05/09/2020

Operator overloading in C++:

```
#
#
class Complex {
    private:
        int a, b;
    public:
        void showdata() {
            cout << "\n a" << "\n b";
        }
        void setdata(int x, int y) {
            a = x; b = y;
        }
        Complex operator + (complex c) {
            Complex temp;
            temp.a = a + c.a;
            temp.b = b + c.b;
            return temp;
        }
};
void main() {
    Complex c1, c2, c3;
    c1.setdata(4, 5);
    c2.setdata(6, 7);
=>  c3 = c1.add(c2);
    // c1 is called object which is
    calling its member function add
    & passing c2 as argument
=>  c3 = c1 + c2;
    or
    c3 = c1.operator + (c2);
    c3.showdata();   getch();
```

operator overloading

* We can overload operators which are function name in C.

Except ;

?: (Conditional)

sizeof()

6/9

Friend function in C++ with operator overloading

```
#  1
#        Complex
class  Imaginary {
        private :
            int a, b;
        public :
            void setdata (int x, int y) {
                a = x;  b = y;
            }
            void showdata ( ) {
                cout << "x = " << x << " y = " << y;
            }
            friend Complex operator + ( Complex, Com)
};
Complex operator + ( Complex x, Complex y)
    Complex temp;
    temp. a = x.a + y.a;
    temp.b = x.b + y.b;
    return (temp);
}

void main( ) {
    Complex
    Imaginary i1, i2, i3;
        i1. setdata ( 4, 5);          i3 = operator + (i1, i2)
        i2. setdata ( 6, 7);
        i3 = i1+i2   or   i1. operator + (i2)
        i3. showdata( ); getch( ); }
```

this pointer in C++

→ this is a keyword.
→ this is local (due to scope) object pointer jo har ek member function main hota hai or object ka address store karta h.
→ Jse hum modify nhi kar sakte h.
→ name conflict dur karta h yani instance member variable or local variable ko "this" keyword se differentiate karta h.

Eg.

```
#
#
class Rectangle {
    private:
        int l, b, h;
    public:
        void setvalue (int l, int b,
                       int h) {
            this -> l = l;
            this -> b = b;
            this -> h = h;
        }
        void showvalue () {
            cout << " l = " << l << " b = " << b << " h "
                 << h
```

Instead of using → we use → in this pointer

```
    }
};
void main()
{
    Rectangle r1, *p;
*p = r;     r1 -> setvalue(10, 20, 40);
            r1 -> showvalue();
            getch();
}
```

7/9/2020

SMA (Static Memory Allocation)
DMA (Dynamic ———)
↳ Creating memory for variables during runtime.

① new

Jis type ka variable bna rhe h usi type ka pointer bhi hona chahiye qki variable ka naam nhi hota. Eg.

```
int *p = new int;
Complex *ptr = new Complex;
```

