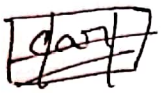
Sharif tech (CPP) — T-02

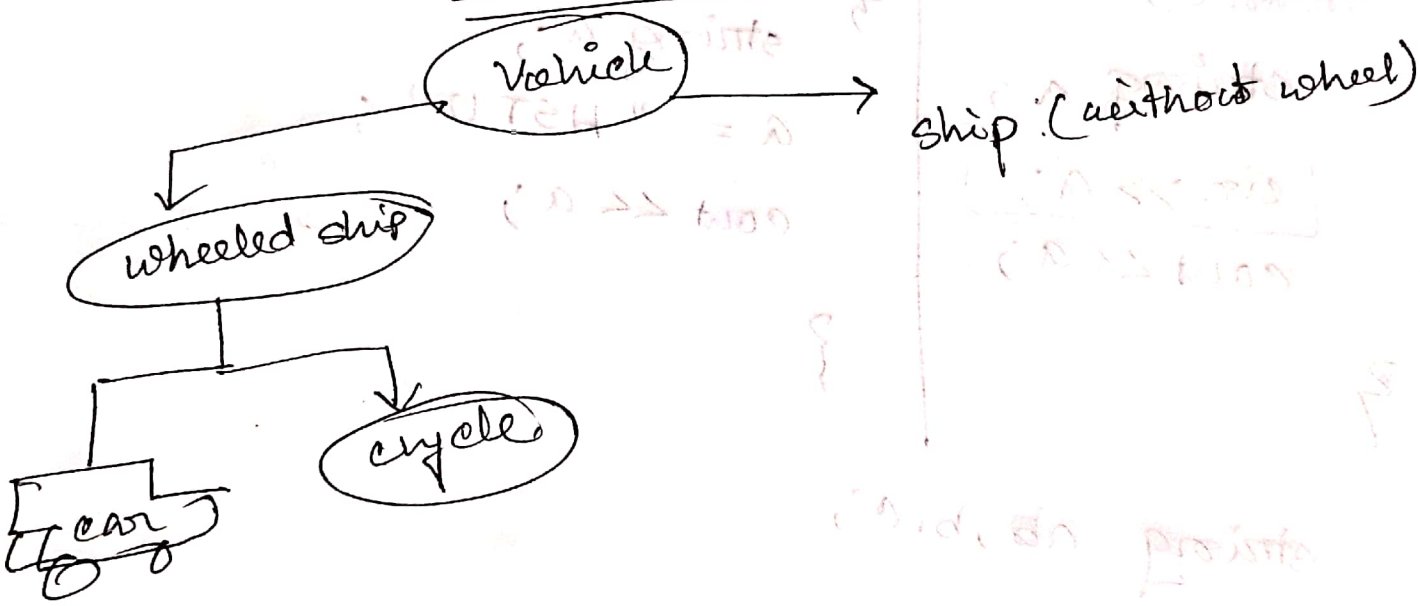
What is an object? → প্রতিবস্তু
→ class (C language)



OOP → characteristics → class

1. Encapsulation (একসাথে থাকা) → variable and methods একসাথে থাকা
2. Polymorphism → একই জিনিস একাধিক ভাবে use করা।
3. Inheritance (একটি জিনিস থেকে একাধিক জিনিস হওয়া) → উপরে আগের কিছু change করা।

Inheritance



T-03 → Compiler কি এবং কয়টি ও কি কি?

T-04 → The starting

```
#include<iostream>   → input output stream.
using namespace std;  → no specifier needed
int main()            → string class has
{
    int a;
    cin>>a;
    cout<<a;          → console output
    return 0;
}
```

```
main()
{
    string a;
    cin >> a;
    cout<<a;
}
```

```
main()
{
    string a;
    a = "HSTU";
    cout << a;
}
```

space 3 টা দিলে

```
string ab, b, c;
a= "HSTU";
b = " DINAJPUR";
c = a+b;
cout << c;
```

→ HSTU DINAJPUR

PTO.

on cout << a << b << endl; → new line → standard

→ on '\n' on "\n"

cout << "HSTU" << " , " << DINAJPUR << " " << 5200 << endl;

A character can be added to a string:

a = "HSTU"

b = "DINAJPUR"

a = a + ' ';

cout << a << b << endl; → HSTU DINAJPUR

```
#include<iostream>
#include<algorithm>
using namespace std;
int main()
{ int arr[] = {1,2,3,4,5,6,7,};
  for(i=0; i<7; i++)
    cout << arr[i] << " ";
  cout << endl;
  sort(arr, arr+7);
```

→ copy and paste

```
  return 0;
}
```

T-05 (First program)

cpp তে যদি c এর header file আমরা use করতে চাই তাহলে সেগুলোর File গুলো include করার একটা standard নিয়ম আছে.

```
#include<cstdio>
#include<cstring>
```

but `#include <cmath>` এর বদলে `#include<math.h>` লেখা উচিত না

`<cmath>` এ `pow()` ও problem হয়।

T-06 ~~Diffe~~ Difference between c and cpp

```
main()
{  int a;
   a = 5;
   int b;
}
```

→ c এর rules অনুযায়ী ভুল → cpp তে allowed যখন খুশি variable declear করতে পারব. ☺

(#)

```
main()
{
   for (int i = 1; i <= 5; i++) {
      int b = i+1;
      cout << b << endl;
   }
   cout << b << endl;
}
```

`cout << b << endl;` → ERROR করবে ভিতরে declare করার কারণে a, b এর scope শুধু loop এর ভিতরে থাকবে।

int i(5); → cpp তে এভাবে declare করা যায়

bool → datatype → determine true or false
L 0

```
{ bool i=0;
  cout<<i<<endl; ⟶ i=0
}
```

```
{ bool i=true; → 5 → i=0 otherwise i=1.
  cout<<i<<endl; ⟶ i=1, i=0
```

CPP ⟶ KEYWORD ⟶ 63 +

```
struct abc{
  int a;
};
main()
{
  abc ob;   → This kind of declaration is allowed in CPP.
  ob.a=5;
  cout<<ob.a<<endl;
}
```

T-06 +8 → class intro → like structure
↳ data, function এর combined রূপ

Class intro

object class এর প্রকারি রূপ

```
#include<iostream>
using namespace std;
class rectangle {
public:
  int height;
  int width;
};
int main()
{
  rectangle obj;
  obj.height = 5;
  obj.width = 6;
  cout << "Area = " << obj.height * obj.width;
  return 0;
}
```

creating object.

cin >> obj.height >> obj.width ;
→ এভাবেও input নেওয়া যায়।

Ex: মানুষ যদি একটি class হয় তাহলে আমি class এর object. মানুষ যা বিভিন্ন রকমের তার আকার আছে কিন্তু আকার বিভিন্ন মান আকার থেকে আকার বলে আমরা আকার থেকে আকার we are in same class but different object.

E-31: Reference Variables

— আগে একটি memory location এর জন্য 2টি name use করতে পারবো। & — use করতে হয়।

```
int a=100;      → reference variable
int &ref = a;   a এর বদলে ref use করতে পারব।
```

∴ c = a+b; } → both same
c = ref+b;

* pointer variable null ও হতে পারে কিন্তু reference variable null হয় না। কারণ যখন ref var. declare করা হয় তখনই এটা initialize করা হয়।

* pointer এর array বানানো যায় কিন্তু ref var. এর array বানানো যায় না।

```
int main()
{ int a=100;
  int &ref=a;
  cout << a;      → 100
  cout << ref;    → 100
  cout << &a;     ┐
  cout << &ref;   ┘→ same address
  return 0;
}
```

এখানে ref এর মান change করলে a এর মান change হবে. আবার a এর মান change করলে ref এর মান change হবে।

proof next page.

