      cout<< mul;
    } else if (ch==4){
      div = a/b;
      cout<< div;
    } else{
      cout<< "Invalid Task";
    }
  return 0;
}
```

switch statement

Switch statement used when we want to select only one case out of multiple cases.

Syntax

```
switch(expression)
{
  case value1:
   // code
   break;
  case value2:
   // code
   break;
  case value3:
   // code
   break;
  default:
    // code

}
```

```
int main(){
   int choice;
   cout<<"Enter your choice: ";
   cin>>choice;
   switch(choice){
      case 1:
         cout<<"Saturday";
         break;
      case 2:
         cout<<"Sunday";
         break;
      case 3:
         cout<<"Monday";
         break;
      case 4:
         cout<<"Tuesday";
         break;
      case 5:
         cout<<"Wednessday";
         break;
      case 6: cout<<"Thuresday";
         break;
      default: cout<<"Friday";
   }}
```

~~while~~ Increment & Decrement

Increment operator is used to increase the value of variable by 1.

Types
- pre-increment (++a)
- post-increment (a++)

Decrement operator is used to decrease the value of variable by 1.

Types
- pre-decrement (--a)
- post-decrement (a--)

Loops

Loop is nothing but iterative statement which allow a block of code to be executed repeatedly.

while loop

while loop is also known as entry controlled loop.

The statement will be executed continuously until the given condition is no longer satisfied.

Syntax

```
while(condition)
{
  // statement
  increment/decrement;
}
```

```
#include<iostream>
using namespace std;
int main(){
   int i=0;
   while(i<=10){
      cout<<i<<endl;
      i++;
   }
   return 0;
}
```

```
int main(){
   int numbers=12345;
   int revNum=0;
   while(numbers){
      revNum = revNum*10 + numbers%10;
      numbers /=10;
   }
   cout<<"Reversed numbers: "<<revNum<<endl;
   return 0;
}
```

```
while(1){
  cout<<"gk";
}
// infinite loop
```

do_while loop

It is also known as exit controlled loop, because it tests conditions at the end of loop body.

→ It executes at one even the given condition is T or F.

Syntax

```
do{
  // code
  incre/decre;
} while (condn);
```

```
int main(){
    int i=0;
    do{
      cout<< i << endl;
      i++;
    } while (i<=10);
  return 0;
}
```

```
int count=1;
do{
  // code
}while(count==1);
// infinite
```

```
// Print the sum of positive numbers
int main(){
    int number=0;
    int sum=0;
    do{
        sum+=number;
        cout<<"Enter a number: ";
        cin>>number;
    } while(number>=0);
    cout<<"The sum is: " << sum <<endl;
  return 0;
}
```

for loop

Unlike while loop, for loop performs all operations in single line.

Syntax

```
for(ini ; cond^n ; in/de)
{
  // code
}
```

```
for(int i=1; i>0; i++)
{ // infinite loop
}
```

```
for( ; ;){
  // code
}
```

```
int main(){
  for(int i=0; i<=10; i++){
    cout<< i <<endl;
  }
  return 0;
}
```

```
// Nested for loop
int main(){
  for(int i=1; i<=2 ; i++){
    cout << i <<endl;
    for(int j=1; j<=3 ; j++){
      cout<< j << endl;
    }
  }
  return 0;
}
```

foreach / ranged-based for loop

```
int numbers[5] = { 1, 2, 3, 4, 5 };
for (int i : numbers) {
    cout << i << endl;
}
```

Nested for loop

```
int main() {
    int rows;
    cout << "Enter number of rows: ";
    cin >> rows;
    for(int i=1; i<=rows; ++i) {
        for(int j=1; j<=i; ++j) {
            cout << "* ";
        }
        cout << "\n";
    }
    return 0;
}
```

output

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

// break

```
int main (){
   int i;
   for(i=1; i<=10; i++)
    {
     if (i==5)
      break;
      cout<< i;
     }
  return 0;
}
```

// continue

```
int main (){
    int i;
    for(i=1; i<=10; i++)
     {
       if (i==5)
        continue;
        cout<< i <<endl;
      }
   return 0;
}
```

// goto

```
int main (){
    int age;
    cout<< "Enter your age: ";
    cin >> age;
    if(age>18)
      goto Vote;
    else
      goto Notvote;

    Vote:
        cout<< "Eligible";