Creating array:-

```
float *ptr = new float[];
```

getting i/p & creating array

```
int x;
cin>>x;
int *p = new int[x];
```

Qki a variable ka naam nahi hota h isilye ptr yani address se access krte hai

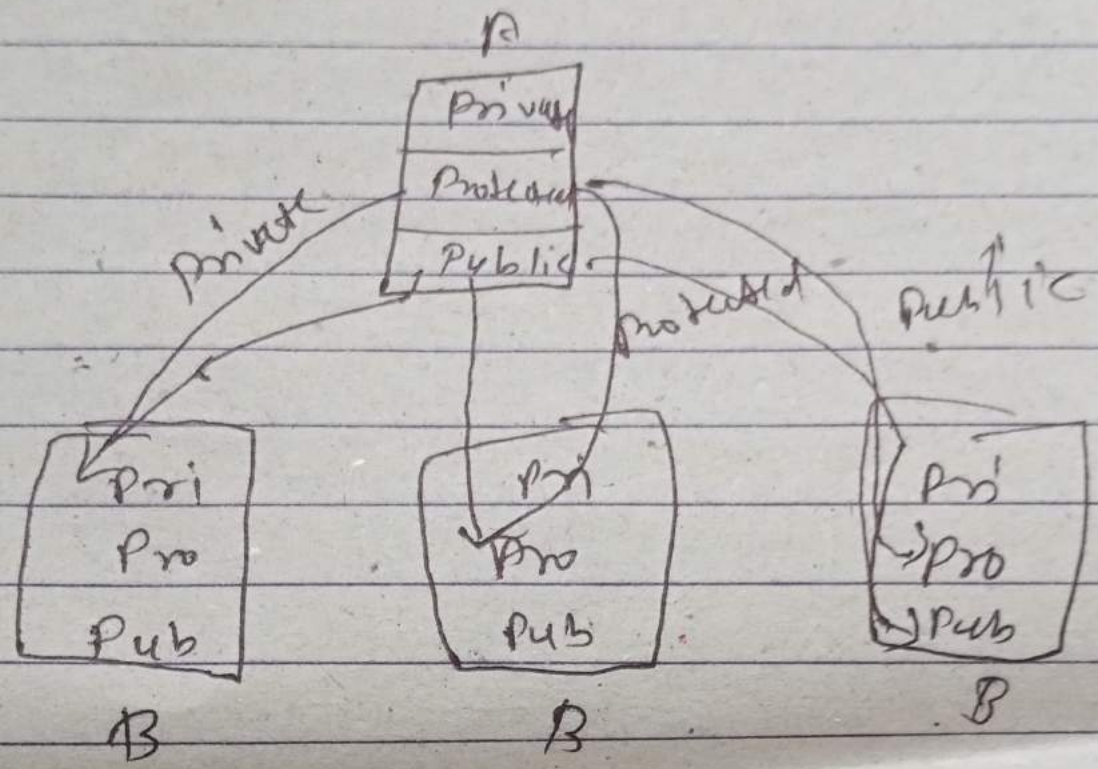
② delete

Iska use new ki madad se bne variable ko release karne ke liye hota h.

for variables ; delete p;
for array ; delete [] p;

19

Inheritance



private → public & pro. become private
protected → — " — become protected
public → public become public &
protected become protected

Eg.

```
class A {
        private:
                int x;
        public:
                void setvalue (int a) {
                        x = a;
};
class B : public class A
{
        private:
                int y;
        public:
                void setvalue (int b) {
                        y = b;
                }
};
void main() {
        A a1;
}
```

public
protected
private

7/9

Constructor & Destructor in Inheritance

```
//
//
class School {
        private:
            int a;
        public:
            School (int x){
                a = x;
            } ~ School(){
            }
};
class Teacher : public School {
        private:
            int b;
        public:
            Teacher (int y):
                School (K) {
            b = y;
            } ~ Teacher(){
            }
};
void main(){
    Teacher t1(2,5);
    getch();
}
```

Destructor

koi value ya K

Explⁿ:

Jis class ka object bnega uska constructor bnega for eg. Teacher class ke Obj. ka constructor phle bnega uske phle wo school class (parent) ke obj. ka constructor call krega. Lekin woh without parameter wala constructor krega, woh yani compiler. Agar hum parent constructor main arg. pass kr dete to hale class parent class ke constructor ko call krega to manually argument pass krna pdega.

Eg.

~~class~~

Teacher (int y) : School (int k)

↑ manually argument pass krna pdega agar parent class arg. accept krega to.

7/9

* (++)pre & post (++) increment operator overloading

```
#
#
class Example{
    private:
        int x;
    public:
        void setdata (int a){
            x = a;
        }
        void showdata ( ){
            cout << "x = " << x;
        }
        Example Operator ++ ()
        {
            Example i;
            i.x = ++x;
            return (i);
        }
        Example Operator ++(int)
        {
            Example i;
            i.x = x++;
            return (i);
        }
};
```

we have to just pass int blor at name conflict in both pre & post

```
void main ()
{   clrscr();
    Example i1, i2;

    i1.setdata (3);
    i1.showdata();
    i2 = i1++;        // post
    i2 = ++i1;        // pre
    i1.showdata.
    i2.showdata;
    getch()
}
```

or

i2 = i1.operator ++ ();

8/9/20 (Dombivali by Dharmendra)

Method overloading, overriding, hiding in C++.

```
#
#
class A
{
    public:
        void fun() { }
        void fun() { ... }
        void funny() { }
        void funny(int x) { }
};
class B : public class A
{
    public:
        void fun() { }
        void funny { }
        void funny(int x) { }
};
void main()
{
    ...
}
```

different code

① method overloading

② method overriding

③ method hiding

① Creating same function with same name & prototype except inside code difference is known as "method overloading" in same class or scope

(2) 1] Creating same function with same prototype (means dito copy) in derived class is know as "Method overriding"

2] Method overriding is used for increasing property / ability of ~~child~~ derived class's obj. for eg. Car is parent class & Supercar is derived class.

(3) Creating same function with name but passing argument in derived class function is known as "Method Oiding".

8/9

✱ virtual function in C++ (for late binding)

✱ Creating pointer of parent's class object & pointing derived class object is caused to early binding means if we want to run derived / base / child class ~~object~~ member function (must / may be method overriding) then it will be run parent's class function by seeing type of pointer not the content of pointer. In this case we have to use virtual function in parent class function. See following Example;

```
#
#
class A {
    public :
        virtual void fine () { }
};
class B : public class A {
    public :
        void fine () {}
};
```

No need to write virtual keyword here.

```
void main()
{
    clrscr()
    A a1; *ptr;
    B b1;

    ptr = &b1;

    ptr -> fune();

    getch();
}
```

9/9/2020

File Handeling in C++

Writing file:

must include

```
#include <fstream.h>
#
#
void main()
{
    ofstream fout;
    fout.open("myFile.dat");
    fout << "hello";
    getch();
    fout.close();
    getch();
}
```

creating obj. of ofstream class for file writing

for checking sir purpose

by default it will ios::out if u not putting any mode

Modes

ios::in - input / read
ios::out - output / write
ios::app - append
ios::ate - update
ios::binary - binary

eg. write - fout.open("my.txt", ios::out);
read - fin.open("my.txt", ios::in);

Binary - It will ignore space, newline, tab
text - It will not ignore

(you can put more than 1 modes)

★ Reading file (file must exist of given name)

```
#include<fstream.h>
#
#
void main()
{
    ifstream fin;
    fin.open("myfile.dat");
    char ch;
    fin>>ch;
    ch = fin.get();
    while(!fin.eof())
    {cout<<ch;
    ch=fin.get();
    }
    fin.close();
    getch();
}
```

obj. of class ifstream for reading existing file.