S-021 References

```
int main()
{ int i;
  i = 5;
  cout << i << endl;        → 5

  int *p;
  p = &i;
  cout << *p << endl;       → 5

  int &ro = i;
  cout << ro << endl;       → 5

  ro = 7 /// i = 7
  cout << i << endl;        → 7
  cout << *p << endl;       → 7
  cout << ro << endl;       → 7
  return 0;
}
```

Tanver Likhon

E-32 Call by reference

```
void swap (int &, int &);
int main ()
{ int a,b;
  cin>> a>>b;
  cout <<a << b;
  swap (a,b);
  cout<< "Aftrer swaping";
  cout<< a <<b;
  return 0;
}
```

Before

ref. variable

```
void swap (int &x, int &x)
{ int temp;
  temp = x;
  x = y;
  y = temp;
}
```

→ মেইনের ভিতরে ও only swap (a,b) call করছি swap করা হয়েছে।

T-09 MEMBER FUNCTION → class এর member function

(1)

```
class rectangle {
public:
int height;
int width;
int area()
{ return height * width;
}
};
```

(2)

```
class rectangle {
public:
int height;
int width;
int area(); };

int rectangle :: area()
{
return height * width;
}
```

rectangle → class name

:: → scope resolution

but process slow

memory বড় করে

Solution next tutorial

same

```
int main()
{ rectangle obj;
  obj.height = 5;
  obj.width = 6;
  cout << "Area = " << obj.area() << endl;
  return 0;
}
```

NOTE: একটা class এ অনেক Function থাকতে পারে। এ জন্য class এর মধ্যে Function এর prototype declare করে class এর বাইরে function লিখতে পারি। এতে সুবিধা— class এর ভিতরে fun dec. করলে obj তৈরী করার আগে obj function এর যতটুকু জায়গা দরকার করে নিচ্ছে আগে জায়গা নিয়ে নিবে। অনেক obj তৈরী করলে সে অনেক জায়গা নিবে কিন্তু prototype declare করলে সে অসম্পূর্ণ জায়গা নিবে না। যখন fun call হবে তখন জায়গা নিবে।

T-10

Inline Function

Inline function যখন কম্পাইল হয় তখন যেখানে function কে তাড়াতাড়ি call করার জন্য compiler যেটা ② করে: process টি সম্ভব হয় না। তবে সেক্ষেত্রে inline function ব্যবহার করা হয় না, যেমন নিচের function টি call করা হয় তখন class টি inline ব্যবহার করা হয় না। যখন ছোট ছোট function মধ্যে দ্রুত call করার প্রয়োজন পড়ে তখন inline ব্যবহার করতে হয়

```
class rectangle {
public:
int height;
int width;
int area(); };
inline int rectangle:: area()
{ return height * width; }
```

③

T-11 private/public/protected — class

↳ Access modifier.

class → member গুলো class এর কোন Access করতে পারে।

→ আগের last example এ height, width And area() main function কোন Access করতে পারে।

⊗ class এর মধ্যে public: private: protected: না দেওয়া হলে তাহলে by default সেটা private হয়ে যাবে।

Accessing private through public

```
class rectangle {
private:
    int height;
    int weight;
public:
    void set(int h, int w);
    int area(); { };
int rectangle :: area()
{ return height * weight; }
void rectangle :: set(int h, int w)
{ height = h; width = w; }
```

↳ set() private এ Access করতে পারে বলে class এর বাইরে থাকতে পারছে।

```
main()
{
    rectangle obj;
    obj.set(5,7);
    cout << "Area = " << obj.area()
    << endl;
}
```

Alternate version next

```
class rectangle {
  private:
    int height;
    int width;
  public:
    int area (int h, int w);
};
int rectangle :: area (int h, int w)
{ height = h;
  width = w;
  return height * width;
}
main()
{ rectangle obj;
  cout << "Area = " << obj.area(5,7) << endl;
}
```

T-12 → Variable Scope: → Age ; a ponchilam

tai.

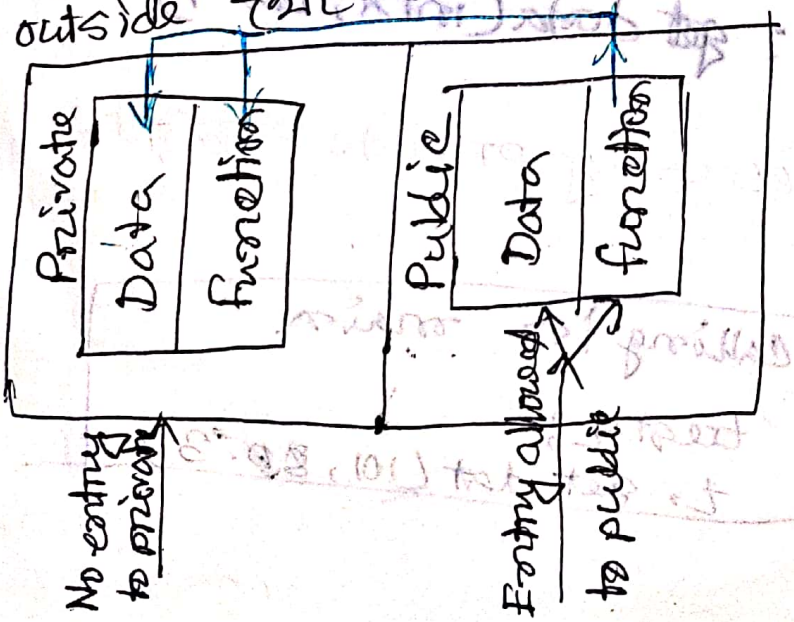
T-13 → Constructors and destructor → data + function

ET4Y - 39 → Refining Class and Creating Object

```
class class_name
{
  private:
    Data declaration;
    function declaration;
  Public:
    Data declaration;
    function declaration;
};
```

ivate এর member কে শুধুমাত্র class এর member access করতে পারে, তাই usually private এর ta দ্বারা হয় এবং Public এর মাধ্যমে ভিতরে দ্বারা হয়।

public এর মাধ্যমে দ্বারা Private এর Data ess করা হয়, And public এর member কে class outside থেকে access করা হয়।



Defining member function:

(i) Inside the class definition
(ii) Outside the class definition

An example having both

```
class test
{ int roll;
  float per;
  public:
    void getdata(int, float);
    void putdata(void)
    { cout << "roll = " << roll;
      cout << "Per = " << per; }
};
```

return_type classname :: function_name (Para)
{
body
}

Outside defining

```
void test :: getdata(int x, float y)
{
  roll = x;
  per = y;
}
```

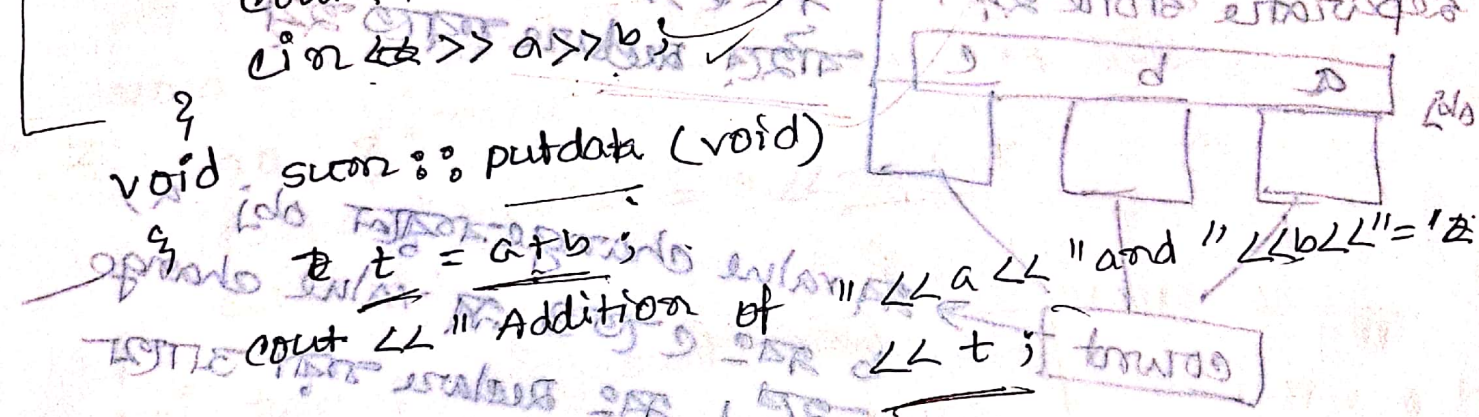
accessing private members

Calling in main.

```
test t;
t.getdat(101, 80.3)
```

A simple program using class and Object T-42 E

```
#include<iostream.h>
using namespace std.
# class sum
{ int a,b,t;
public:
    void getdata (void);
    void putdat (void);
};
void sum::getdata (void)
{
    cout<< "Enter the value of A & B";
    cin<<>> a>>b;
}
void sum::putdata (void)
{
    t = a+b;
    cout << "Addition of "<<a<< "and "<<b<<"="
    << t;
}
main()
{
    sum obj;
    obj.getdata();
    obj.putdata();
    getch();
    Return
}
```

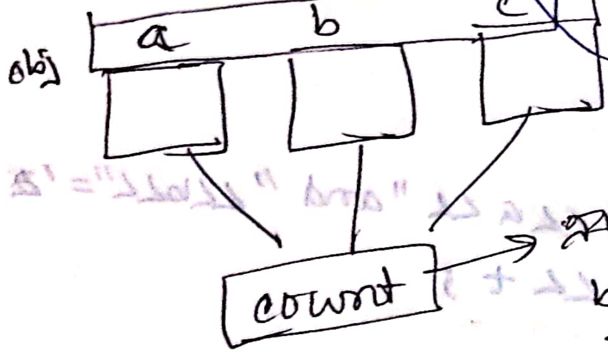


⊞ Static Data Members in C++ [T-43] E

```
class bca
{  int no;
   static int count;
   -- --.
};
int bca :: count;
```