    Notvote:
        cout<< "Not eligible";
   return 0;
}
```

// exit

```
int main (){
   cout<< " gk " <<endl;
   cout<< " pk " << endl;
   exit(0);
   cout<< " mk " <<endl;
   cout<< " jk " <<endl;

return 0;
}
```

Derived data: Array

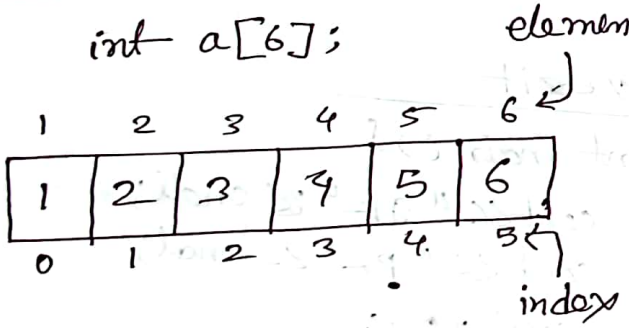
Array is a derived datatype which is constructed by the help of primitive datatype. It stores multiple values in single variables with continuous memory locations.

Syntax

data_type array_name[size]

example

```
int a[6];
```




```
a[2] = 3;
a[1] = 2;
a[0] = 1;
```

Types

- 1D array
- 2D array

```
int main(){
  int arr[5]={5,10,15,20,25};
  int i;
  for(i=0; i<5; i++){
    cout<<arr[i]<<endl;
  }
  return 0;
}
```

```
int main(){
  int arr[5]={5,10,15,20,25};
  cout<<arr[0];
  return 0;
}
```

```
//change array element
arr[0] = 50;
```

```
// Get the size of an array
int arr[5] = { 10, 20, 30, 40, 50};
cout << sizeof(arr);

// Get the size of each elements
int getArray = sizeof(arr) / sizeof(arr[0]);
cout << getArray;
```

```
int main (){
    int ages[8] = { 20, 22, 18, 35, 48, 26, 87, 70};
    float avg, sum = 0;
    int i;
    int length = sizeof(ages) / sizeof(ages[0]);
    for (i=0; i< length; i++){
        sum = sum + ages[i];
    }
    cout << "The sum is: " << sum << endl;
    avg = sum/length;
    cout << "The average is: " << avg << endl;
    return 0;
}
```

2D Array

int x[3][3]; — row, column

	col 0	col 1	col 2
row 0	x[0][0]	x[0][1]	x[0][2]
row 1	x[1][0]	x[1][1]	x[1][2]
row 2	x[2][0]	x[2][1]	x[2][2]

```
int x[3][3] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
int x[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
```

	c0	c1	c2
r0	1	2	3
r1	4	5	6
r2	7	8	9

```
#include<iostream>
using namespace std;
int main(){
    int x[3][3], i, j;
    for cout<<"Enter Array Elements:"<<endl;
    for(i=0; i<=2; i++){
        for(j=0; j<=2; j++){
            cin >> x[i][j];
        }
    }
    cout<<"output Array:"<<endl;
    for(i=0; i<=2; i++){
        for(j=0; j<=2; j++){
            cout<< x[i][j];
        } cout<<endl;
    }
    return 0;
}
```

String

A string is a one-dimensional array of characters terminated by null characters.

```
#include<iostream>
#include<string.h>
using namespace std;
int main (){
    char str[] = "kaderye";
    cout<< str<<endl;
    int n = strlen(str);
    cout<<n;
  return 0;
}
```

```
strrev(str2)    // reversed
strcat(str1, str2)    // concatenation
strcpy(str, str1)    // str1 -> str
strlwr(str2)    // lower case
strupr(str2)    // upper case
```

Pointers

A pointer is a variable that stores the address of another variable.

Syntax

```
data_type* name;
```

example

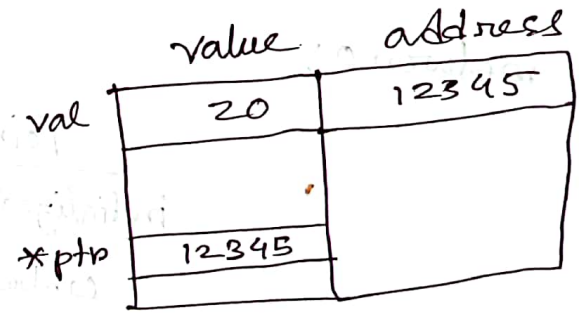
```
int* ptr;

int val = 20;
int* ptr = &val;
```

Size

8 bytes for a 64-bit OS.
4 bytes for a 32 bit OS

While working with pointer we need to required two unary operators.

(a) & - Address of operator
(b) * - Value at address operator




