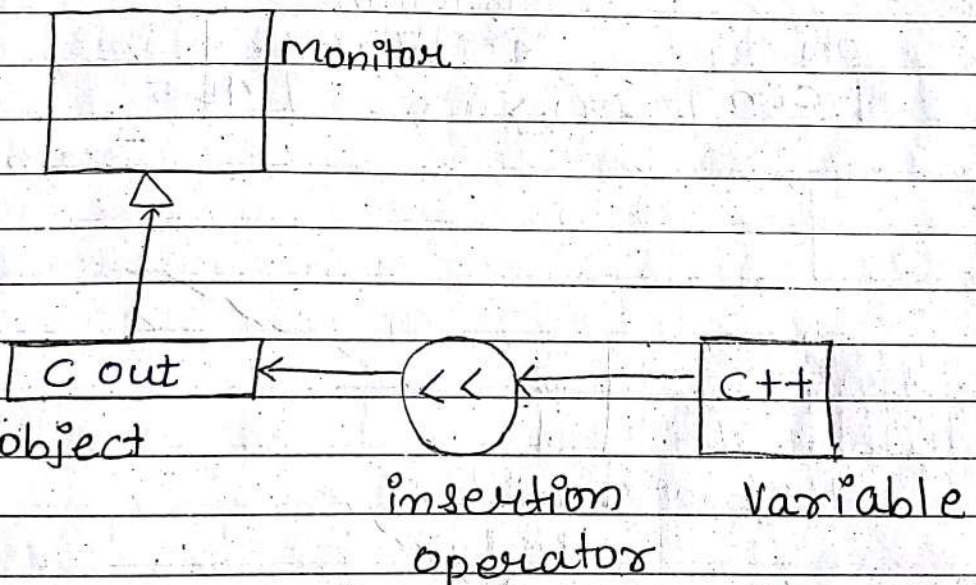
Date.: 22-6-18
MON TUE WED THU FRI SAT

Chapter-2

```
#include <iostream>
using namespace std;
int main()
{
    cout << "c++ is better than c in";
    return 0;
}
```

- Output Operator

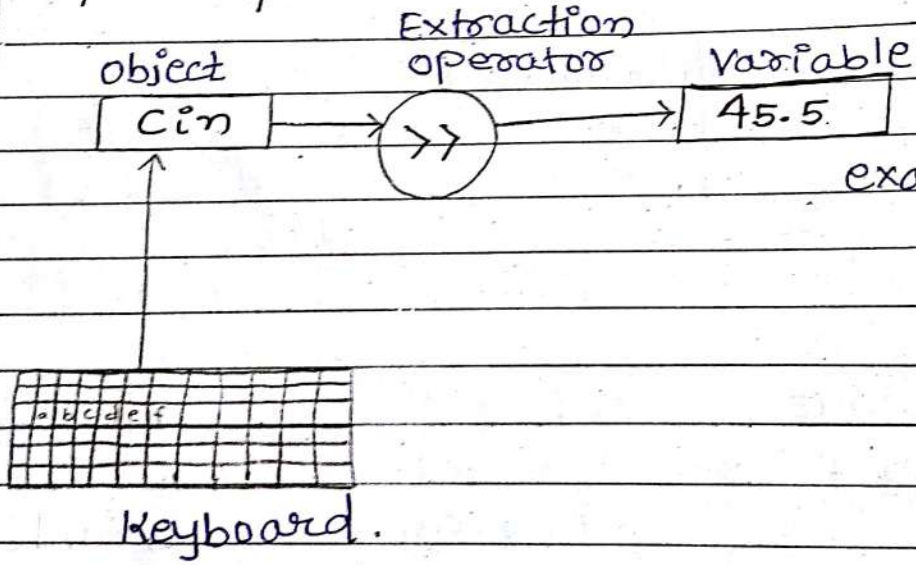



```
#include <iostream>
using namespace std;
int main()
{
    float num1, num2, sum, avg;
    cout << "Enter the number";
```

```
cin >> num1;
cin >> num2;
Sum = num1 + num2;
avg = Sum / 2;
cout << "sum = " << sum << "\n";
cout << "avg = " << avg << "\n";

return 0;
}
```

- Input Operator.



- Cascading of I/O Operators

```
Cout << "Sum = " << Sum << ", "
     << "Average = " << Average << "\n";
```

```
#include<iostream>
using namespace std;
class Person
{
  var{ char name[30];
       int age;
       public : → specifier
  fun{ void getdata(void);
       void display(void);
};

Void Person :: getdata(void)   // scope resolution operator
{
    cout << "Enter name";
    cin >> name;
    cout << "Enter age";
    cin >> age;
}
Void Person :: display(void)
{
    cout << "\n name : " << name;
    cout << "\n age : " << age;
}
int main()
{
    person.p;
    P.getdata()
    P.display();
    return 0;
}
```

function member definition

Scanned by CamScanner

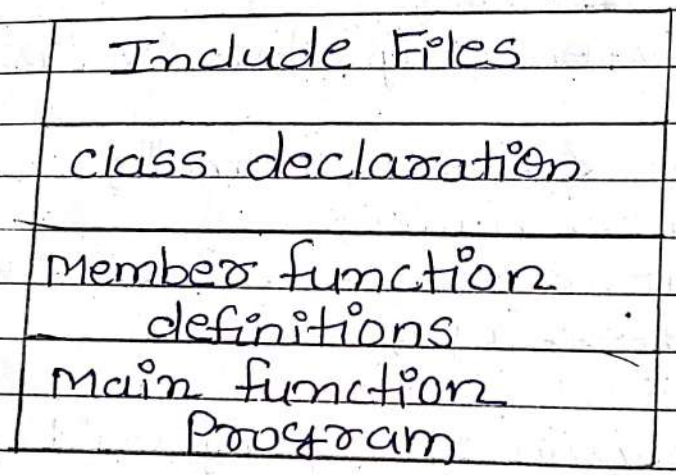
Date:
MON TUE WED THU FRI SAT

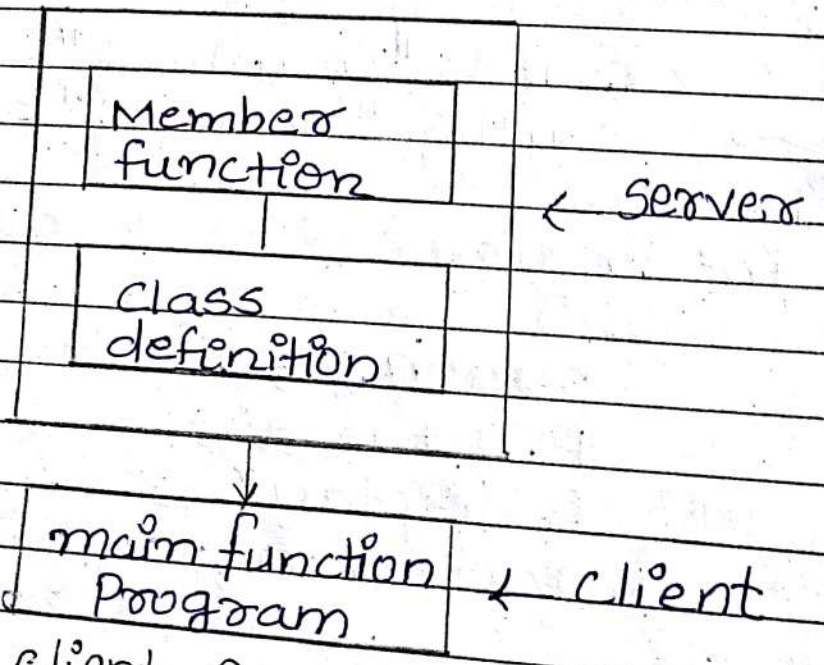
g++ text.cpp -o cypit
./cypit

Output

Enter name
Snehal
age
19
name : Snehal
age : 19

Structure of C++ program





The client Server model

Date.:
MON TUE WED THU FRI SAT

Keyword (New Keyword in C++)

class	private
delete	public
catch	protected
friend	throw
inline	try
new	this
operator	template
	Virtual

```
#include <iostream>
using namespace std;
class Student
{
    int mark1, total; mark2, mark3;
    public:
     void stu_marks(void);

};

Void Student::stu_marks(void);
{
    Cout << " Enter Marks in Three
            Subject = \n" ;
    cin >> mark1 >> mark2 >> mark3 ;

    total = mark1 + mark2 + mark3;
    Cout << " Total Marks = \n" <<
            total ;

}
```

Date.:
MON TUE WED THU FRI SAT

```
int main()
{
    Student S;
    S.stu_marks();

    return 0;
}
```

Token: Smallest individual unit in a program are known as token. C++ has the foll. tokens

1. Keyword
2. Identifier
3. Constant
4. String
5. Operator

Identifiers And Constant

Rules for naming the identifier.