(by default ios::in)

store to get file

to get file

(It will ignore space, newline or tab) so,

It won't ignore space, newline, tab

to print on monitor

9/9

Initializers in C++

→ Initializers are used to initialize values of variable at just side of constructor by using colon. And the values of reference variable and constant variable too.

Eg.

```
#
#
class Example{
        private:
constant ——→ int x;
                const int a;
        public:
                                                  variable
            Example (int &a) : a(5),
        Example (int &n) : x(6),
                        a(5), y(n)
                Constant →
                                    ← reference variable
            {}
};
void main()
  {   int  m = 6
      Example e1 (m);
}
void fun ()
{
  Constant const int k = 10;
}
```

passing as reference & catch by int &n & store in y(n).

10/5 Lexically streampos tellp()
integer streampos tellg()

tellg & tellp in C++

① tellg - It is used for to point position in the ifstream (read).

② tellp - It is used to point the position in the ofstream (write) from start.

① tellg

```
#include<fstream.h>
#
#
void main(){
        ifstream fin;
        fin.open("MyFile.txt");
        int pos;
        char ch;

        pos = fin.tellg();
        cout << pos;
        fin >> ch; or ch = fin.get();
        post = fin.tellg();
        cout << pos;
        fin >> ch;
        get ch();
        fin.close();
}
```

② tellp()

```
#include<fstream.h>
#
#
void main()
{
    ofstream fout;
    fout.open("abc.txt", ios::app);
    int pos;
    pos = fout.tellp();
    cout<<pos;
    fout << "rahul";
    pos = fout.tellp();
    cout << pos;
    getch();
    fout.close();
}
```

It will not erase previous contents

12/9/2020

Seekg & seekp in C++ :-

(1) Seekg() - It is used to access the position of character randomly in input / Reading (ifstream)

Syntax:-

(1) istream& seekg(streampos pos);
(2) istream& seekg(streamoff off, ios_base::seekdir way)

(2) seekp() :-

It is used to access the position of character randomly in output / writing (of stream).

Syntax :-

(1) ostream& seekp(streampos pos);
(2) ostream& seekp(streamoff off, ios_base::seekdir way);

where,

i) pos, is new absolute position within the stream (relative to beginning for seekp & from ending to seekg)

ii) off, is offset value, relative to way parameter

iii) way, values ios_base::beg, ios_base::cur. & ios_base::end.

(v) seekp()

```
#include <fstream.h>
#
#
void main() {
        ofstream fout;
        fout.open("Myfile.dat", ios::out);
        int pos;
        pos = fout.tellp();
fout << "Hello
Rahul";
        cout << pos;
        fout.seekp(2, ios-base::beg);
        cout << fout.tellp();
        fout.close();
        getch();
}
```

(2) seekg();

```
#include<fstream.h>
#
#
using namespace std;
int main(){
    ifstream fin;
    fin.open("myfile.txt");
    cout << fin.tellg();
    cout<<"\n" << (char) fin.get();
    // otherwise it will print ASCII

    cout << "\n" << fin.tellg();
    fin.seekg(6);
    cout<<"\n" << fin.tellg();
    cout<<"\n"<<(char) fin.get();
    getch();
}
```

12/9

Abstract Class in C++

- Any class which contains one or more pure virtual function is known as "Abstract class". Eg. virtual void fun() = 0;
- Unlike Java, C++ donot use Abstract keyword.

Why we use Abstract class?

If we create a parent class as Person and child class as Teacher & student if we count that student class doesn't access some of the functn of teacher class or vice-versa like salary, questionpaper etc. That's why we use "Abstract class".

- We can't create object of an Abstract class. To use other members of abstract class we have to use child class objects and also we have to virtual keyword bcoz in pointer, parent class pointer points to parent class members not child class member (virtual function). And also

we have to override that method in child class too, but without making it pure virtual function.