```
print *ptr;
  // output value: 20
print ptr;
  // output address : 12345
```

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

int main() {
    string food = "Pizza";
    string* ptr = &food;
    cout << food << endl;      // Pizza
    cout << &food << endl;     // address
    cout << *ptr << endl;      // Pizza
    *ptr = "mango";
    cout << *ptr << endl;      // mango
    cout << food << endl;      // Mango
    return 0;
}
```

Pointer	Reference
initialized after declaration	init^{n} at the time of decl^{n}
can be null	cannot be null
can be reassigned	cannot be reassigned

structure

A structure is a collection of variables of different data types and member functions under a single name.

```
// create a structure
struct {
    string brand;
    string model;
    int year;
} myCar;
```

```
#include <iostream>
#include <string>
using namespace std;
int main () {
  struct {
    string brand;
    string model;
    int year;
  } myCar1, myCar2;

  myCar1.brand = "BMW";
  myCar1.model = "X5";
  myCar1.year = 1999;

  myCar2.brand = "Ford";
  myCar2.model = "Mustang";
  myCar2.year = 1969;
  cout << myCar1.brand << " " << myCar1.model << " " << myCar1.year << endl;
  cout << myCar2.brand << " " << myCar2.model << " " << myCar2.year << endl;

  return 0;
}
```

Function

A function is a block of code which only runs when it is called.

Types
- Pre defined / Standard library / Built-in
- User defined

A function has three parts
- function definition — একটা function কি কাজ করবে তা নির্ধারণ করে
- function prototype — প্রয়োজনীয় statement লেখা হয় function definition এর আগে
- function calling or invoking
  - call by value
  - call by reference

function definition

```
data_type function_name (parametre/arguments list)   } function header
{                                                     } function body
   // statement
}
```

```
void xyz(int x, float y, char ch) {   // formal parametre
}
xyz(10,20,N)   — actual parametre
```

Storage class

Storage class defines the scope and lifetime of variable and functions.

Storage class	Memory	Default value	Scope	lifetime
auto (local)	RAM	garbage	within block	still block is active
static	RAM	0	within block	till the terminate of program
extern (global)	RAM	0	anywhere	〃
register	Register	garbage	within block	still the block is active

```
#include<iostream>
using namespace std;
int a;      // extern (global)
int main(){
    int b;      // auto (local)
    static int c;    // static
    register int d;    // register

    cout<<a<<endl;      // 0
    cout<<b<<endl;      // garbage
    cout<<c<<endl;      // 0
    cout<<d<<endl;      // garbage

return 0;
}
```

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

void fun(){
    auto int a=10;
    static int b=10;
    cout<<a<<endl<<b<<endl;
    ++a; ++b;
}
int main(){
    fun();
    fun();
    fun();
    fun();
  return 0;
}
```

output

10 10
10 11
10 12
10 13

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

int a =10;
void fun (){
    cout<< a<< endl;
}
int main (){
      int b=20;
      fun();
      fun();
      fun();
      fun();
       cout<< a<< ends<< b;
   return 0;
}
```

```
//output
10
10
10
10 20
```

```
#include<iostream>
using namespace std;
void fun(int a, int b) {
    cout<<a<<endl<<b;
}
int main() {
    fun(10,20);
  return 0;
}
```

//output

```
10
20
```

Union

The size of union is equal to its biggest member size.

Syntax

```
union union-name
{
    // members
};
```

```
#include<iostream>
using namespace std;
struct stu{
    int marks;      // 4 bytes
    float avg;      // 4
    double salary;  // 8
};
union struct stu2{
    int marks;
    float avg;
    double salarey;
};
int main(){
    struct stu s;
    union stu2 s2;
    cout<<"structure "<<sizeof
        (s);
    cout<<"union "<<sizeof
        (s2)
    return 0;
}
```

enum

Enumeration is a user defind name list consists of integral constants.