- Only alphabetic character, digit & underscore (_) are permited.
- The name cannot start with digit.
- Uppercase and lowercase are distinct.
- Declare keyword cannot be used as a variable name.
- In Ansi C recognize only first 32 character in a name, ANSI C++ place no limit on the length of variable name.

Date:
MON TUE WED THU FRI SAT

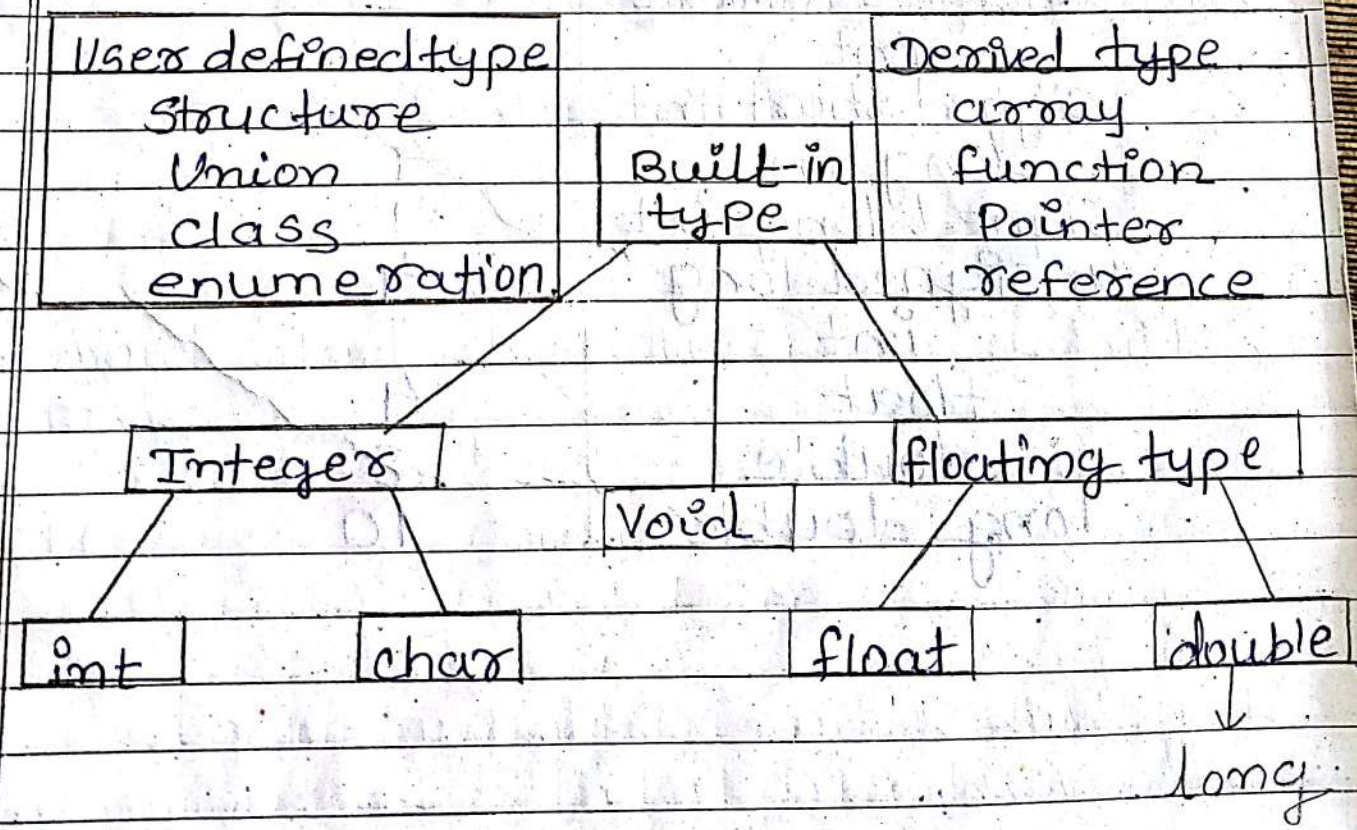
List of Literal Constant

```
123       // Decimal value
12.34     // floating point integer
037       // octal integer
0x2       // Hexadecimal integer
"C++"     // string Constant
'A'       // character
L 'ab'    // wide-character Constant
```

datatype to declare wide-character Constant `wchar_t`

Datatypes

C++ datatypes



Date.:
MON TUE WED THU FRI SAT

Modifier — signed, unsigned, long, short } char & int

Size

type	bytes
char	1
unsigned char	1
signed char	1
int	2
unsigned int	2
signed int	2
short int	2
unsigned short int	2
signed short int	2
long int	4
signed long int	4
unsigned long int	4
float	4
double	8
long double	10

Date.:
MON TUE WED THU FRI SAT

eg: Struct book.
{
char title[15];
Char Author[20];
int pages;
float price;
};
Struct book b1;
b1.pages = 500;
b1.price = 225.75;

Union result
{
int marks;
char grade;
float percent;
};

Structure	Union
1. Structure is defined with 'Struct' Keyword.	Union is defined with 'union' Keyword.
2. All members of an structure can be manipulated simultaneously	The members of a union can be manipulated only one at a time.
3. The size of structure object is equal to the sum of the individual sizes of the member	Size of union object is equal to the size of largest member object.

Date.:
MON TUE WED THU FRI SAT

object.	
4. Structure members are allocated distinct memory location.	Union members share common memory space for that uses.
5. Structure are not consider as a memory efficient in comparison to union.	Union are considers as a memory efficient in situation when members are not required to be accessed simultaneously.

Enumerated Data type

enum shape {circle, square, triangle};
enum colour {red, blue, green, yellow};
enum position {off, on};

(By default) C: int

↑ Keyword ↑ Tag name

shape ellipse;
colour background;

Colour background = blue; // allow
Colour background = 7; // Error in C++
colour background = (colour)7; // OK
↳ type casting

enum Colour {red, blue=4, green=8}
enum Colour {red=5, blue, green}

0 4 8
5 6 7

Date.:
MON TUE WED THU FRI SAT

C++ also permits the creation of anonymous enums (enums without tag name)

0 1

enum {off, on};

int switch-1 = off

int switch-2 = on

Storage classes

	Automatic (By Default)	External	Static	Register
lifetime	Function block.	Entire program	Entire program	Function block
Visibility	local	global	local	local
initial Value	Garbage	0	0	Garbage
Storage	Stack (primary memory) Segment	Data Segment	Data Segment	CPU register
Purpose	local Var. used by single fun	global Var. used throughout the Program	local var. retaining their values thro. o. the p	Var. using CPU register for storage Purpose register ↳ CPU
Keyword	auto	extern	Static	

Date:
MON TUE WED THU FRI SAT

Array

```
char strings[3] = "xyz";   // will take 10 automatically
char strings[4] = "xyz";   // C++
```

Symbolic Constant

Const & enum

`Const int size = 10;`	`enum {x, y, z};`
	`Const x = 0;`
	`Const y = 1;`
	`Const z = 2;`

Type Compatibility

sizeof('x') ⟶ sizeof(int) // C

size of ('x') ⟶ Size of (char) // C++

Dynamic Initialization of Variable

```
float area = 3.14 x rad x rad;

{float average;
 average = sum/i;

→ float average = sum/i;
```

Date.:
MON TUE WED THU FRI SAT

&
data type and reference.name = Variable.name

Ex: float total = 100;
float &Sum = total;

int n[10];
int &x = n[10];

char &a = '\n';

A major application of reference variable is in passing arguments to functions.

alias

```
void f(int &x) //use reference
{
    x = x+10; // x is incremented, so also n
}
int main()
{
    int m=10;
    f(m); // fun call
}
```

Date.: MON TUE WED THU FRI SAT

:: Scope resolution op.