Eg.

```
#
#
class Person {
        public:
            virtual void fun() =0;
        // pure virtual function
            void f1() { }
        // other members
};
class Student : public Person {
        public:
            void fun() { }
        // Method overriding without
        making it pure virtual function
};
void main() {
        clrscr();
        ~~Person~~ Student s1;
            s1.f1();
        getch();
}
```

10/9/2020

Shallow & Deep Copy in C++

(1) Shallow copy:

i) When we copy members of one class to other class dicto" then it is "Shallow Copy"

ii) It can achieve by 2 methods

a) Copy constructor -
when you initialize at same time like Dummy d1 = d2;

b) Assignment operator :-
When you initialize at next line like Dummy d1;
d1 = d2;

(2) Deep Copy :-

i) When we can copy members of one class but it also contains pointer which refers any outer resources to other class then it is known as "Deep Copy"

ii) It also can achieve by above 2 methods like copy constructor & assignment operator

Eg. Shallow Copy :

```
#
#
class Dummy {
    private:
        int x, y;
    ~~public~~:
    public:
        void setdata (int a, int b)
        { x = a; y = b; }

        void showdata () {
        cout << "a = " << a << " b = "
                << b;
        }
};
void main () {
    clrscr();
```

Shallow Copy (a) → `Dummy d1 = d2;`

(b) → `d1 = d2;`

```
    d2.showdata();
    getch();
}
```

(2) Deep Copy

→ Deep Copy main hume manually Copy constructor banana pdta h pointer ke liye.

```
#
#        Dummy
class ~~Doppter~~ {
        private:
            int a, b;
        ~~public~~:
            void ~~setdata~~
        public:
            Dummy ( ){
                int p = new int;
            }
            void setdata (int x, int
                y, int z) {
                a = x; b = y; *p = z;}

            void showdata ( ){
                cout<<" a="<<a<<" b="<<b ;}
            Dummy (Dummy &d) {
                a = d.a; b = d.b;
                p = new int;
                * p = *(d.p);
                                    ← d1 ka p
            }
        };
```

Default constructor → Dummy ( ){

Creating pointer variable dynamically → int p = new int;

Copy Constructor → Dummy (Dummy &d) {

we can also write assignment operator

```
void main()
{
    clrscr();
    Dummy d1;
    d1.setdata(3,4,5);
    Dummy d2 = d1;

    d2.showdata();
    getch();
}
```

12/9 If the working of function & prototyping is same just datatype is diff. then we can use it function template.

Template in C++

making functions & classes more generalized.

(1) function Template
(2) Class Template

(1) **function Template** (a.k. Generic function)

Syntax : template < class Type >
Type fun-name (Type a, Type b) {

where Type = placeholder

Eg,

```
template <class X>
X big (X a, X b) {
    if (a>b)
        return(a);
    else
        return (b);
}
```

P.T.O

```
int main() {
        cout << big(4, 5);
        cout << big(5.6, 7.9);
        return 0;
}
```

* If we want to pass more than 2 class in function template. Eg:

```
template <class X, class Y>
X big(X a, Y b)
```

& we can also pass diff. datatype of argument Eg:

```
int big(int x, float y)
```

like

```
X big(X x, Y y)
```

13/9/2010

by Constructor

Type Conversion - (1) primetive to non-primetive

If we want to copy or convert primetive type (int, float) to non-primetive (class) then we have to "Type conversion". We have to use Constructor.

Eg.

```
#
#
class Conversion {
        private:
            int a, b;
        public:
            void setdata (int x, int y) {
                a = x; b = y;
            }
            void showdata () {
            cout << "a =" << a << "b =" << b; }
            Conversion () { }
            // default constructor here
            we are not passing any values.

            Conversion (int k) {        - (2)
                a = k; b = 0; }
};
```

```
void main() {
        Conversion c1;  — ①
        int ~~x = 5~~;                    int x = 5;
        c1.~~setdata(x, 5)~~;
        c2.Showdata();
        c1 = x;   // Conversion
        c1.Showdata();
        getch();
}
```

⇒ We have to create default Constructor bcoz we are not passing any value — ①

⇒ We are creating Constructor for type Conversion. — ② 1st page

13/3

by operator Casting

Type Conversion ② Non-primitive to primitive

- We have to use operator Casting.

Syntax: operator datatype ()

{

return (type-data);

}

Eg.

```
#
#
Class Conversion {
        private:
            int a,b;
        public:
            void setdata (int x, int y){
                a=x ; b=y; }
            void Showdata () {
            cout << "a=" << a << "b=" << b;
        }
//Operator Casting
        operator int () {
            return (a);
        }
};
```

```
void main()
{
    Conversion c1;
    int x;
    c1.setdata(4,6);
    c1.showdata();
    x=c1;        // Conversion
    cout<<"\n x=" << x;
    getch();
}
```

O/P = a = 4 b = 6
x = 4

13/9

Type Conversion - ③ non-primitive to non-primitive

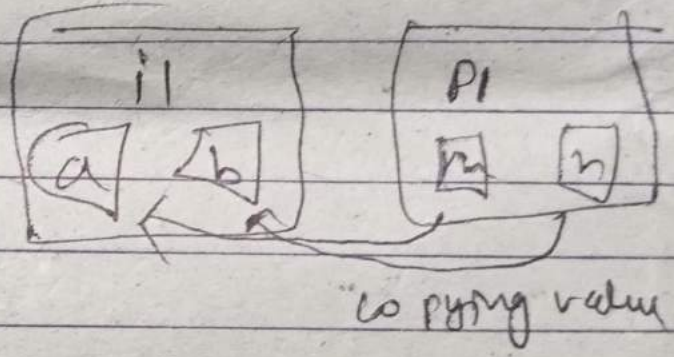
- We can convert one class to other class by 2 methods -
  ① Constructor ② operator casting