কোন class থেকে যত গুলো obj বানাইনা কেন তার মধ্যে যদি একটি static member থাকে তাহলে এটি এর obj এর আলাদা হবে না। মানে obj এর আলাদা এর obj এই member এ আলাদা আলাদা value রাখতে পারবে না। এর obj এর ক্ষেত্রে এই static member এর value same হবে। এবং এ এ member কে class এর বাইরে declare করতে হয়।

এবং এটি memory তে separate store হয়।



এর value change করলে obj a, b এবং c তেও এর value change হবে। এবং declare করার আগে এর value 0 হয়ে যায় [যদি কোন value assign না করি]

```
int bca :: count = 10;
```

এভাবে initial value দেওয়া যায়।

A program using static data member

```
#include <iostream>
using namespace std;
class test
{
    int no;
    static int count;
public:
    void getval(int);
    void dispcount(void);
};
void test::getval(int x)
{
    no = x;
    cout << "Number = " << no;
    count++;
}
void test::dispcount(void)
{
    cout << "Counter = " << count;
}
int test::count;
main()
{
    test t1, t2, t3;
    t1.dispcount();
    t2.dispcount();
    t3.dispcount();
    t1.getval(100);
    t2.getval(200);
    t3.getval(300);
}
```

→ প্রথমবার তিনটি object কে call করে তখন নিচের প্রথমবার obj এর count এর value দুটোর কি আসে

→ 3 times 3 object এর no এর call initialize হবে এবং প্রতিবার count এর মান increase হবে 1 করে definition ।

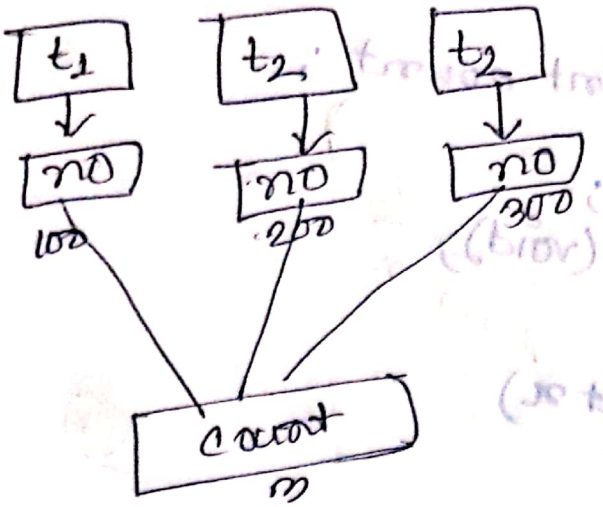
```
t1.dispcount();
t2.dispcount();
t3 dispcount();
```

output : 0 0 0 / 100 200 300

3
3
3

→ अलग count same

}



E.T - 44 Static member function

~~only~~ a static member function can access only a static member. अन्य किसी member को नहीं

```
class test
{ int no;
  static int count;
public:
  void setno(void);
  void dispno(void);
  static void dispcount(void);
};
```

```cpp
void test :: setno(void)
{
    no = ++count;
}
void test :: dispno(void)
{   cout << "object number " << no;
}
~~static~~ void test :: dispcount(void)
{
    cout << " Counter " << count;
}
int test :: count;

main()
{   test t1, t2;
    t1.setno();   // n-1, c-1
    t2.setno();   // n-2, c-2
    ~~t2.dispcount(~~
    test :: dispcount();   // -2
                           // t2.dispcount
    test t3;
    t3.setno();   // n-3, c-3
    test :: dispcount();   // -3
                           // t3.dispcount
    t1.dispno();   // -1
    t2.dispno();   // -2
    t3.dispno();   // -3
}
```

static function [illegible] call [illegible] member variable [illegible]

static [illegible] declare [illegible]

[illegible] static function [illegible] call [illegible]

ET-45 Passing Objects as Function Argument

```
class test
{ public:
      void disp(test);
};
void test :: disp(test t)
{
}
```

```
class time
{ int h, m;
  public:
     void gettime(int, int);
     void puttime(void);
     void sum(time, time);
};
```

→ আমরা class কেও built in datatype এর মত ব্যবহার করতে পারি। So function এ [illegible] call করার সময় call করবো। এখানেও তাই like datatype — time [class]

```
void time :: gettime(int x, int y)
{  h = x; m = y;
}
void time :: puttime(void)
{
   cout << " Hours = " << h;
   cout << "\n minutes = " << m;
}
void time :: sum(time t1, time t2)
{  m = t1.m + t2.m
   h = m/60;
   m = m % 60;
```

h = h + t1.h + t2.h;

}

```
main()
{ time t1, t2, t3;
  t1.gettime (3,40);
  t2.gettime (4,30);
  t3.sum(t1,t2);
  t1.puttime();
  t2.puttime();
  t3.puttime();
}
```

Run on ~~CB~~ but main Online compiler

class BCA
{
}

class DCA
{
}

ET-46: Friend Function:

Friend function:

— একই-সাথে দুটি (একাধিক) class এর data ~~access করা~~ use করা যাবে। ~~2 class দ্বারা~~

— 2 ক্লাস দ্বারা share হয়।

— non member function. ~~কিন্তু~~ যে class এর data use করতে হবে সেখানে friend keyword declare করতে হবে।

এর আগে declare করতে হবে।

— Public/Private both এ করা যাবে।

— declared outside the class like ~~gen~~ general function declaration.

- যেকোন object দ্বারাই call করা যায়।
- no scope resolution needed because it is not a member of a class
- argument হিসেবে class এর object use করা হয়।

Example

```
void dsp (BCA b, DCA d)
{
    b.roll;
    d.roll;
}
```

যেহেতু এটি কোন class এর member নয় তাই obj দ্বারা access করতে হবে।

One Program

```
class A;          → forward declaration
                  → declared not defined
class B
{   int b;
    public:
    void getval (int x)
    {   b = x;
    }
    void putval (void)
    {   cout << b << endl;
    }
};
```

— PTO.

class name

friend void add (A, B);

};

class A

{ int a;

public:

void getval (int x)

{ a = x;

}

void putval (void) { cout << a << endl; }

friend void add (A, B);

};

void add (A ob1, B ob2)

{ cout << "addition of A & B" << ob1.a + ob2.b;

}

main ()

{ A a;

B b;

a.getval (100);

b.getval (200);

a.putval ();

b.putval ();

add (a, b); normal function এর মতো call.

}

Scanned by CamScanner

5 - 20 Friend function

```
class rectangle {
    int height;
    int width;
public:
    void set (int a, int b)
    { height = a; width = b; }
    int area () { return height * width; }
    friend class cost;
};

class cost {
    int costRate;
public:
    void setvalue (int a) { costrate = a; }
    int total cost (rectangle A) {
    return costRate * A.height * A.width;
    }
};
```

friend class cost; — cost class rectangle class এর private data কে access করতে পারে

public করলে access করতে পারে

class কে object passing

```
main()
{ rectangle r;
  r.set(5,6);
  cost c;
  c.setvalue(100);
  cout << "area(r) = " << r.area() << endl;
  cout << "Total cost = " << c.totalcost(r) << endl;
}
```

now I want to declare total cost function as friend function.

```
class cost; → forward declaration.
class rectangle
{ int height;
  int width;
public:
  void set(int a, int b)
  { height = a; width = b; }
  int area() { return height * width; }
  friend int totalcost(rectangle, cost);
};
```

class as datatype

```
class cost
{ int costRate ;
 public :
   void setvalue(int x)
   { costRate = x;
   }
 friend int totalcost (rectangle, cost) ;
};

int totalcost(rectangle r, cost c)
{
 return r.height * r.width * c.costRate ;
 or return r.area() * c.costRate;
}

main()
{ rectangle r;
  r.set(5,6);
  cost c;
  c.setvalue(100);
  cout << " Area = " << r.area() << endl;
  cout << " Total cost of the are = " << totalcost (r,c) << endl;
```

passing two objects of the two class.