::* pointer-to-member declarator

->* pointer-to-member op.

.* Pointer-to-member op.

delete memory release op

\n ← endl Line feed op.

new memory allocation op,

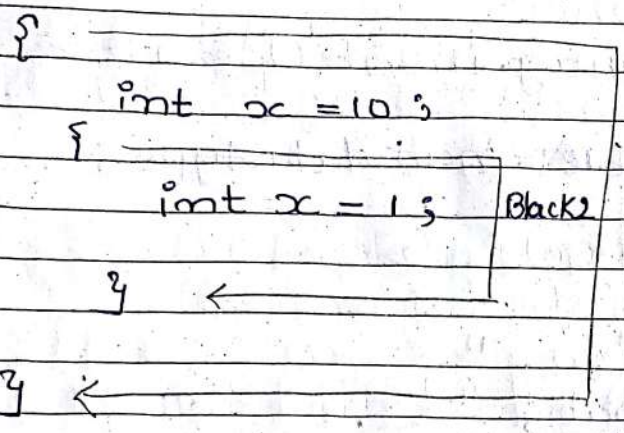
setw field width op.

Scope Resolution Operator

e.g:

```
{
    int x=10;
    ....
}
    ....
{
    int x=1;
    ....
    ....
}
```

Date.:
MON TUE WED THU FRI SAT



e.g:

```
#include <iostream>
using namespace std;
  int m = 10;  // global m
  int main()
{
   int m = 20;  // m redeclared, local to main
   {
      int k = m;
         int m = 30;
    Cout << "We are in inner block";
    Cout << "k = " << k << endl;  // k = 20
    Cout << "m = " << m << "\n";  // m = 30
    cout << "::m = " << ::m << "\n";  // m = 10
   }
   Cout << "\n We are in outer block";
   Cout << "m = " << m << "\n";  // m = 20
   Cout << "::m = " << ::m << "\n";  // m = 10
   return 0;
}
```

Date.:
MON TUE WED THU FRI SAT

Memory Management Operators :-

Pointer_Variable = new data_type ;

e.g :-
```
int *p ;
float *q ;
p = new int ;
q = new float ;

int *p = new int ;
float *q = new float ;
*p = 25 ;
*q = 7.5 ;

pointer_variable = new data_type (value) ;
   int *p = new int (25) ;
   float *q = new float (7.5) ;
```

Array data_type

```
point_variable = new data_type [size] ;
   int *p = new int [10]

p = new int [3] [5] [4] } // legal
p = new int [m] [5] [4] }

p = new int [3] [5] [ ] } // illegal
p = new int [ ] [5] [4] }
```

Date.:
MON TUE WED THU FRI SAT

e.g :- class sample
{
Private:
int P;
float q;
....
};
int main()
{
.....
Sample *ptr = new sample;
}

Delete
delete pointer variable;

e.g:- delete P;
delete q;

For array
delete [size] pointer-variable;

e.g:-
delete []p; // Delete whole array
delete [5]p; // Delete values from 0-4

Date.:
MON TUE WED THU FRI SAT

(iomanip)
Manipulator (तुमारे तुमारा output कुछ Format में होना चाहिये)

endl

```
int m = 2597;
int n = 14;
int p = 175;

cout << "m = " << m << endl;
     << "n = " << n << endl;
     << "p = " << p << endl;
```

o/p

m =	2	5	9	7
n =	1	4		
p =	1	7	5	

setw

```
int Sum = 345;
cout << setw(5) << Sum << endl;
```

o/p

		3	4	5

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    int Basic = 950, Allowance = 95,
        Total = 1045;
```

Date.:
MON TUE WED THU FRI SAT

```
cout << setw(10) << "Basic" << setw(10)
        << Basic << endl;
<< setw(10) << "Allowance" << set
   w(10) << Allowance << endl;
<< setw(10) << "Total" << setw(10)
        << Total << endl;
return 0;
}
```

O/P

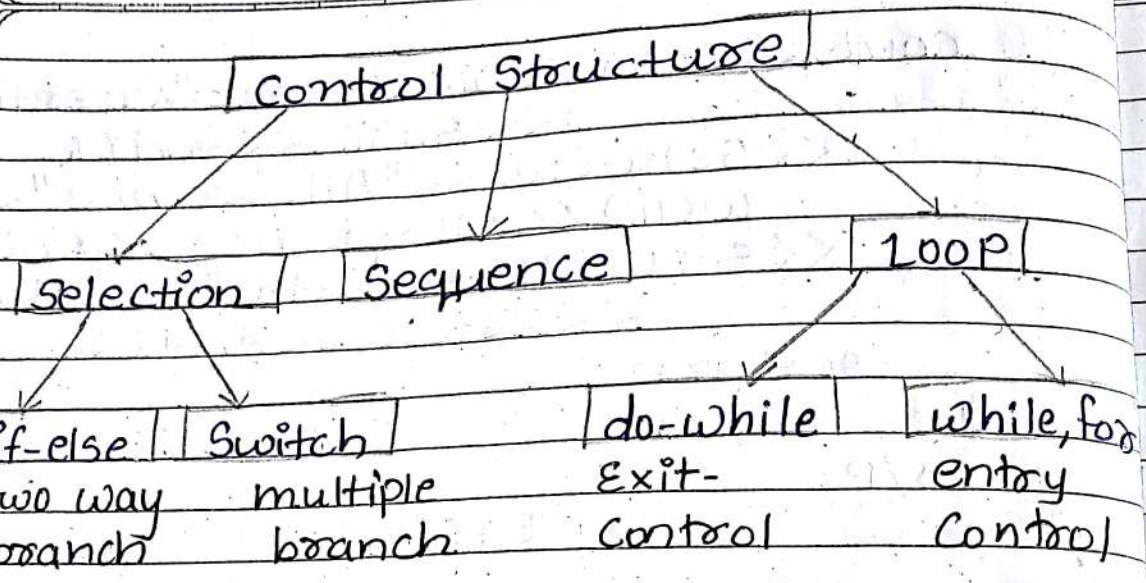
Basic	950
Allowance	95
Total	1045

(type.name) expression // C notation
type.name (expression) // C++ notation

Ex:

```
average = Sum / (float)i; // C
average = Sum / float(i); // C++
```

Date.:
MON TUE WED THU FRI SAT



Arithmetic Operator
+, −, *, /, %.

Relational
Increement/Decrement
==, !=, >, <, <=, >=

Logical
&&, ||, !

Bitwise (Bit by Bit Comparison)
&, |, ∧, ∽, <<, >>

Assignment
=, +=, -=, *=, /=, %=, <<=, >>=,
&=, ∧=, |=

Conditional
(condition) ? x : y → ternary

Date.:
MON TUE WED THU FRI SAT

Increment / Decrement
++, --

Ex:

```
int a=21;
int c,
 c=a++;
cout<<c;        // c=21
cout<<a;        // a=22
 c=++a;
cout<<c;        // c=23
```

Function in C++

```
Void show(); // type fun-name (argu.);
 main()
{
  -----
  show(); // funⁿ call
  -----.
  ----
  return 0;
}
Void show() // funⁿ defi°
{


}
```

Date.:
MON TUE WED THU FRI SAT

ex: float volume (int x, float y, float z);
float volume (int, float, float);
float volume (int, float y, z);
volume (v, w, d);

Call by Reference

```
Void swap (int &a, int &b)
{
    int t = a;
        a = b;
        b = t;
}
swap (m, n);

Void show(int a);
 main()
{
   -----
   show(b);
   ----
}
Void show(int a)
{
   -----
}
```

Date.:
MON TUE WED THU FRI SAT

Return by Reference

```
int &max (int &x, int &y)
{
    if (x>y)
      return x;
    else
      return y;
}
```

Inline fun^n.