Syntax

enum_name { value1, value 2, . . . value(n) };

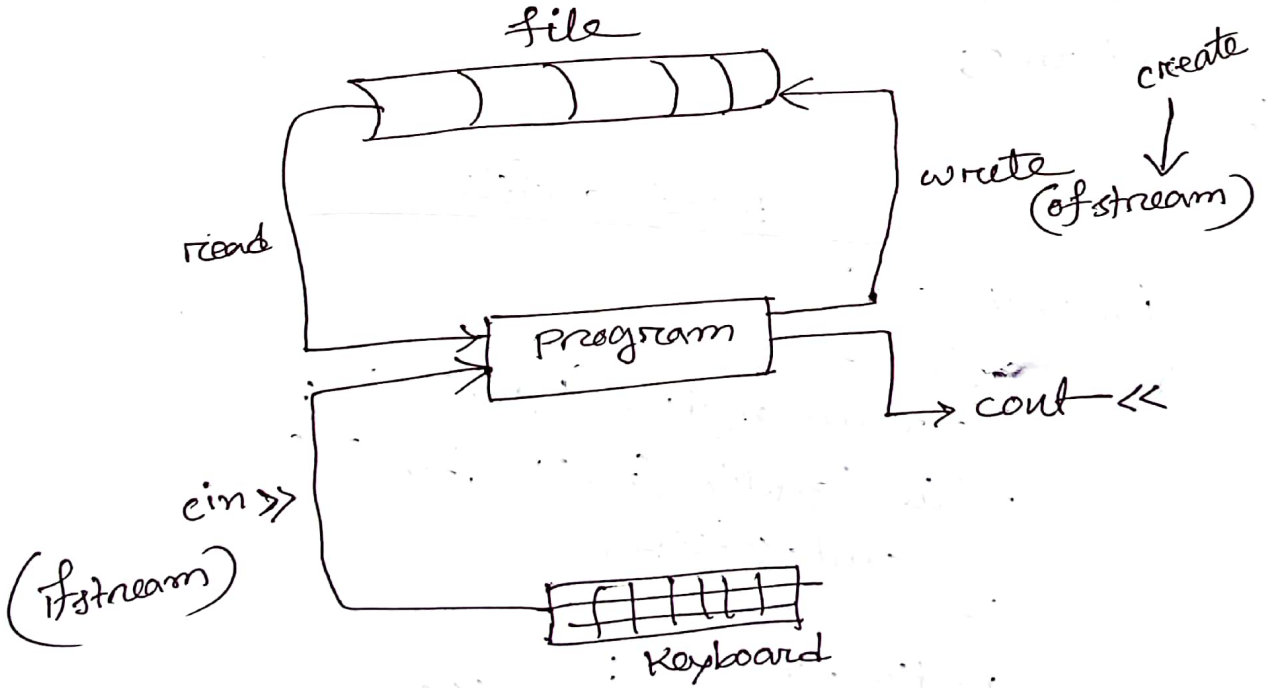
#define

```
#include <iostream>
# define num 10
using namespace std;
int main() {
    int i, r;
    for(i=1 ; i<=10 ; ++i) {
        r = num * i;
        cout << num << "*" << i << "=" << r << endl;
    }
    return 0;
}
```

```
#include <iostream>
# define num(a, b) ((a > b)? a : b)
using namespace std;
int main (){
    cout << num (236, 167) << endl;
    cout << num (564, 1675) << endl;
    cout << num (342, 167) << endl;
    cout << num (236, 453) << endl;
    cout << num (5643, 167);
return 0;
}
```

File Handling

File handling is a mechanism so that we can store the output of the program in the file and we can perform many operations on the data present in a file.



Create and Write a file

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
    ofstream myfile("gk.txt");
    myfile << "I am Golam Kaderye.";
    myfile.close();
return 0;
}
```

Read a file

```
int main(){
    string str;
    ifstream myFile("gk.txt");
    while(getline(myFile, str)){
        cout << str;
    }
    myFile.close();
return 0;
}
```

Exception handling

An exception is unexpected/ unwanted/ abnormal situation that occured at runtime called exception.

Syntax

```
try {
    throw exception
} catch (type arg) {
    // solved problem
}
```

```
#include <iostream>
using namespace std;
int main () {
  cout<< " Exception starting " <<endl;
  int a, b, c;
  cout<< Enter two numbers: ";
  cin >>a>>b;
  try {
      if (b==0)
        throw b;
        c=a/b;
        cout<<" Result: "<<c;
    } catch (int x) {
        cout<<" Can't divide by " << x;
    }
  cout<<" Exception end ";
return 0;
}
```

Function

```
#include<iostream>
using namespace std;
void fun( ){
  cout<<"gk";
}
int main(){
   fun();
 return 0;
}
```

```
#include<iostream>
using namespace std;
void fun();
int main (){
     fun();
     fun();
     fun();
  return 0;
}
void fun( ){
     cout<<"gk";
}
```