Eg.

```
Item i1;
Product p1;
i1 = p1;   then
```

1st we have to use constructor inside item class (left side of equal) & 2nd method operator casting inside product class (right side of equal)

① by Constructor




```
#
#
class Product {
    private:
        int m, n;
    public:
        void setdata (int x, int y){
            m = x ; n = y;
        }
        ~~void showdata ()~~
        int getM(){
            return m; }
```

// to access m & n in item class bcz they are private

* we are pulling an copying values of m & n variable of product class to variable a & b of item class.

```
        int getN() { return n; }
};

class Item {
        private:
                int a, b;
        public:
                void Showdata() {
        cout << "a=" << a << "b=" << b;
        }

        Item() { }.   // default Constructor

        Item (Product p){ // accepting args
                                of product
        a = p.m; // It'll give error
                        bcoz they are private
                        in product class, so
        a = p.getm();
        a = p.getN();
};

void main(){
        Item i1;
        Product p1;
        p1.setdata(3,4);
        i1 = p1;   // Conversion
        i1.Showdata();
        getch();
}
```

14/9

Exception Handeling

Try Catch throw

-> Catch must be declared just after try block.

-) We can declare Catch as many as we want

-) We can declare throw or function which containg throw statement at place of throw.

-) We can't declare try without Catch & vice-versa.

-) We can declare throw even without try Catch.

#
#

```
using namespace std;
int main()
{
    int i = 3;
    Cout << "welcome";
```

```
void fun(){
    throw 10;
}

try {
    if (i==1)
        fun(1);
```

```
try {
    if (i==1)
        throw 1
    if (i==2)
        throw 2
    if (i==3)
        throw 3
```

we throw without no. or even we fun(n which contains throw statement

```
    cout << "\n In try";
    // It executes only after above
all condition false.
}
catch (int e) {
        cout << "Exception no." << e; }
catch (double e) {
        cout << "Exception no." << e; }
        or
catch (...) { // to catch all types
        cout << "Exception";
}
cout << "\n Last line";
getch();
}
```

14/09/2020

Dynamic Constructor

18/9/2020

⇒ Dynamic Constructor doesn't mean that we are creating a constructor with 'new' keyword.

⇒ It means object handling it's member pointer variable which refers to the obj of the class.

⇒ or Constructor ka dynamic, object ka dynamic handel hone wali memory ka creation "dynamically" "Dynamic constructor" kehlata h.

Eg. #

```
using namespace std;
class A{
    private:
        int a,b;
        int *p;   // pointer
    public:
        A(){      // constructor
            a=10; b=0;
            p= new int;
        }
```

```
A(int x, int y, int z){
    a=x; b=y;
    p= new int;
    *p = z;
}

int main(){
    A o1, o2, o3(3,4,5);
    o1.
}
```

14/9

Virtual Destructor

18/9 - ⇒ Destructor ka use memory release krne ke liye hota jab "Memory leak" ke problem se bchi hai.

⇒ Jab koi child class ko destructor ko chalana ho tab hum "virtual destructor" ka use krte h. Aki agar parent class ka pointer h to woh parent class ko point krega jisse parent class ka destructor child class ke destructor se pehle chal jayega. or Destructor main child class destructor pehle parent class ko call krta par pehle child class) ka destructor chalata h.

Eg.

```
#
using namespace std;
class A {
        int a;
        public:
            void f1() {}
```

```
    virtual ~A(){}      // Destructor
};
class B: public A {
        int b;
        public:
            void f2(){}
            ~B(){}      // no need to write virtual
};
int fun();      // Declaring
int fun(){      // Defining

    A *p = new B;
    p->f1();    // correct
    p->f2();    // error (if u don't mention virtual keyword).
    delete p;
}
int main(){
    fun();      // calling
}
```

14/9

* We can also create unnamed namespace like:-

namespace namespace{
}

* namespace is container of identifiers.
* it has same syntax as class but not ; at end. Syntax:-

namespace namespace_name{

}

It can't be instantiated.

* We can create variables, functions even classes in namespace.
* We can also declare namespace in different header file.
* When we create 2 namespaces with same name then it will merge / extend.
* namespace can also use for name-conflict issue. We can declare same name variable but in different namespace.
* We can use alias of namespace.

Eg:

namespace myspace{
}

namespace ms = myspace;

* It must be declared globally. We can't declare it inside function.

g.

```
#
using namespace std;
// It contains cout in namespace std.
namespace mypace {
        int a;      // variable
        int f1();   // function
        class A {   // class
            public:
                void fun();
        };
}
int mypace::f1(){                    // function
    cout << " Hello f1";             // calling
}
int mypace::A::fun(){                // class's
    cout << " Hello fun()";          // function
}                                    // accessing

using namespace mypace;
int main(){
        a = 5;      // accessing variable
        f1();       // We don't need an
                    // necessary to use
        A obj;      :: like,
        obj.fun();  mypace::f1();
    getch();        when we are calling.
}
```

19/9

Nested class

- class inside a class is called "Nested class".