```
inline function header    fast execution
{
    function body
}
inline double cube (double a)
{
    return a*a*a;
}
```

```
#include<iostream>
using namespace std;
inline float mul (float x, float y)
{
    return (x*y);
}
inline double div (float P, float q)
{
    return (P/q);
}
```

Smehal Singh Maharia

Date.:
MON TUE WED THU FRI SAT

```
main()
{
    float a = 12.34;
    float b = 9.82;
    Cout << mul(a,b) << "\n";
    Cout << div(a,b) << "\n";
    return 0;
}
```

Default Arguments

```
float amount (float Principal, int period,
              float rate = 0.15);
Value = amount (5000, 7); // arg. missing
Value = amount (5000, 5, 0.12); // no arg. missing
```

Only trailing arguments can have default value

right-to-left

```
int mul (int i, int j=5, int K=10); // legal
int mul (int i=5; int j); // illegal
int mul (int i=0, int j, int K=10); // ill
int mul (int i=2, int j=5, int K=10); // le
```

Const Arguments

```
int strlen (const char *p);
int length (const string &s);
```

Date.:
MON TUE WED THU FRI SAT

```
Class x
{
  -----
  -----
  int fun1(); //member fun^n of class x
  -----
};
Class y
{
  -----
  friend int x::fun1(); // fun^n 1 of class x
  ----                    is friend of class y
};
```

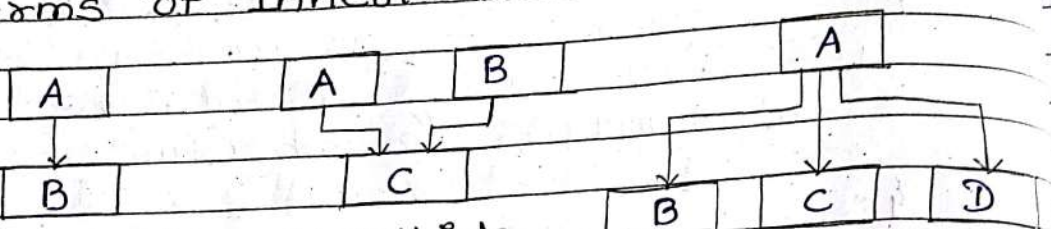
SYNTAX for friend class

```
Class z
{
  friend class x; //all member fun^n of
                    class x are friends
};                  to class z
```

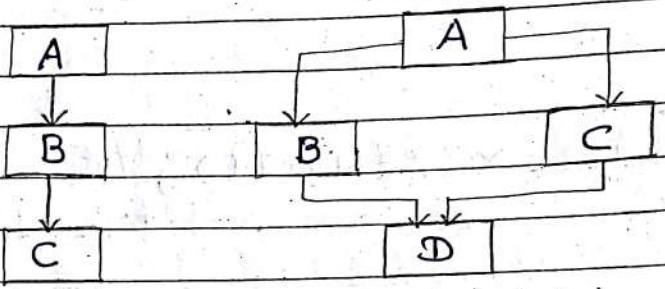
Date:
MON TUE WED THU FRI SAT

Unit - 4

Forms of Inheritance



a) single inher. b) Multiple inher. c) hierarchical inh.



d) Multilevel inheritance e) hybrid inheritance

The mechanism of deriving a new class from an old class is called inheritance. The old class is refered to as a base class, parent class or super class and new class is called derived class, sub-class or child class.

1) Single inheritance : The derived class with only one base class is called single inheritance.

2) Mutiple inheritance : A derived class with several base classes is called multiple base classes.

Date.:
MON TUE WED THU FRI SAT

3) Hierarchical inheritance: One class may be inherited by more than one classes this process is called hierarchical inheritance.

4) Multilevel inheritance: The mechanism of deriving a class from another derived class is known as multilevel inheritance.

Defining Derived classes

```
class derived-classname : Visibility-mode
                          base.-classname
{
    ----- //member of derived class
    -----
};
class derived : Private base // Private deri
{
    // member of derived class
};
class derived : Public base // Public deriva.
{
    //
};
class derived : base
{
    //
};
```

class A
private
Public

Date.:
MON TUE WED THU FRI SAT

```
#include<iostream>
using namespace std;
class B
{
    int a; //private; not inheritable
  public:
    int b;
    void set_ab();
    int get_a(void);
    void show_a(void);
};
class D : Public B //public derivation
{
    int c;
  public:
  void mul(void);
  void display(void);
};
Void B:: Set_ab(void)
{
    a=5; b=10;
}
int B:: get_a()
{
    return a;
}
Void B:: Show_a()
{
    Cout<<"a:"<<a<<"\n";
}
```

Date:
MON TUE WED THU FRI SAT

```
Void D :: mul()
{
    c=b * get_a();
}.
Void D :: display()
{
    cout << "a=" << get_a() << "\n";
    cout << "b=" << b << "\n";
    cout << "c=" << c << "\n";
}
int main()
{
    D d;
    d.set_ab();
    d.mul();
    d.show_a();
    d.display();
    d.b=20;
    d.mul();
    d.retur
    d.display();
    return 0;
}
```

Class D

private Section

c

Public Section

b

set_ab();

get_a();

show_a();

main();

display();

Date.:
MON TUE WED THU FRI SAT

Output

a = 5
a = 5
b = 10
c = 50
a = 5
b = 20
c = 100

```
#include<iostream>
using
class B.
{
    int a;
  public:
    int b;
    void get_ab();
    int get_a(void);
    void show_a(void);
};
class D : Private B
{
    int c;
  public:
    void mul(void);
    void display(void);
};
```

Date.:
MON TUE WED THU FRI SAT

```
Void B::get_ab(void)
{
    cout<<"
    cin>>a>>b;
}
int B::get_a()
{
   return a;
}
Void B::show_a()
{
   cout<<"a="<<a<<"\n";
}
Void D::mul()
{
    get_ab();
    c=b*get_a();
}
void D::display()
{
   show_a();
  cout<<"b="<<b<<"\n";
  cout<<"c="<<c<<"\n";
}
int main()
{
       Dd;
  //d.
  d.mul();
  //d.
   d.display();
```

Date.:
MON TUE WED THU FRI SAT

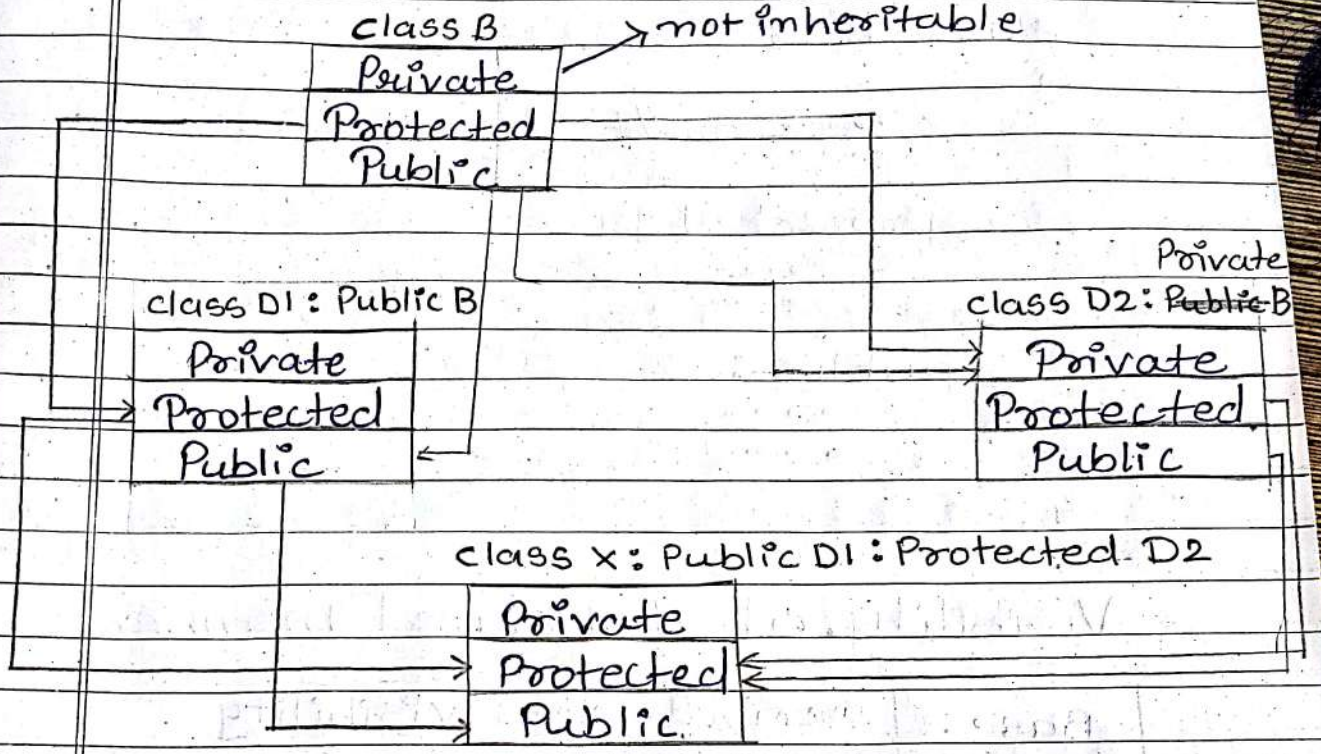
```
// d.b=20;
d.mul();
d.display();
return 0;
}
class D
Private Section
---- c ----
     b
get_ab();
get_a()
show_a();

class alpha
{
private:  // optional Visible to mem. fun^n within
.....          it's class.
.....