ET-47: Returning Objects

```
class test
{   int a, b;
    public:
        void getval (int x, int y)
    {   a = x; b = y; }
    friend test sum (test, test);
    void dispval (test);
};
test sum (test t1, test t2)
{   test t;
    t.a = t1.a + t2.a;
    t.b = t1.b + t2.b;
    return t;
}
void test :: dispval (test t)
{   cout << " Value of A " << t.a;
    cout << " Value of B " << t.b;
}
main()
{   test A, B, C;
    A.getval (10, 20);
    B.getval (30, 40);
    C = sum (A, B);
    A.dispval (A);
    B.dispval (B);
    C.dispval (C);
}
```

test (class) object

return type

friend test sum (test, test);

arguement taking two objects

test sum (test t1, test t2) → 2 obj as argument

return t; → returning another object

function returning object in C

A.dispval (A);

B.dispval (B);

C.dispval (C);

ST-15 Assigning Object

- Simple variable assign করা যায় তেমনি একটি object কে আরেক object এ assign করা যায়। তবে condition হলো object দুটো same class এর হতে হবে।

```
class rectangle{
    int height; int width;
public:
    void set(int a, int b);
    int area();
};
inline int rectangle::area()
{ return height * width; }
void rectangle::set (int a, int b) {
    height = a; width = b; }
main()
{ rectangle obj, obj1;
  obj.set (2, 3);
  obj1.set (5,6);
cout<< "Area = "<<obj.area()<<endl;
cout<< "Area = "<<obj.area()<<endl;
obj1 = obj;
cout<< "Area = "<<obj.area()<<endl;
cout<< "Area = "<<obj1.area()<<endl;
}
```

☐ Constructor: (a special type function)

```
class test
{ public:
   test() { — }
}
```

constructor এর name class এর মতোই। — obj create করার আগে value initialize করার আগে এর কাজ।

— একে constructor বলা হয়েছে কারণ যদি class এর data member এর জন্য value construct করে।

```
class test
{ int a, b;
public:
   test()
   { a=0
     b=0;
   }
}
```

যখন obj বানাবো তখনই automatically value initialize হবে।

① Construc.

① public এ declare করতে হয়।

(ii) It doesn't return any value (void ও হয় না)

③ Inherit হয় না।

④ virtual হয় না

⑤ constructor কে তার Address হবে refer করা যায় না।

⑥ Default Argument use করা যায়।

— default constructor কোন Argument pass করে না।

Scanned by CamScanner

FT-250

Parameterized Constructor

```
class test
{  int a, b;
 public:
   test (int x, int y)
 {  a = x;  b = y }
   void disp (void)
 {  cout << "value of A" << a;
    cout << "value of B" << b;
 }
};

main()
{  test t(100, 200);
   t.disp();
}
```

// দুটো variable এর মান আগে থেকে দেওয়া নেই, কিন্তু যখন object create করা হচ্ছে, constructor এ parameter ও argument দিতে হবে।

Copy Constructor :

obj to create & initialize করার জন্য ব্যবহার করা হয়।

```
class name (const classname &obj)
{
}
```

reference variable

Example:

```
class test
{
}
```

test t
test t1(t)

or test t2 = t1

Program:

```
class test
{   int code;
    float price;
public:
    test(int c, float p);   → Parameterized constructor
    test (const test &t);   → Copy constructor
    void disp();
};

test::test(int c, float p)
{   code = c; price = p;
}

test::test(const test &t)
{   code = t.code;
    price = t.price;
}

void test::disp()
{   cout << "Code: " << code;
    cout << "Price: " << price;
}
```

```
test t2 (t1);   → calling copy constructor.
or→ test t3 = t2;  → এখানে t2 এর value t3 তে যাবে
```

```
cout<< " t1 object ";
t1 . disp();
t2 . disp();
t3 . disp();
}
```

আমরা যদি কোন copy constructor না লিখি তাহলে default copy constructor call হবে (C++ এ আছে)।

তাহলে copy constructor এর code কেন লিখবো কি?

default copy constructor এ object এর member by member copy হয়। কিন্তু copy constructor এর member copy করা যায়, কোন object আমি নিজের মত করে copy করা যায়, আবার কোন member এর কতটুকু copy করতে হবে। আবার copy করার সময় এর মানকে modify করে ~~copy~~ price = 2 * t.price; করবে।

~~code = 2*~~

default copy constructor এর কাজ *[নির্দিষ্ট একবার ভুলে না নিলেও হবে।

* copy constructor এ যখন কোনো object create করা হয় তখনই obj initialize করা হয়।

ST-24: Overloading introduction

(means) — একটি নাম দিয়ে একাধিক কাজের মধ্যে particular একটি কাজ করা

— (i) function overloading

(ii) Overloading

A + B

```
int sum(int a, int b)
{
  return a+b;
}
float sum(float a, float b)
{
  return a+b;
}
```

এই দুটো আলাদা function এর definition একটি name দিয়ে represent করতে পারি

— এখানে function দুটিকে আলাদা আলাদা ভাবে declare করা হয় কিন্তু তাদের নাম হবে same.

~~sum~~ sumint(A,B)   ~~sumf~~ sumFloat(A,B)

↓

sum(A,B)

Operator overloading:

(i) A B — int, float, double → arithmetic op... are applicable

(ii) A B — string character array → generally arithmetic operati... are not applicable

Scanned by CamScanner

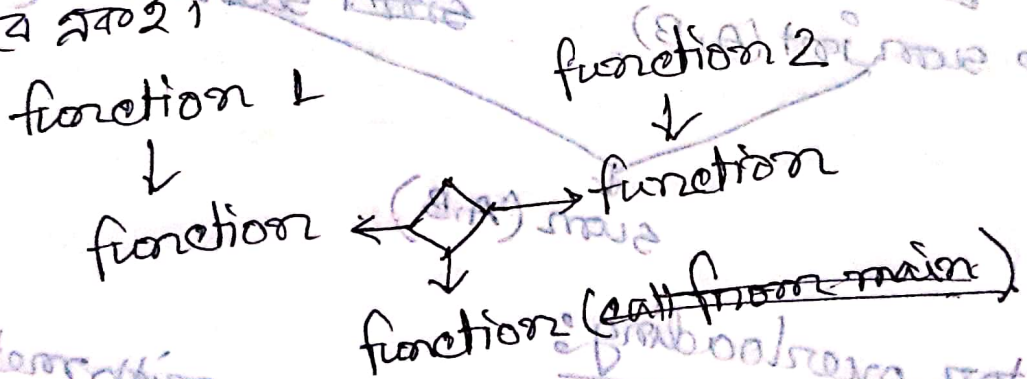
So to add two string by using (+) we have to use operator overloading.

cpp তে string আগে থেকেই + operation overload করা আছে।

কিন্তু আমরা যদি চাই দুটো string এ number দিব এবং যোগ করার জন্য + operation করব সেক্ষেত্রে আমরা অবিকল string class নিয়ে এই কাজটা করতে পারব না, কাজটা করার জন্য + use করতে পারব যদি এটা ~~ekhe~~ overload করি।

ST → 24 Function overloading introduction

একাধিক function দুটি ভিন্ন কাজ করবে কিন্তু তাদের নাম হবে একই।



যখন কোন আমরা যেকোন function কে call করব ~~আমরা~~ তখন যে ফিচার্স দেয় f1 এর মতো কাজ করবে না f2 এর মতো।

```
int add_two_int(int a, int b){
    return a+b;
}
```

add_two_int(5,6)

↓

```
int add (int a, int b)
{ return a+b
}
```

add(5,6)

```
int add_three_int(int a, int b, int
{ return a+b+c;
} add_three_int(5,6,7)
```

```
int add(int a, int b, int c)
{ return a+b+c;
}
```

add(5,6,7)

function call করার সময় parameter 2 টা হলে add(5,6) এবং 3 টা হলে add(5,6,7) এভাবে automatically call করবে।

3 → (i) name
(ii) number of arg.
(iii) type of arg

add(4,5,6,7) → 10 int এরর

```
float add(float a, float b)
{ return a+b;
}
```

→ add(4.5, 6.8)

Error

এখানে double use করতে হবে তা

main এ, float a = 4.5, b = 6.8 করে add(a,b); করতে হবে।

```
int add(int a, int b){
  return a+b;
}
float add (float a, float b) {
  return 2*(a+b);
}
double add (double a, double b){
  return 3*(a+b);
}
```

float (double a, double b) {
return (float) (2*(a+b));
}

if i call add (2.7 + 3.8) it will call 3rd function not the second.

If the the difinition is like this the compiler will show errors because there is same functions having same number of arguments and same ~~number~~ type of arguments but their return type is different.

Function overloading will not work in this case.

ET

52: Constructor overloading

Use: বহু সংখ্যক arguments এর জন্য একাধিক constructor

And the over loaded constructor.

```
class test
{ int a,b;
  public:
     test();
     test (int x);
     test (int x, int y);
     void disp ();
  };
test :: test()
{  a=0 ; b=0; }

test :: test (int x)
{  a=b=x ; }
test :: test (int x, int y)
{  a=x; b=y ;

}
void test :: disp()
{ cout << "A:" << a ;
  cout << "B:" << b;

}
```

```
OBJ A
A - 0
B - 0
OBJ B
A - 100
B - 100
OBJ C
A - 100
B = 200
```

```
main()
{
 test A ;
 test B (100);
 test C (100, 200);
cout << "OBJ A :" ;
 A.disp();
cout << "OBJ B" ;
 B.disp();
cout << "OBJ C" ;
 C.disp() ;

}
```

ET-3 Destructor :- destroy a object.