Eg.

```
#
using namespace std;
class Student {
        private:
                int rollno; char name[20];
// nested class  class Address {
in private            private:
member of                int house no;
class student            char street[20];
                         char city[20];
                         char state[20];
                         char pincode[20];
                     public:
void setAdd (int h, char *s, char *c,
        char *st, char *p) {
        house no = h;
        strcpy (city, c);
        strcpy (state, st);
        strcpy (street, s);
        strcpy (pincode, p);
}
```

```
void showAdd (){
        cout << "\n" << houseno << endl;
        cout << street << " " << city;
        cout << " " << pincode << endl;
        cout << state;
    }
};
    Address add;        // obj. of Address class
public:
    void setRollno (int r){ rollno = r; }
    void setname (char *n){
        strcpy (name, n); }
    void setAdd (int h, char *s,
            char *c, char *st, char *p)
    {
        add.setAdd (h, s, c, st, p);
    }

    void showStudent (){
        cout << "student data " << endl;
        cout << rollno << " ";
        cout << name << " ";
        add.showAdd ();
    }
};

int main (){
    student s1;
    s1.setRollno (288);
    s1.setname ("Rahul");
    s1.setAdd (301, "Bhivani Road",
            "Amsterdam", "462033",
            "Holland");
    s1.showStudent ();
}
```

* Note :- We are not creating nested class obj. in main function since it is private inside student class.

(2) Template class : template <class type>
class classnames
type a = 15;

(8/9/2020)

STL

=) STL (Standard template library)
=) It is powerful set of template classes
=) Structure -

(1) Containers -

=) (1) Containers are used to manage collections of objects of a certain kind
=) (2) Containers library in STL provide containers that are used to create data structures like arrays, linked list, trees, etc.
=) (3) These container are generic, they can hold elements of any data type.
Eg. vectors - used to dynamic array.

(2) Algorithms :

=) (1) Algo. act on containers. They provide the means by which you will perform initialization, sorting, searching, and transforming of the contents of containers.

=> (1) Algorithms library contains built in functions that perform complex algorithms on the data structures.

Eg. reverse(), sort(), binary-search()

* Algo. library provides abstraction. i.e. you don't need to know how the algo. works.

(3) Iterators :- (Just pointer)

=> (1) Iterators are used to step through the elements of collections of objs. These collections may be containers or subset of containers.

=> (2) Iterators in STL are used to point to the container.

** => (3) Iterators actually acts as a bridge betn containers & algorithms.

Eg. Sort() algorithm have two parameters starting iterator and end iterator.

Eg.

vector<int> :: iterator it = v1.begin();

8/9 * Containers in STL

=) Containers library is ~~conta~~ collection of classes.

=) The containers are implemented as generic class templates.

=) It helps to implement and replicate simple & complex data structures very easily like "arrays, list, tree, associative array (index are not numeric eg. map), etc.

=) Containers can be used to hold different kind of objs.

Eg. Common Containers :-

i) vector : replicates arrays
ii) queue : replicates queues
iii) stacks : replicates stacks
iv) priority_queue : replicates heaps
v) list : replicates linked list
vi) set : replicates trees
vii) map : ~~replicates~~ ~~associate~~ associative arrays.

* Classifications of Containers

(1) Sequence - like array, linked list etc.

Sorted DS like

(2) Associative Containers — like map, set, etc.

(3) Unordered Associative Container
— Unsorted DS

(4) Containers Adapters —

Interface to sequence Containers.

★ How to Containers library?

=) Include header file e.g. list,

```
# include <iostream>
# include <list>
int main(){
    list<int> mylist;
}
```

use constructor to initialize it like mylist(...);

Arrays in STL

=) Array is a linear collection of similar elements.

=) Array container in STL provides us the implementation of static array (fixed size).

=) Use header file.

#include <array>

=) Syntax:

type size

array<int, 4> array-name = {11, 22, 33, 44}

Member functions of array template

1. - at (give value at that index)
2. - [] (subscriptive operator) (same as above)
3. - front() (give 1st index value)
4. - back() (" last ")
5. - fill() (fill all indexes with same value)
6. - swap() (swap 2 arrays of same type & size)
7. - size() (to give size of array)
8. - begin()
9. - end() } (begin() return iterator pointing 1st element of array while end() return pointer iterator pointing last element of array)

Eg. #

```
#include< array >
using name space std;
int main() {
//creating array => array< int, 5> my_array = {11, 22,
                                              33, 44, 55};
```

① — cout << my-array.at(2);

② — cout << my_array[5];

③ — ——— . back();

④ — ——— . front();

⑤ ——— . fill(10);

```
for (int i=0; i<=5; i++)
    cout << my-array[i];
```

⑥ array<int, 5> my-array2 = {1, 2, 3, 4, 5};

my-array.swap(my-array2);

just change my-array2[i];

⑦ cout << my-array2.size();

⑧ using iterator;

⑨ array<int,5>::iterator it = my_array.begin();

or

my-array.end();

15/9/2020

Pair in C++ (not part of container)

=) It is also template library class but not part of container.