protected: // visible to mem fun^n of its own &
.....          that of derived class
.....

public:  // visible to all fun^n in the program.
.....
.....
};
```

Date.:
MON TUE WED THU FRI SAT




```
Class beta
{
    protectected :
        ....
        ....
    public :
        ....
        ....
    private :
        ....
        ....
    public :
        ....
        ....
};
```

Date.:
MON TUE WED THU FRI SAT

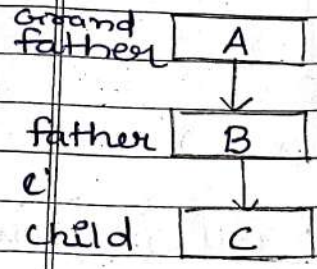
```
class beta
{
    .....    // by default private
    ....
protected :
    ....
    ....
public :
    ....
    ...
};
```

Visibility of Inherited member.

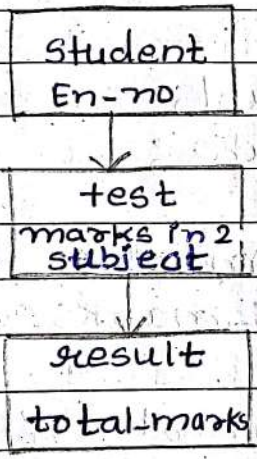
Base class visibility	Derived class visibility		
	Public derivation	Private derivation	Protected derivation
Private	Not inheri.	Not inheritable	Not inherita
Protected	Protected	Private	Protected
Public	Public	Private	Protected

Date.:
MON TUE WED THU FRI SAT

Multilevel Inheritance




```
class A {-----}; // base class
class B : Public A {-----}; // B derived from A
class C : Public B {----}; // C derived from B
```




```
#include <iostream>
using namespace std;
class Student
{
    Protected:
        int roll_number;
    Public:
        Void get_number(int);
        Void Put_number(void);
};
```

```
Void student :: get_number (int a)
{
        roll_number = a;
}
Void student :: Put_number ( )
{
   Cout << "Roll Number " << roll_number
                << "\n";
}

Class test : Public student // first level derivation
{
   Protected :
        float Sub1;
        float Sub2;
   public :
        Void get marks (float, float);
        Void put_marks (void);
};
void test :: get_marks (float x, float y)
{
        Sub1 = x;
        Sub2 = y;
}
Void test :: Put_marks ()
{
   Cout << "marks in Sub1 = " << Sub1 << "\n";
   Cout << "marks in Sub2 = " << Sub2 << "\n";
}
```

Date.:
MON TUE WED THU FRI SAT

```
class result : Public test // 2nd level derivation
{
    float total; // private
  public :
    void display (void);
};
Void result :: display (void)
{
    total = Sub1 + Sub2 ;
    Put_number() ;
    Put_marks() ;
    Cout << " Total = " << total ;
}
int main()
{
    Result student 1 ;
    Student 1. get_number (111) ;
    Student 1. get marks (75.0, 59.5) ;
    Student 1. display() ;
    Return 0 ;
}
```

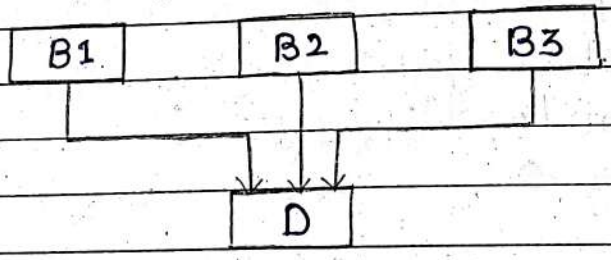
OUTPUT:

Roll Number 111
marks in Sub1 = 75.0
marks in Sub2 = 59.5
Total = 134.5

Members of result class after derivation:

```
Private:
    float total;
Protected:
    float sub1;
    float sub2;
    int roll_no;
Public:
    void display(void);
    Void get_marks(float, float);   } // from test
    Void put_marks(void);           }
    void get_number(int);           } // from
    Void put_number(void);          }
```

Mutiple Inheritance



Class D: Public B1, Private B2, Protected B3

```
#include <iostream>
using namespace std;
Class M
{
    protected:
        int m;
    public:
        void get_m(int);
```

```
};
Class N
{
    protected:
        int n;
    public:
        Void get_n(int);
};
Class P: Public M, Public N
{
    public:
        Void display(void);
};
Void M::get_m(int x)
{
    m=x;
}
Void N::get_n(int y)
{
    n=y;
}
Void P::display(void)
{
    Cout<<"m="<<m<<"\n";
    Cout<<"n="<<n<<"\n";
    Cout<<"m*n="<<m*n<<"\n";
}
```

```
int main()
{
    P r;
    r.get_m(20);
    r.get_n(10);
    r.display();
    return 0;
}
```

Output

m = 20
n = 10
m * n = 200

* Members of derived class P after inheritance.

```
protected:
    int m;
public:
    void get_m(int);
protected:
    int n;
public:
    void get_n(int);
public:
    void display(void);
```

Date.:
MON TUE WED THU FRI SAT

Ambiguity Resolution in multiple Inh.

```
class M
{
    public:
      void display(void)
      {
        cout << "class M \n";
      }
};
class N
{
    public:
      void display (void)
      {
        cout << "class N \n";
      }
};
class P : Public M, Public N
{
    Public:
    void display(void) // overwrite display()
                          of M & N
    {
      M::display();
    }
};
int main()
{
    P p;
    p.display();
    return 0;
}
```

Date.:
MON TUE WED THU FRI SAT

Ambiguity Resolution in Single Inh.

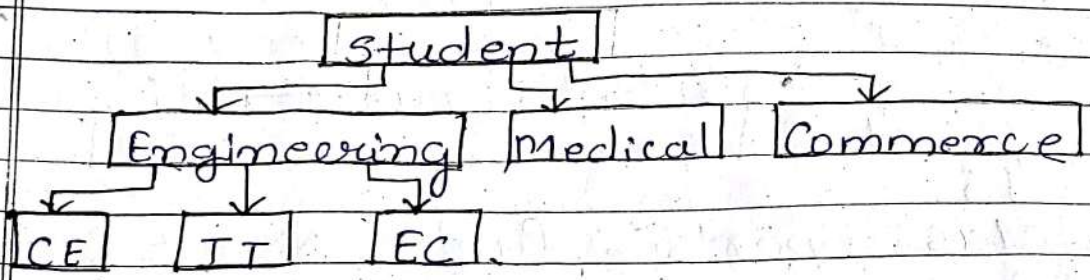
```
Class A
{
  Public:
    Void display()
    {
      cout << "A \n";
    }
};
Class B : Public A
{
  Public:
    Void display()
    {
      Cout << "B \n";
    }
};
int main()
{
    B b;                    // derived class object
    b.display();            // invokes display in B
    b.A::display();         // invokes display in A
    b.B::display();         // invokes display in B
    return 0;
}
```

Output:

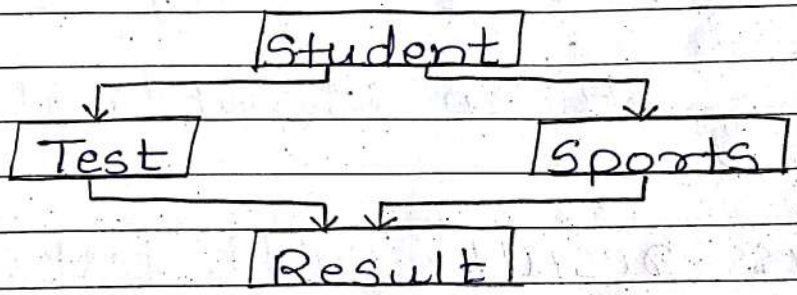
```
B
A
B
```

Date.:
MON TUE WED THU FRI SAT

Hierarchical Inheritance



Hybrid Inheritance

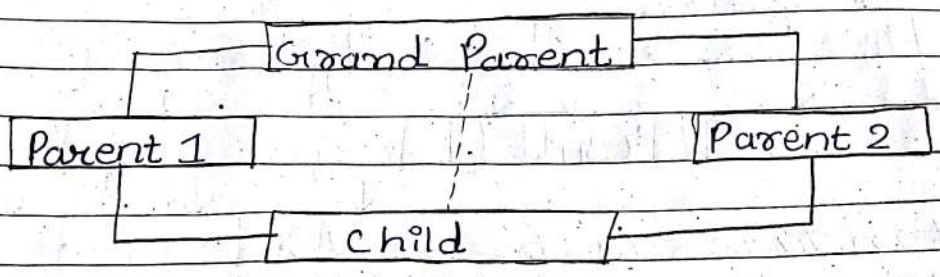