(i) name is same as class name. ↓ TILDE
(ii) no return type like constructor.
(iii) can not take arguments.

destructor is automatically called by compiler if you have made ~~de~~ destructor. It realeses the memory captured by a object created by a constructor. When I create any object in a ~~to~~ block and the block executed then the destructor is called by the compiler to ~~destor~~ destroy the object and the memory space.

programme :

```
class test
{ static int count;
public
  test()
  { ~~cout~~ count++;
    cout<< count<<" object created";

  }
  ~test()
   { cout << count<<" object destroyed ";
     count--;
  }

};
int test::count;
```

→ static member को create declare करना पड़ता है

```
int main()
{
    cout<< "main Block";
    test t1;
    {  cout<< "Block 1";
       test t2, t3;
       cout<< "Exit From Block 1";
    }
    cout <<"Exit from main";
    return 0;
}
```

output
main block
object created

DEFAULT ARGUMENTS

```
int abc (int a, int b=0)
{  return a+b;
}
int main();
{
   cout<< abc (5)<<endl;
   return 0;
}
```

(i) এখানে যদি abc(5,6) দিয়ে call করি তাহলে 11 return করবে।
(ii) যদি পারবু Argument হিসেবে না দিই তাহলে b এর value default হিসেবে 0 ধরে নিবে।

* default value এর সময় ডানের argument এর দিক থেকে আসবে। First এর দিকে দিলে ERROR

* default value always constant অথবা global variable হতে হবে। main থেকে এর মান পাঠাতে পারবে না

⊞ ST-29 : Over loading and ambiguity

ambiguty → ~~&~~ when I couldn't make my choices because all the optitions fulfil my demands.

Ex

```
int abc (int a, int b)
{ return a+b;
}
float abc (int a, int b)
{ return a/b;
}
```

অবশ্যই হচ্ছে এটা ambiguity
এর কারণ হলো return type doesn't matter.

overloading এ ৩টা বিষয়
(i) name
(ii) number of arg.
(iii) type of argument

→ এখানে ৩টা বিষয়ই same তাই এখানে ambiguity.

Again —

```
int abc (int a, int b=0)
{ return a+b;
}
int abc (int a)
{ return a;
}
abc (5,6) → correct
abc (5) → Error
```

int b=0 → default argument

কারণ first function এ default argument দেওয়া আছে

Previous classes ET-08 [IOSTREAM]

istream → Input ↦ cin → console input

ostream → Output ↦ cout → console output

cin>>a; → Extraction/getfrom operator

cout << "C++" → insertion

```
for (a=1 ; 0 ; a++)
{ cout <<a;
}
```

ST → 22 const

constant এবং define এর মাঝে main এ পার্থক্য হলো define এর কোন নির্দিষ্ট Data type থাকে না।

* class এ কোন function const এ declare করা থাকলে ঐ ক্লাসের কোন member এর value change করতে পারবে না।

```
class exampleConst {
  public :
    int a ;
    int change (int x) const
  {  a = x;  return a ; }
};
```

→ Error দেখাবে

x = a+x হলেও Error দেখাবে না

```
int main()
{ int i=0;
  const int &n = i;
  n=5; ⟶ Error
  return 0;
}
```

ST - 14 NEW and DELETE (used for dynamic memory allocation)

c তে malloc
calloc realloc
parenthesis ব্যবহার
[] ব্যবহার array

```
int main()
{ int *ptr;
  ptr = new int(5); ⟶ ডাইনামিক variable তৈরি করে address
  cout << *ptr << endl;
  *ptr = 7; ⟶ আগের 5 বদলে গেল.
  cout << *ptr << endl;
  delete (ptr); //deallocating ⟶ c তে "free" use
  return 0;
}
```

```
int * ptr;
ptr = new int [5];
ptr [0] = 6;
cout << ptr[0] << endl; ⟶ 6
cout << *ptr << endl; ⟶ 6
delete [] ptr;
```

↳ deleting the array

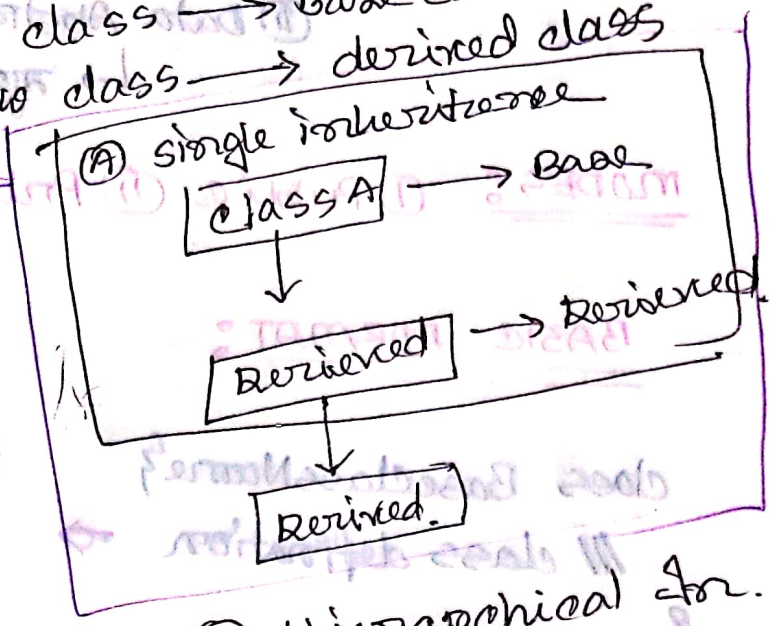
E-55 + S-44

Introduction to inheritance.

- new class create करने के old class के properties use करना
- OOP feature

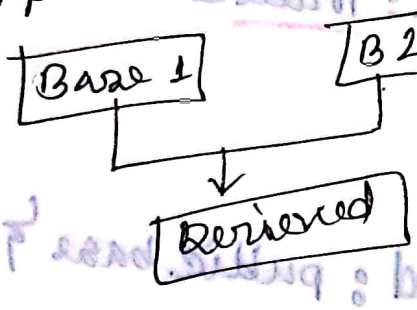
old class → Base class
new class → derived class

(A) single inheritance

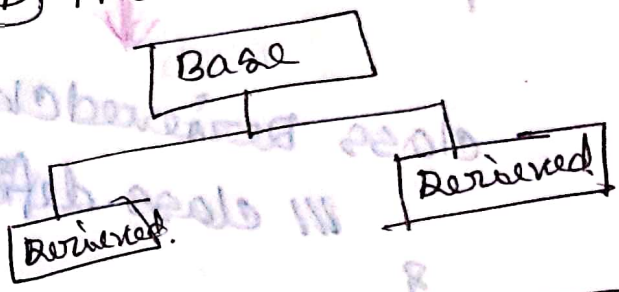


(B) multilevel inheritance

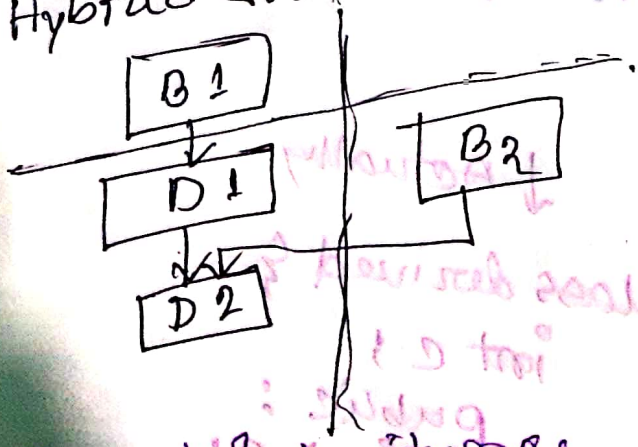
(C) multiple inheritance



(D) Hierarchical Inh.