=) In this we can pass 2 arguments / values / datatypes which are related to each other.

Syntax:

```
pair <string, int> pair-name;
```

=) We must have to include,

```
#include <pair>
```

=) We can create pair by obj. of pair class,

```
pair <string, int> pl;

pl.make-pair("Rahul", 20);
```

=) To access / print,

```
pl.first;   // for 1st value

pl.second;  // for 2nd or last value
```

(Here we have only 2 values in pair)

=) We can even pass non-primitive datatype like class. eg

```
pair <string, student> pl;
```

Q. #

```
#include <pair>
int main()
        pair <int,
class student {
        private:
                int age; string name;
        public:
                void setdata(int a, string s){
                    age = a; name = s; }
                void showdata(){
                cout << "name " << name;
                cout << " Age " << age; }
        };

int main(){
        pair < string, int > p1;
        pair < string, student > p2;
        p1.makepair("Shivam", 26);
        //for non-primitive
        student s1;
        s1.setdata("Amit", 20);
        p2.make_pair(1, s1);
        cout << "Pair 1";
        cout << p1.first << " " << p1.second;
        cout << "Pair 2";
        cout << p2.first << " " ~~<< p2.second~~;
        student s2 = p2.second;
        s2.showdata();
}
```

15/9

(5) ☆ Tuple in C++ (not part of container)

☆ => It is also template library class but not the part of container.

=> In this we can pass 3 arguments / values / datatypes which are related to each other.

=> Syntax:

```
tuple < string, int, int > tuple_name;
```

=> we must have to include in header;

```
# include < tuple >
```

=> we can create tuple by object of tuple class;

```
tuple < string, int, int > t1;
t1.make_tuple("Imran", 84, 25);
```

=> to access / print, we have to use

```
get < 1 to 3 > (tuple obj);  like
```

```
cout << get<0> (t1);
cout << get<1> (t2);
cout << get<3> (t3);
```

} i.e. there are only 3 values.

=> we can even pass non-primitive datatype like class;

```
tuple < string, int, student > t1;
```

Eg. #

```
#include<tuple>
class Employee {
        private:
                int salary; string Ename;
        public:
                void setdata(int x, string e){
                    salary = x; Ename = e;}
                void show data(){
                    cout << "Employee name" << Ename;
                    cout << "Salary" << salary;
                }
        };
int main(){
        tuple<int, string, Employee> t1;
        Employee e1; // object creation.
        e1.setdata(20000, "money");
        ~~t1.make_tuple(20000, "Money",~~
        t1.make_tuple(1, "Singh", e1);
        cout << get<0>(t1);
        cout << get<1>(t1);
        ~~cout << get<2>(t1);~~ Employee e2 =
                                get<2>(e1);
        e2.showdata();
}
```

here we are accessing class

(6) Relational operator works on vectors, tuple, pair etc.
↳ ==, !=, >, <, >=, <=

18/9

Vector in STL

→ The most general purpose container is the vector. It supports a Dynamic Array.

→ Array v/s vector
- Array size is fixed, no flexibility to grow or shrink during execution.
- Vector can provide memory flexibility

→ Syntax:

```
vector <int> v1;
```

It doesn't need size bcoz it's dynamic.

```
vector <string> v {"Sumrah", "Gupta"};
```

Initialization during declaration.

→ More vector Declaration
- `vector <int> v1;` - zero length vector
- `vector <char> v2(5);` - 5 elements of char vector
- `vector <char> v3(4, 'a');` - Initialize 4 elements with 'a'

→ Member function on vector template

(1) - push_back() - insert elements at last
(2) - pop_back() - remove
(3) - at() - return value at given index
(4) - [] - subscript operator
(5) - capacity() - return capacity
(6) - size() - return size
(7) - front() - give 1st element of vector
(8) - back() - give last
(9) - insert (with iterator) (10) clear

#

```
# include < vector >
using namespace std;
int main() {
        vector < int > v1;
        vector < int > v2 (3);
        vector < int > v3 (5, 10);   // 5 blocks of value 10
        vector < int > v4 (10, 20, 30, 40, 50);
(3) -   Cout << v4.at (2);
(4) -   Cout << v4[3];
(1) -   Cout << v4.push-back (60);
(2) -   Cout << v4.pop-back (60);
(5) -   Cout << v4.capacity;
(6) -   Cout << v4.size;

        Cout << v1(0);
        for (int i=0; i<=9; i++)
            v1.push-back(10 * (i+1));

(10) -  v1.Clear();
        Cout << v1.size();
        for (int i=0; i<v1.size(); i++)
            cout << endl << v1[i];
(7) -   Cout << " First value " << v1.front();
(8)     cout << " last -> " << v1.back();
(9)     vector < int > :: iterator it = v1.begin();
        v1.insert (it+2, 45);
```

to insert values randomly using insert

18/9 ①

* List in STL

=) List class support a bi-directional list
=) Vector support random access but a list can be accessed sequentially only.
=) List can be access front to back or back to front.