```
Class student
{   private : char name
    public :
        void getdata();
    {
        Cout << "Enter name";
        Cin >> name;
    }
        void putdata()
    {
        Cout << "Name is " << name;
    }
};
class test : Public student
```

```
{   public :
      void display()
      {
        student :: putdata();
      }
};
class sports : Public Result Student
{
      public :
        void display()
      {
        student :: putdata();
      }
};
class result : public test, Public Sports
{
      public :
        void display()
      {
        cout << name;
      }
};
int main()
{
      student S;
      S. display();
      S. student. getdata();
      S. test. display();
      S. sports. display();
      S. result. display();
      return 0;
}
```

Date.:
MON TUE WED THU FRI SAT

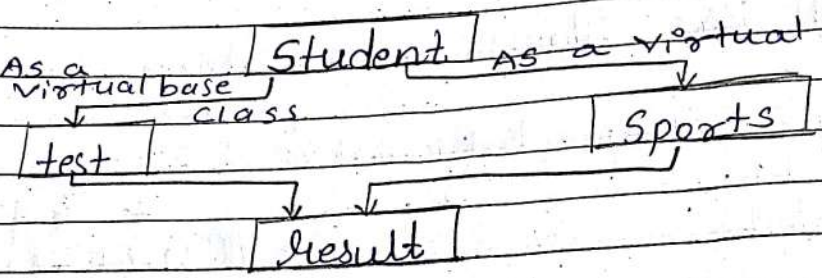
Virtual Base Class



Multipath Inheritance

```
Class A.                                    //grandparent
{
    -----
    -----
};
Class B1 : Virtual Public A  // Parent 1
{
    -----
    -----
};
Class B2 : Public Virtual A  //Parent 2
{
    -----
    -----
};
Class C : Public B1, Public B2  // child
{
    ----
    ---     //only one copy will be
            inherited from A.
};
```

Date.:
MON TUE WED THU FRI SAT




```
#include<iostream.h>
using namespace std;
class student
{
    protected:
        int roll_number;
    public:
        void get_number(int a)
        {
            roll_number = a;
        }
        void put_number(void)
        {
            cout << "Roll number :" << roll_number;
        }
};
class test : virtual public student
{
    protected:
        float Part1, Part2;
    public:
        void get_marks(float x, float y)
        {
            Part1 = x; Part2 = y;
        }
```

Date.:
MON TUE WED THU FRI SAT

```
    void put_marks (void)
    {
    cout << "marks obtained" << "\n"
        << "Part1 = " << Part1 << "\n"
        << "Part2 = " << Part2 << "\n";
    }
};
Class Sports : Public Virtual Student
{
    Protected:
        float score;
    public:
        void get_score (float s)
    {
        score = s;
    }
        void put_score (void)
    {
        cout << "Score" << score << "\n";
    }
};
class result : Public test, Public Sports
{
        float total;
    public:
        void display (void);
};
void result :: display (void);
{
    total = Part1 + Part2 + score;
    Put_number();
```

```
    put_marks();
    put_score();
    cout << " Total Score " << total << "\n";
};

int main()
{
    Result student1;
    student1.get_number(678);
    student1.get_marks(30.5, 25.5);
    student1.get_score(7.0);
    student1.display();
    return 0;
}
```

OUTPUT:

Roll Number : 678
marks Obtained
Part1 = 30.5
Part2 = 25.5
Score 7.0
Total Score = 63.0

What is the meaning of Inheritance in C++?

Describe the Syntax of Single inheritance.

When do we make a Virtual base class?

Explain hybrid inheritance with an example.

Date.: 19/9/18
MON TUE WED THU FRI SAT

↱ fun^n without implementation.

Abstract Class

Sometimes implementation of all fun^n cann't be provided in a base class, we don't know the implementation. Such class is called Abstract class.
For ex: shape base class, we cann't provide implementation of draw fun^n in shape. But we know every derived class must have implementation of draw fun^n.

Abstract class is a class which Contain atleast one pure Virtual fun^n. Pure virtual fun^n in C++ is a Virtual fun^n for which we don't have implementation, we only declare it. So, pure virtual fun^n is a virtual fun^n with no definition.

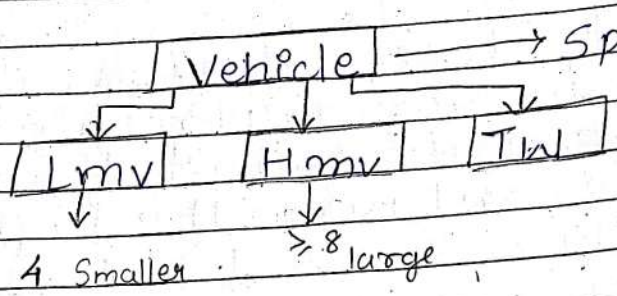
They start with virtual Key-word & ends with = 0.

Abstract class cann't be instant iated, but pointer & references of abstract class can be created.

Abstract class can be es have normal fun^n & Variable along with a pure virtual fun^n.

classes Inheriting an abstract class must implement all pure Virtual class, otherwise they will become abstract too.

Date.:
MON TUE WED THU FRI SAT




```
class Vehicle  // abstract base class
{
  private:
    data-type d1;
    data-type d2;
  public:
    virtual void Spec () = 0 // pure virtual
};
class Lmv : Public Vehicle
{
  public:
    Void Spec ()
    {
      // Lmv definition of Spec funn
    }
};
class Hmv : Public Vehicle
{
  public:
    Void Spec ()
    {
      // Hmv definition of Spec funn
    }
};
```

```
class Two : public Vehicle
{
    public :
        void spec()
    {
        // Two definition of spec fun^n
    }
};
```

Constructor in derived class

SYNTAX: Derived-Constouctor (Arglist 1, Arglist 2, ... ArglistN, Arglist D);
base1 (Arglist 1), base2 (Arglist 2), ... baseN (Arglist N).
{
    Body of derived constouctor
}

Example: D (int a_1, int a_2, float b_1, int d_1):
A (a_1, a_2), // call to Constouctor A
B (b_1), // call to Constouctor B
{
    $d = d_1$; // execute its own body
}

Date:
MON TUE WED THU FRI SAT

Method of inheritance	Order of execution

class B: Public A
{
};
 	
Class A: Public B,
Public C
{
};
 	
Class A: Public B,
Virtual Public C
{
};
	A(); base constructor
	B(); derived constructor
	
	B(); base (first)
	C(); base (second)
	A(); derived.
	
	G(); Virtual base
	B(); Ordinary base
	A(); derived

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

class alpha
{
    int x;
    public:
        alpha (int i)
        {
            x=i;
        cout<<" alpha initialized \n";
        }
    void show_x (void)
    {
        cout<<" x=" <<x<<"\n";
    }
};
```