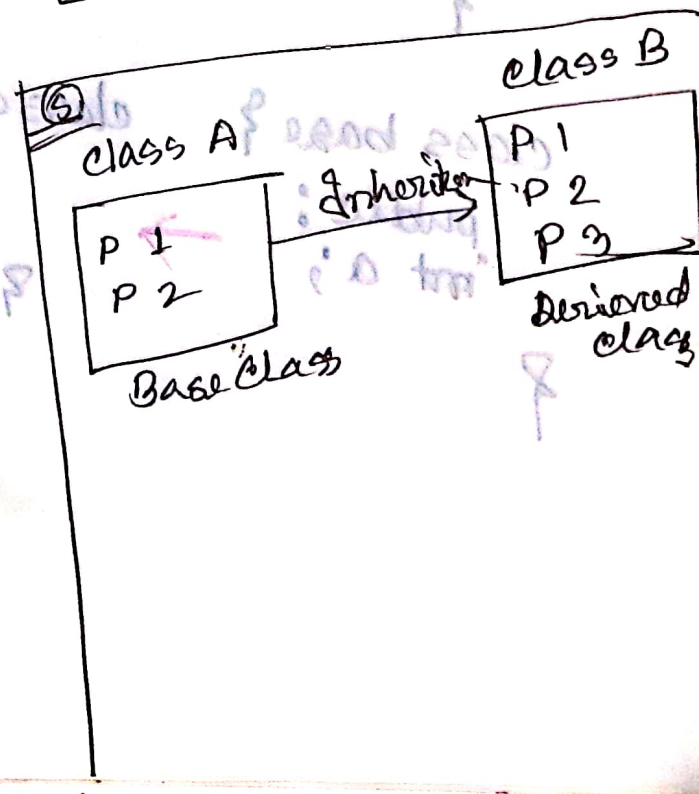


(E) Hybrid Inheritance



(F) Virtual Inheritance.

(G)



Advantages: ① Data Reusability —
② Data Abstraction. — ~~addition~~
↳ নতুন ক্লাস তৈরি করার

MODES: ① Public ② Private ③ Protected ~~Letter~~.

BASIC FORMAT:

class BaseclassName{
/// class defination
}

~~class derived~~

class DerivedClassName : mode BaseClass Name {
/// class defination
}

→ by default private.

class base {
public:
int a;
}

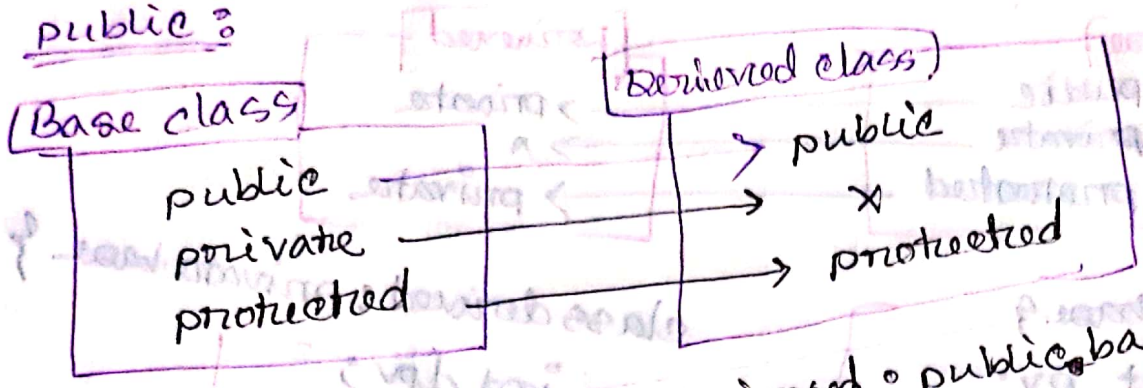
class derived : public base {
int c;
}

↓ Actually

class derived {
int c;
public:
int a;
}

Modes:

public:

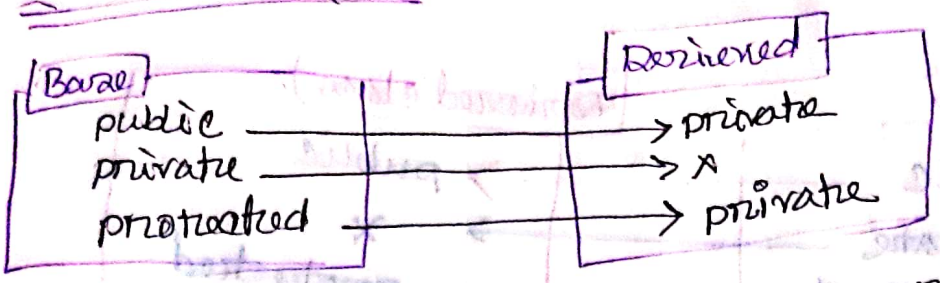



```
class base {
    int pv;
protected:
    int pt;
public:
    int pb;
    void print();
};

class derived : public base {
    int dpv;
protected:
    int dpt;
public:
    int dpb;
    void dprint();
};
```

→ actually class derived { → not private data has come from base:

```
    intpv;
protected
    int pt, dpt;
public:
    int pb, dpb;
    void print();
    void dprint();
};
```

Private mode:




```
class base {
    int pv;
protected
    int pt;
public :
    int pb;
    void print();
};

class derived : private base {
    int dpv;
protected:
    int dpt;
public :
    int dpb;
    void dprint();
};
```

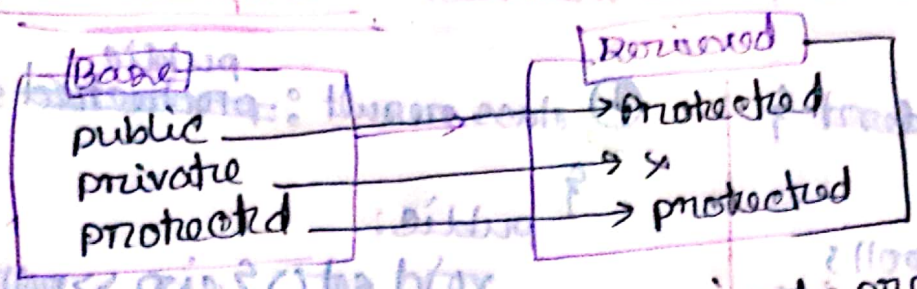
new.

→ actually

```
class derived {
    int dpv, pb, pt;
    void print();
protected :
    int dpt;
public :
    int dpb;
    void dprint();
};
```

new.

protected mode :




```
class base {
    int pv;
protected :
    int pt;
public :
    int pb;
    void print();
};
```

```
class derived : protected base {
    int dpv;
protected :
    int dpt;
public :
    int dpb;
    void dprint();
};
```

```
class derived {
    int dpv;
protected :
    int dpt, pb, pt;
    void print();
public :
    int dpb;
    void dprint();
};
```

S-046 Inherite public member

protected
private
public

Base

```
class student {
public:
    int roll;
    int mark;
};
```

① class result : ~~protected~~ public student

```
{
public:
    void set() { cin >> roll >> mark; }
    void print() { cout << roll << " " << mark; }
```

② ~~class result : private student~~

```
};
int main()
{
    student ob;
    Result ob;
    ob.roll = 5;
    ob.mark = 75;
    ob.set();
    ob.print();
    cout << ob.mark << " " << ob.roll << endl;
    return 0;
}
```

→ private and protected হলে এরর Access করা যায় না

TANVER LIKHON

S 047 Inherite private member.

Actually Base class এর কোন private member কে public এ inherite করলে কোন inherite হয়না। আমরা Derived class থেকে Base class এর private member. কে access করতে পারিনা। এ জন্য বাইরের কোন access করতে হয়।

```
class student {
  private:
    int roll;
    int mark;
  public:
    void set() { cin>>roll>>mark; }
    void print() {cout<<roll<<mark;}
};
class result: public student {
  public:
    void all(){set(); print();}
};
```

→ Bypassing.

এখানে যদি void p(){ cout<<roll<<mark; } Error দেখাবে, কারণ Access করা যাবে না।

public করলে main থেকে all() দিয়ে call করতে হবে
তবেই set() print() দিয়ে call করা যাবে।

কিন্তু private or protected করলে only all() দিয়ে call করতে হবে।

S 048 Inherite protected member

~~class~~ protected ক public ২ inherite করলে আগে protected ই থাকবে private ২ করলে আগে private হবে, তাই এগুলোকে access করতে হলে public function use করতে হবে।

E-56 Single inheritance (ST-49) → NOTHING XTRA

Class A
↓
derived class

```
class A
{ int a;
  public:
    void getval_a(int);
    int get_a();
};

void A::getval_a(int x)
{ a = x; }

int A::get_a()
{
  return a;
}
```

```
class B : public A
{ int b,c;
  public:
    void getval_b(int);
    void add();
    void dispval();
};

void B::getval_b(int x)
{ b = x; }

void B::add()
{ c = get();
  c = c + b;
}
```

nesting of member function

```
void B::dispval();
{ cout << "value of A" << get_a();
  cout << "B" << b;
  cout << "c" << c;
}
```

```
int main()
{
  int a,b;
  B obj;
  cin >> a;
  cin >> b;
  obj.getval_a(a);
  obj.getval_b(b);
  obj.add();
  obj.dispval();
  return 0;
}
```