* Creating List object :-

① list <int> i1;
② list <int> i2 {11, 22, 33, 44};
③ list <string> l3 {"Delhi", "Dhaka", "Holland"}

* Member function of List class

① sort() - to sort string in alphabatical order
② size() - to return size
③ push_back() - to insert value at end
④ push_front() - — " — at ~~back~~ starts
⑤ pop_back() - to remove value at end
⑥ pop_front() - — " — at starts
⑦ reverse() - to reverse
⑧ remove() - to
⑨ clear() - to create the whole list

** [] operator doesn't work in list

Eg. #
#include <list>
using namespace std;
int main(){
list<int> l1{1, 2, 3, 4};
list<string> l2{"India", "Iran", "Pak"};
① - l2.sort();
② - cout << l1.size();
③-④ { l2.push_back("Nepal");
l2.pop_front("India");
⑤ - l1.remove(4);
⑥ - l1.clear();
list<int>::iterator it = l1.begin();
while (p != l1.end()){
cout << *it << " ";
p++;
}
cout << " total values are:" << l1.size();
⑦ - l1.reverse();
}

18/9 (8)

☆ map in STL

=) maps are used to replicate associative array (Index are not no.)

=) It contains key-value pair. Each key is unique & cannot be changed.

☆ Syntax:

```
map< int, string> map-name;
```

eg.

```
#
# include<map>
using name space std;
int main(){

    map< int, string> customer;

// Initializing at new line
    customer [100] = "Manoj";
    customer [150] = "Arjun";
    customer [200] = "Vishal";
    customer [555] = "Honey singh";
    customer [777] = "Aniket";
```

or

```
map<int, int> m1{{1,10},{2,58},
// Initializing                {6,75},{3,73}};
at same
time.   map<int, string> :: iterator p =
                                  customer.begin();
Iterator
        while (p != customer.end()){

// print values
  key or of second   cout << p -> second << endl;

                     p++;
        }

        cout << customer.at(555);
        cout << customer.size();
        cout << customer.empty();

        // To insert value with key using
           pair,

        customer.insert(pair<int, string>
                        (203, "Abhishek"));
        cout << customer.size();

}
```

18/9 * => we can use operators on string like, ==, !=, >, <, >=, <=, =, +, [ ]

* String Class in C++

char s1[10];

=> we can't perform C++ operators like concat, comparison on "char array".

=> null-terminated character arrays are not technically data type.

=> So, string class introduced.

* String class:

=> Since creating class in C++ is creating new data type, string is derived data type.

=> String class is derived class of template class 'basic_string'

=> We can use C++ operators on string

* Constructors of string class:

1. string()
2. string (const char *str)
3. string (const string & str)

eg.

```
#
# include <string>
using namespace std;
int main(){

Different ways for creating string
        string s1 = "Hello";
        string s2("Hello");
        string s3 = string("Hello");

// by 2nd constructor of string => char str[] = "Hi";
        string s4(str); // or string s4=str;

        string s5 = s2; // or string s5(s2);
// by 3rd constructor of string
// we are passing s2 refrence in s5.
we are not using &s2 because formal argl. not pointer.

        string s6 = s1+s2;

        or
        ~~s6 += s1+s2~~;
        s6 += s1; // s6 = s1+s6;
}
```

18/9

(2) String in STL

⇒ U can mix string obj. with another string obj. or C style string.

⇒ Cff string can also be concatenated with character constant.

* Useful methods

(1) - assign()
(2) - append()
(3) - insert()
(4) - replace()
(5) - erase()
(6) - find()
(7) - rfind() / reverse find
(8) - compare()
(9) - c_str()
(10) - size()

Eg. of mixed string, one object must be string,

```
#
#include <string>          //without .h
using namespace std;
int main()
{
    string s1 = "Hello";
```

```
char str[] = "Students";
string s2 = s1 + str[];
cout << s2;
string s3;        for append   string s3 = "Hello";
① -  s3.assign("Hola!"); cout << s3;
② -  s3.append("students");
string s4 = "Namaste";
③ -  s4.insert(2, "123");   // 123 at index 2
④ -  s4.replace(3, 2, "A");  //(length, index, value)
⑤ -  s1.erase();
string s5 = "Hello Online students";
⑥ -  int i = s5.find("online");
cout << i;
⑦ -  int i = s5.rfind("students");
cout << i;   // to find in reverse ord
string s6 = "Amit";
string s7 = "Aman";
⑧ -  int v = s6.compare(s7);
cout << v;
// c style
⑨    string s8 = "Hello";
     char str[20];
// copy s8 to str char   strcpy(str, s8.c_str());
     cout << str;
⑩ -  int i = s8.size();
     cout << i;
```