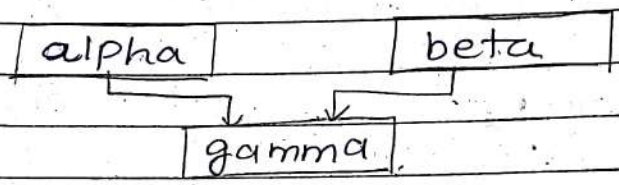
Date.:
MON TUE WED THU FRI SAT

```
class beta
{
    float y;
    public:
      beta (float j)
    {
        y=j;
        cout<<"beta initialized \n";
    }
      void show_y (void)
    {
      cout<<"y="<<y<<"\n";}
};
class gamma : public beta, public alpha
{
      int m,n;
      public:
gamma (int a, float b, int c, int d):
      alpha(a), beta (b) => not
{
        m=c;
        n=d;
      cout<<"gamma initialized \n";
}
    void show_mn (void)
{
      Cout<<"m="<<m<<"\n";
      Cout<<"n="<<n<<"\n";
}
};
```

on this base execution will be done

Scanned by CamScanner

```
int main()
{
    gamma g(5, 10.75, 20, 30);
    cout << "\n";
    g.show_x();
    g.show_y();
    g.show_mn();
    return 0;
}
```



Output

```
beta initialized
alpha initialized
gamma initialized
x=5
y=10.25
m=20
n=30
```

* Constructor (arglist) : initialization-Section
{
    argument - Section
}

Date.:
MON TUE WED THU FRI SAT

Ex:
```
Class xyz
{
    int a;
    int b;
    Public:
        xyz(int i, int j): a(i), b(2*j)
        {
        }
};
int main()
{
    xyz x(2,3);
}
```

→ xyz(int i, int j): a(i), b(a*j) { }
→ xyz(int i, int j): a(i) { b=j; }
→
```
xyz(int i, int j)
{
    a=i;
    b=j;
}
```

* Member classes Nesting of classes

Ex:
```
Class alpha {....};
Class beta {....};
Class gamma
{
    ------
    alpha a;
    beta b;
    ......
    ......
};
```

Date.:
MON TUE WED THU FRI SAT

```
class gamma
{
    alpha a;
    beta b;
    public:
    gamma (arglist): a (arglist 1),
                     b (arglist 2)
    {
        // constructor body
    }
};
```

Ex:

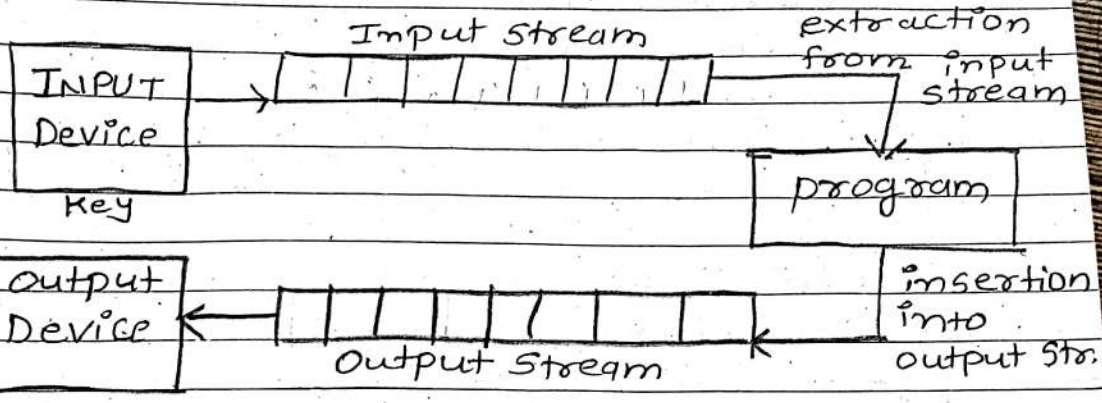
```
gamma (int x, int y, int z) : a(x), b(x,z)
{
    Assignment Section (for Ordinary other
                        members)
}
```

6 marks

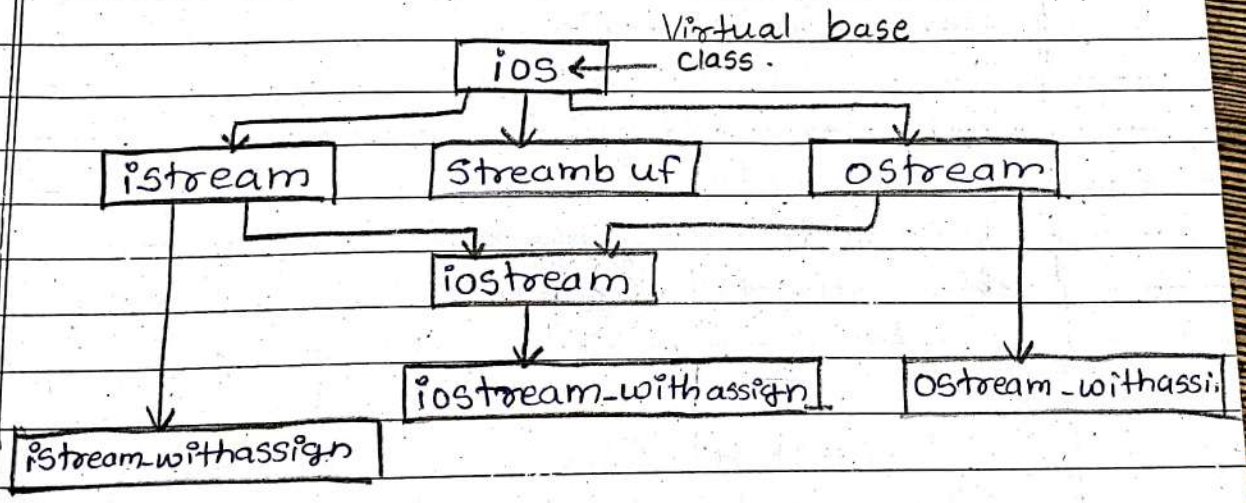
Date: 8/10/18
MON ☑ TUE ☐ WED ☐ THU ☐ FRI ☐ SAT

Chapter - 6

File Management



Stream classes



put() & get() fun^n

Ex:
```
char c;
Cin.get(c); // get a character from Keyb
            // and assign it to c
while(c!='\n')
{
    Cout<<c; // display the char. on screen
    Cin.get(c); // get another char.
```

Date.:
MON TUE WED THU FRI SAT

```
}
    cin >> c;
    cin.get(c);

get(void) → Version
-----
-----

    char c;
    c = cin.get();

    cout.put('x');

    cout.put(ch);

    cout.put(16);
```

Ex:

```
char c;
cin.get(c); //read a char
while (c != '\n')
{
    cout.put(c); //display the char on screen
    cin.get(c);
}
```

(Pg: 264 7e)

Ex:

Date.:
MON TUE WED THU FRI SAT

getline() and write() function

Syntax: cin.getline(line, size);

Ex: char name[20];
cin.getline(name, 20);

Cgpit UTU <Press return>
object oriented programming <Press return>
(, o/p

cin >> name

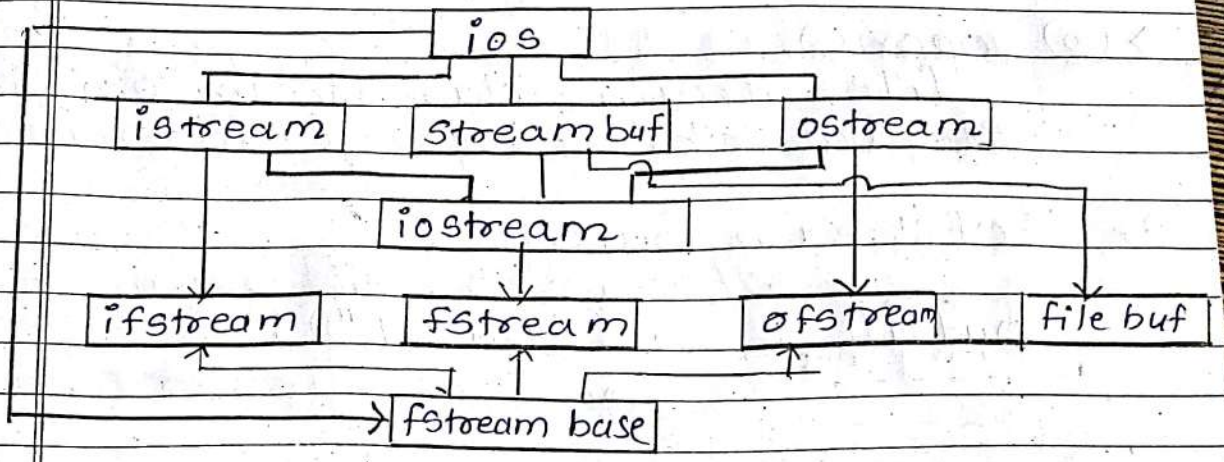
```
#include<iostream.h>
int main()
{
    int size = 20;
    char city[20];
    cout << "Enter city name \n";
    cin >> city;
    cout << "City name : " << city;
    cout << "Enter city name again \n";
    cin.getline(city, size);
    cout << " city name now" << city << "\n";

    cout << "Enter another city name \n";
    cin.getline(city, size);
    cout << " City name" << city << "\n";
```