E- 57 - Multilevel Inheritance

Base ⟶ Derived ⟶ New derived

```
class stu {
protected:
    int roll;:
public:
    void get_roll(int);
    void put_roll();
};
```

```
void stu :: get_roll (int x)
{
    roll = x;
}
void stu :: put_roll()
{
    cout << "Roll:" << roll;
}
```

```
class test : public stu
{
protected:
    float m1, m2;
public:
    void get_marks(float, float);
    void put_marks();
}
```

```
void test :: get_marks(float x, float y)
{ m1 = x; m2 = y;
}
void put_marks()
{
    cout << m1;
    cout << m2;
```

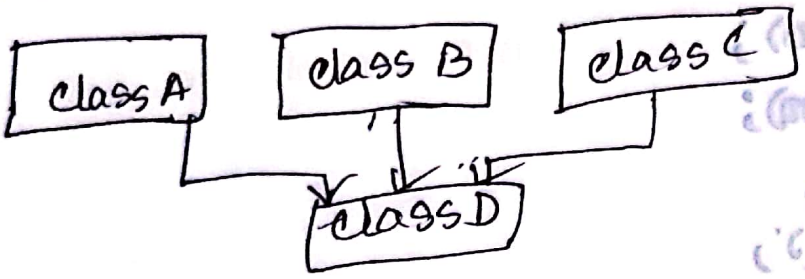
```
class result : public test
{
    float total;
public:
    void display();
};
```

```
void result :: display()
{ total = m1 + m2;
    put_roll();
    put_marks();
    cout << "\nTotal =";
         << total;
```

m1, m2 result
protected mode inherite

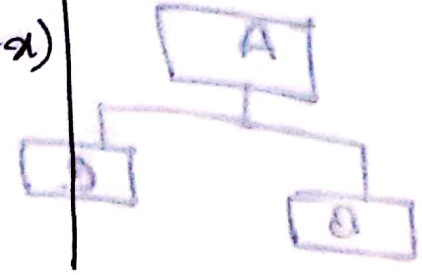
```
int main()
{ result obj;
  obj.getdata_roll(101);
  obj.get_marks(53.3,70.1);
  obj.display();
  return 0;
}
```

Multiple Inheritance: → 1 derived class from more than one base class.




```
class A
{ protected:
    int m;
  public:
    void getm(int)
};
void A::getm(int x)
{ m = x;
}
```

```
class B
{ protected:
    int n;
  public:
    void get_n(int);
};
```



P.T.O.

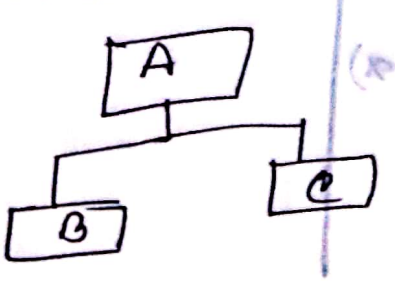
```
class C : public A, public B
{
    int c;
    public:
     void add();
     void display();
};
void c:: add()
{ c = m+n;
}
void c:: display()
{ cout << m << n;
  cout << c;
}
int main()
{
    C obj;
    obj.get_n(100);
    obj.get_m(200);
    obj.add();
    obj.display();
    return 0;
}
```

visibility mode

E-50 Hierarchical Inheritance

when problem is categorised

1 Base class → 2+ derived class



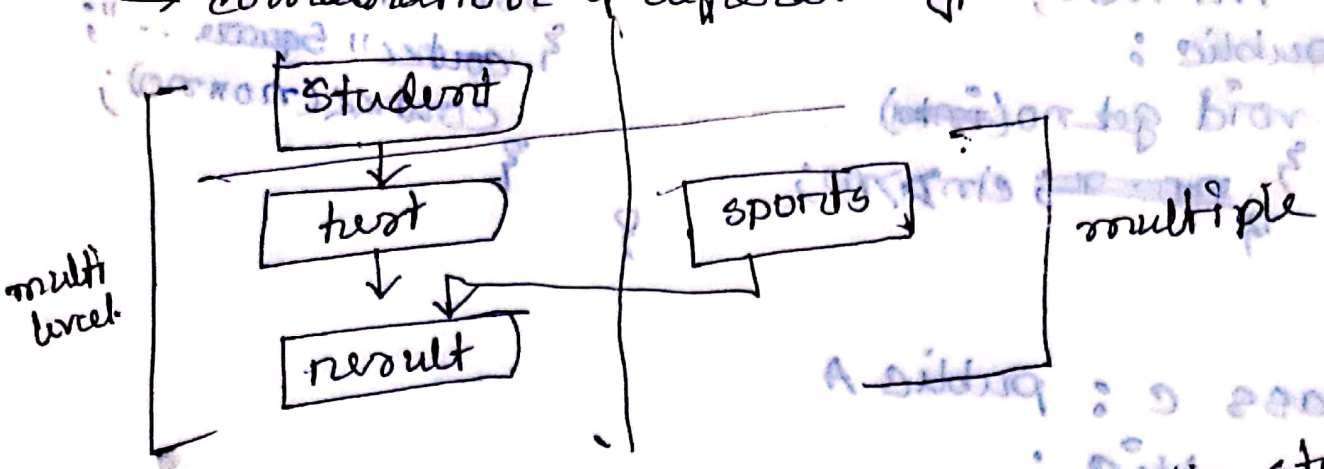
```
class A
{ protected:
    int no;
  public:
    void get_no()
    { cin>>no;
    }
};

class B : public A
{ public:
    void square()
    { cout<<" Square .. ";
      cout<<(no*no);
    }
};

class C : public A
{ public:
    void cube()
    { cout<<" Cube .. " << (no*no*no);
    }
};

int main()
{
    B bobj;
    C cobj;
    bobj.get_no();
    bobj.square();
    cobj.getno();
    cobj.cube();
    return 0;
}
```

→ এখনকার ফাংশনে square or cube এ nesting করলে এখানে main এ call করতে পারতো না।

→ আবার value গুলো যাচ্ছে class A তে

E-60 Hybrid Inheritance:

→ combination of different types of Inheritance.




```
class student
{   int roll;
  public:
    void getroll(int x)
    {  roll = x;
    }
    void putroll()
    {  cout << roll;
    }
};
```

```
class test : public student
{  protected:
     float t1, t2;
   public:
     void getmarks(float x, float y)
     {  t1 = x;
        t2 = y;
     }
     void putmarks()
     {  cout << "--" << t1 << t
     }
};
```

```
class sports
{ protected:
    float sp;
  public:
    void get_sp(float x)
    { sp=x;
    }
    void put_sp()
    { cout<<sp;
    }
};
```

```
class result: public test, public sports
{ float total;
  public:
    void disp();
};
```

Tanver Likhon

```
class result : public test, public sports
{ float total;
  public:
    void disp();
};
void result::disp()
{ total = t1+t2+sp;
  put_roll();
  put_mark();
  put_sp();
  cout<<"Total score "<<Total;
}
```

nesting of function

```
int main()
{ result obj;
  obj.get_roll(101);
  obj.get_mark(80.5, 82.3);
  obj.get_sp(9.5);
  obj.disp();
  return 0;
}
```

```
class sports
{ protected:
    float sp;
  public:
    void get_sp(float y)
    { sp=x;
    }
    void put_sp()
    { cout<<sp;
    }
};
```

class result: public test, public sports.

```
{ float total;
  public:
    void disp();
};
void result::disp()
{ total = t1+t2+sp;
  put_roll();
  put_marks();
  put_sp();
  cout<<"Total score"<<Total;
}
```

nesting of function

```
int main()
{ result obj;
  obj.get_roll(101);
  obj.get_marks(80.5, 82.3);
  obj.get_sp(9.5);
  obj.disp();
  return 0;
}
```

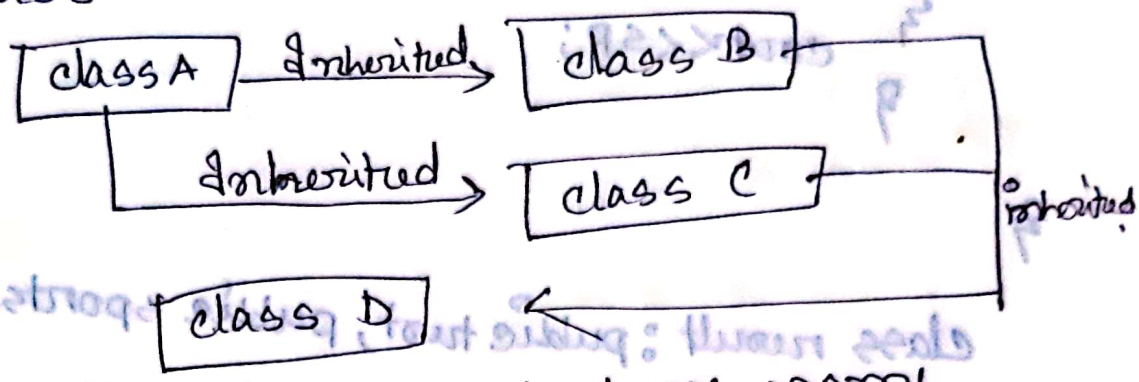
class result: public test, public sports

{ float total;
  public:
    void disp();

S-54: Virtual Inheritance

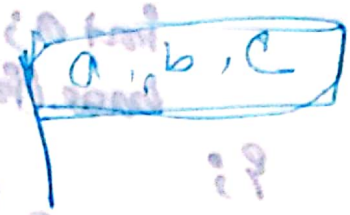
Virtual Inheritance is a c++ technique that ensures only one copy of a base class's member variables are inherited by by grandchild derived class.