```
#include<iostream>
#include<string>
using namespace std;
void fun(string name)
{
  cout<< name<<" mk"<<endl;
}
```

```
int main( ){
   fun("gk");
   fun("pk");
   fun("rk");
  return 0;
}
```

// parameter/arguments

```
// default parameter
void fun(string country = "Canada") {
    cout << country << endl;
}

int main() {
    fun("USA");
    fun("KSA");
    fun();
    fun("BD");
    return 0;
}
```

//output

```
USA
KSA
Canada
BD
```

// multiple parameters

```
void fun(string name, int age){
    cout<< name << "is " <<age << " years old" <<endl;
}
int main(){
    fun("gk", 28);
    fun("mk", 26);
    return 0;
}
```

// output

gk is 28 years old
mk is 26 years old

// return value

```
int fun(int x){
    return 5+x;
}
int main(){
    cout<< fun(4);
  return 0;
}
```

// output
9

```
int fun(int x, int y){
    return x+y;
}
int main(){
    cout<< fun(3,7);
  return 0;
}
```

// output
10.

// pass by reference

```
#include <iostream>
using namespace std;
void swapNumbers(int &x, int &y){
    int temp = x;
        x = y;
        y = temp;
}
int main(){
    int number_1 = 30;
    int number_2 = 50;
    cout << "Before swap" << endl;
    cout << number_1 << number_2 << endl;
    swapNumbers(number_1, number_2);
    cout << "After swap" << endl;
    cout << number_1 << number_2 << endl;
return 0;
}
```

4 pass array

```
#include <iostream>
using namespace std;
void fun(int arr[5]){
    for(int i=0; i<5; i++){
        cout<< arr[i] <<endl;
    }
}
int main(){
    int arr[5] = {5, 10, 15, 20, 25};
    fun(arr);
    return 0;
}
```

// Fahrenheit to celsius

```
float toCelsius(float fahrenheit){
    return (5.0/9.0)*(fahrenheit-32.0);
}
int main(){
    float f_value=98.8;
    float result=toCelsius(f_value);
    cout<<"Fahrenheit: "<<f_value<<endl;
    cout<<"celsius: "<<result<<endl;
    return 0;
}
```

output

Fahrenheit = 98.8

celsius = 37.1111

Function Overloading (occurs in compile time)

With function overloading, multiple functions can have the same name with different parameters. Compile time polymorphism / early binding / static polymorphism.

```
int add(int x, int y){
    return x+y;
}
double add(double x, double y){
    return x+y;
}
int main(){
    int num-1 = add(2,3);
    double num2 = add(2.7, 8.2);
    cout<< "Int: " << num-1 <<endl;
    cout<< "Double: " << num-2;

    return 0;
}
```

Function Overriding (execution time its occured)

Function overriding is when a subclass provides its own version of a method that is already defined in its parent class, using the same name, return type and parameters.

runtime polymorphism / late binding / dynamic polymorphism.

```
class Parent{
  public:
    void display(){
      cout<<" Base class "<<endl;
    }
};
class Child : public Parent {
  public:
    void display(){
      cout<<" Derived class "<<endl;
    }
};
int main(){
  child obj-1;
  obj-1. display();
  return 0;
}
```

A local variable cannot be used outside the function it belongs to.

A global variable created outside of a function and can therefore be used by anyone.

```
#include <iostream>
using namespace std;
int x=10;          // global variable
void fun(){
    int x=25;      // local variable
    cout<< x <<endl;
}
int main(){
    fun();
    cout<< x ;
    return 0;
}
```

// output

25
10

Recursion

A function that calls itself is known as a recursive function.

This technique is known as recursion.

```
int sum (int k){
    if (k > 0) {
        return k + sum (k-1);
    } else {
        return 0;
    }
}

int main () {
    int result = sum (10);
    cout << result;
    return 0;
}
```

```
10 + sum (9)
10 + 9 + sum (8)
10 + 9 + 8 + sum (7)
10 + 9 + 8 + 7 + sum (6)
    :
    ;
```

output

55

// Factorial Numbers

```
#include <iostream>
using namespace std;
int factorial (int);

int main(){
    int n, result;
    cout<<"Enter a number: ";
    cin>>n;
    result=factorial(n);
    cout<<"Factorial of "<<n<<" = "<<result;
    return 0;
}

int factorial(int n){
    if(n>1){
        return n*factorial(n-1);
    }else{
        return 1;
    }
}
```