Date:
MON TUE WED THU FRI SAT

Syntax:
Cout.write(line, size)

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    char *s1 = "C++";
    char *s2 = "Programming";
    int m = strlen(s1);
    int n = strlen(s2);
for(int i=1; i<n; i++)
{
    cout.write(s2, i);
    cout << "\n";
}
for(int i=n; i>0; i--)
{
    cout.write(s2, i);
    cout << "\n";
}
// concatenating strings
cout.write(s1, m).write(s2, n);
cout << "\n";
// crossing the boundary
cout.write(s1, 10);
return 0;
}
```

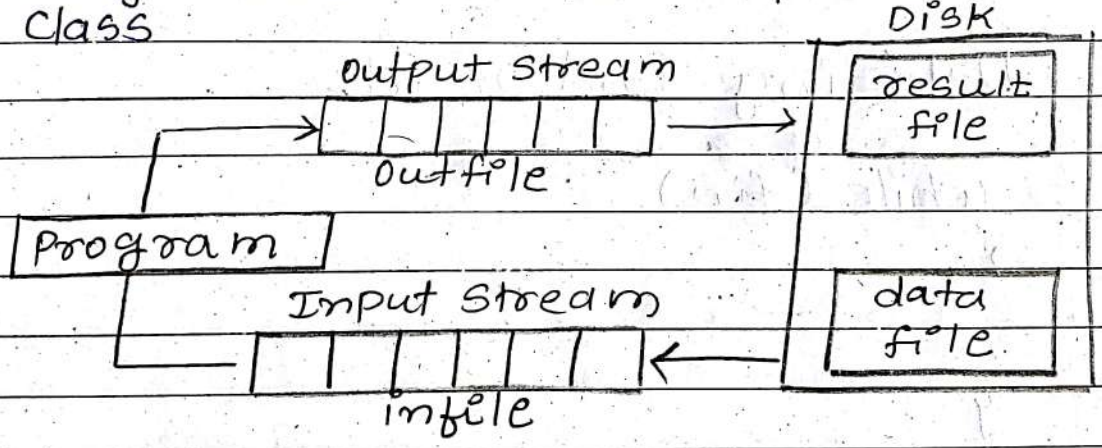


Opening a file

① Using the constructor function of the class

② Using member function open() of the class

→ ①




```
ofstream outfile ("result");
                    // o/p only
ifstream infile ("data");
outfile << "Total"; // i/p only
outfile << sum;
infile >> number;
infile >> string;
```

→ (2) open()
file-stream-class stream object
stream_object.open("file name");

Ex ofstream outfile;
// create stream for output
outfile.open("Data1");
----- // connect stream to Data1
-----
outfile.close();
// disconnect stream from Data1
outfile.open("Data2");
----- // connect stream to Data2
-----
outfile.close();
// Disconnect stream from Data

Detecting End of file

while(fin)
{

}

if(fin1.eof()!=0)
{ ↓ ios ↓ Non-Zero Value
}

Date.:
MON TUE WED THU FRI SAT

More about open() file mode

Stream-object.open("filename", mode);
read → ios::in ↳ purpose for which file is opened

input Stream → fin.open("result"); // by default read.

output Stream → fout.open("data"); // by default write.

file mode Parameters

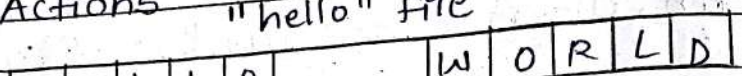
Parameters	Meaning
ios::app	Append to end-of-file
ios::ate	Go to end-of-file on opening
ios::binary	binary file
ios::in	open file for reading only
ios::nocreate	open fail if the file doesn't exist
ios::noreplace	open fail if the file already exist
ios::out	open file for writting only
ios::trunc	Delete the content of file if they exist

fout.open("data", ios::app|ios::nocreate

file pointer and manipulator

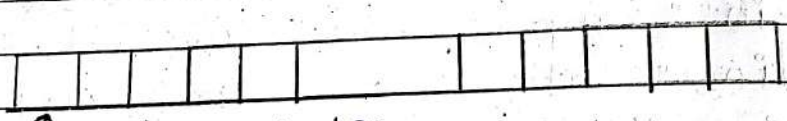
Default Actions "hello" file

open for Reading only



open in app. mode (for writting more data)

open for writting only



get input → Reading pointer.

(put) output → writing pointer

Scanned by CamScanner

Date.:
MON TUE WED THU FRI SAT

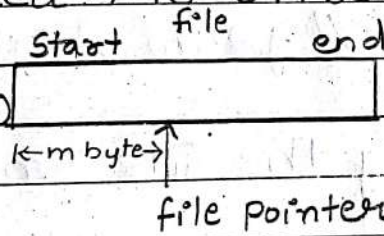
functions for manipulations of file pointers

seekg() → moves get pointer to a specified location
seekp() → move put pointer to a specified location
tellg() → give the current position of the gate pointer
tellp() → give the current position of the put pointer

→ 10th byte → start with zero

Ex: infile.seekg(10);
Ex: ofstream fileout;
fileout.open("hello", ios::app);
int p = fileout.tellp();

Specified the offset



seekg(offset, refposition);
seekp(offset, refposition);
ios::beg → start of the file
ios::cur → current Position of the pointer.
ios::end → End of the file

Ex: fin.seekg(m, ios::cur);
fin.seekg(-m, ios::end);

```
#include <iostream.h>
#include <fstream.h>
#include <string.h>

int main()
{
    char string[80]; string
    cout << "Enter a name";
    cin >> string;
int len = strlen(string);
  fstream file;
cout << "\n opening a file 'Text' and storing
 the string in it \n";
file.open("Text", ios::in|ios::out);
for (int i=0; i<len; i++)
{
   file.Put(string[i]);// Put a char to file
}
  file.seekg(0); // go to start
  Char ch;
  cout << "Reading the file content";
  while (file)
  {
     file.get(ch); // get a char from file
     cout<<ch; // display it on screen.
  }
  return 0;
}
```

Date.:
MON TUE WED THU FRI SAT

Output :
Enter a String
C++ Programming
Opening a file 'Text' and storing the string in it
Reading the file content
C++ Programming

Error Handling during file operation.

1. eof()

	Function	Return Value & Meaning
1.	eof()	Return true (non-zero) if end-of-file is encountered, otherwise return false (0).
2.	fail()	Return true when ip/op operation has failed.
3.	bad()	Return true if any invalid operation is attempted or any unrecoverable error is occured.
4.	good()	Return true if no error has occur.

```
#include <iostream>
#include <fstream>
#include <string.h>
#include <cstdlib>
using namespace std;
int main()
{
    int noc=0, now=0, nol=0;
    FILE *fr;
    char fname[20], ch;
    cout<<"\n Enter Source file name";
    gets(fname);
    fr= open(fname, "r");
    if(fr==NULL)
{
    cout<<"\n Invalid file name";
    exit(0);
}
ch=fgetc(fr);
while(ch!=EOF)
{
    if(ch!=' ' && ch!='\n')
        noc++;
    if(ch==' ')
        now++
    if(ch=='\n')
{
        nol++;
        now++;
}
```

```
    ch= fgetc (fp);
}
    fclose(fp);
    cout<<"\n Total No of chars"<<noc;
    cout<<"\n Total No of words"<<now;
    cout<<"\n Total No of lines"<<nol;
    cout<<
    return 0;
}
```