Suppose



So there is a possibility to have same ~~name~~ data member on class D which ~~and~~ brings ambiguous. To remove this ambiguous situation, we have to use virtual keyword to class B and class C.

```
class base {
  public :
    int a;
};
class new1 : public base
{ public:
    int b;
};
class new2 : public base
{ public :
    int c;
};
class drive : public new1, public new2
{
    void get() { cin >> a >> b >> c; }
    int add() { return a+b+c; }
};
int main()
{ drive ob;
  ob.get();
  cout << ob.add();
  return 0.
}
```



S-55 The Inheritance and constructor

* Base class ঐ constructor থাকলে

Inherit করলে Base class ঐ constructor আগে call হবে derive class ঐ পরে। And destructor will call latter in base class and first in derive class.

If we take another base class having constructor then the constructor should be placed here

```
class base {
public:
    int a;
    base(int x) { a=x; }
};
class derive: public base {
public:
    int b;
    derive(int y) : base(5)
    {
        b=y;
    }
};
int main()
{ derive ob(6);
  cout << ob.a << endl; → 5
  cout << ob.b << endl; → 6
  return 0; }
```

first calling base class's constructor then derive class's constructor.

base(5)

Details on Books ☺

5 - 56 Inheritance and Same member

- If Base class and derived class is member are the name
- If I initialize the common variable of the base and derive class it will automatically be initalized to derive class:

```
class base{
 public:
    int a;
    int get() {return a;}
};
class derive : public base{
    public:
      int a;
      int getd() {return a+get();}
};
int main()
{ derive obj;
    obj.base::a = 1;
    obj.derive::a = 5;
    cout<< ob.getd() <<endl;
    return 0;
}
```

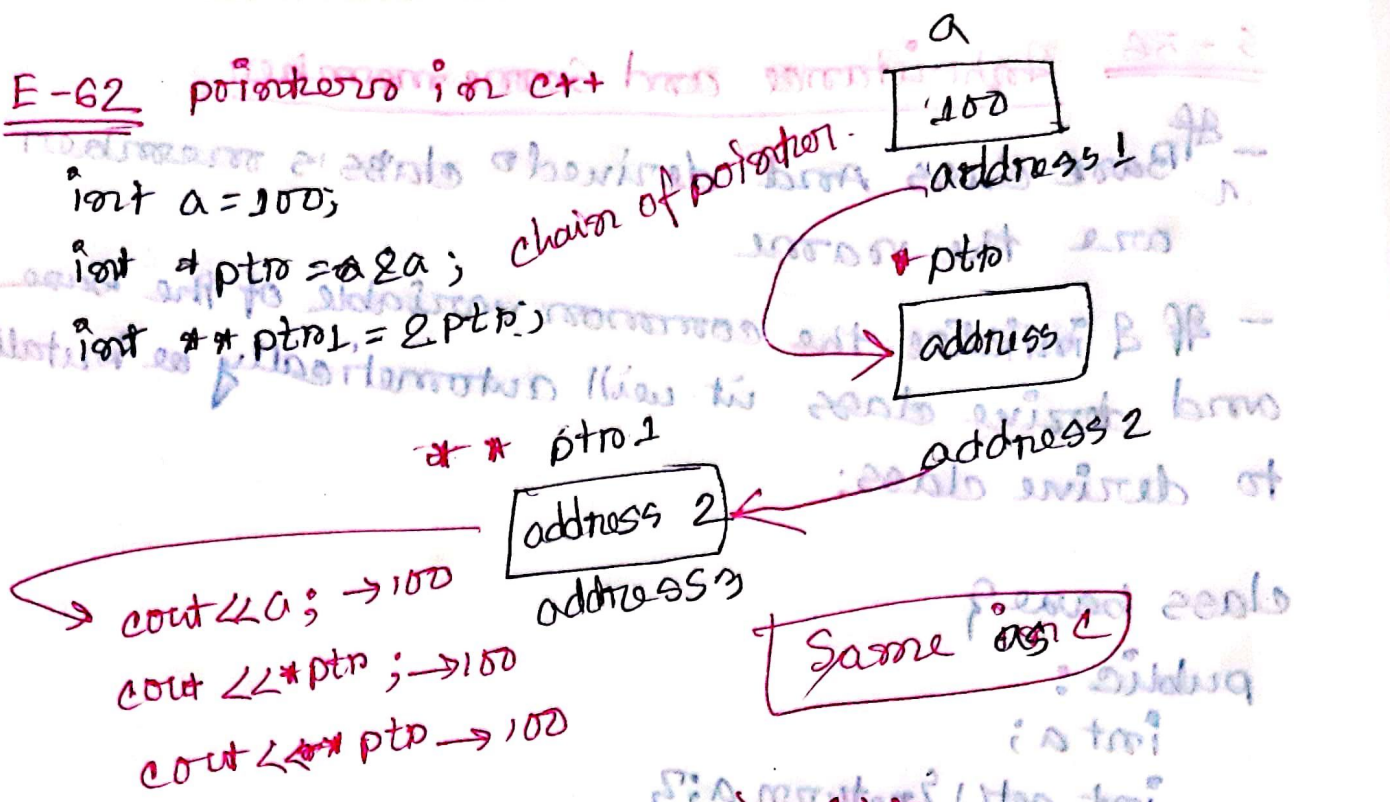
→ initializing the member of base

→ optional.

→ initializing to derive class.

E-62 pointers in C++

```
int a = 100;
int *ptr = &a;
int **ptr1 = &ptr;
```



E-63 Pointers and Arrays in C++

Same as C

E-64 Pointers to objects

```
student → class
student obj;
student *obj1;
obj1 = &obj;
```

generally we call

obj.disp()

through pointer obj1 → disp(); — (1)

or (*obj1).disp(); — (2)

→ parenthesis are used because of (*)'s precedence

(1) & (2) ways

```
class student
{ int roll;
public: char name[10];
  void getdata(int x, char *y)
  { roll = x;
    strcpy(name, y);
  }
  void putdata()
  { cout<<" student name" <<name;
    cout<<" Student Roll " << roll;
  }
};
int main()
{ student obj;
  student *p;
  p = &obj;
  p -> getdata(101, "biggboss");
  p -> putdata();
  return 0;
}
```

equivalent to char y[] → big Legal

(*p).getdata(101, "biggboss");

(*p).putdata();

~~reader~~ not accepted by com

S-57 Inheritance and Pointer

```
class base {
  public: int a;
};
class derive : public base {
  public: int b;
};
int main()
{
  base b;
  derive d;
  d.a = 5;
  cout << d.a << endl;
  d.b = 6;
  cout << d.b << endl;
  base *ptr;
  ptr = &d;
  ptr -> a = 1;
  cout << ptr -> a << endl;
  ptr -> b = 2;
  return 0;
}
```

→ can be accessed because derived class has the member of base class.

→ taking pointer of base class.

→ error. not accessable because we have take pointer of the base class not the derived class and (b) is the member of derived class.

S-58 ~~Inheritance~~ Function overid. overriding

- one kind of ambiguity in inheritance
- when we use same function in derive and base class this happens.

```
class base {
  public: void print() {cout << " Base " << endl; }
};
class derive : public base {
    void print() { cout << " Derive " << endl; }
};
int main()
{
    derive ob;
    ob.print()
    return 0;
}
```

this function will call the function from derive class. the call will first search in the derive class than the base. So if I comment this line it will show print function of the base class. This is called overriding.

the function in this case can't be overloaded, we put

For example: If in base class and call ob.print(2,3) it will show error because the call will search in the derive class if there was no 'print()' fun. in derive class then it will would execute print(2,3) of base class.

void print(int, int)

S-59 Inheritance and pointer part – 2

From last tutorial → object firing function call করতাম আর derive class এর function টিকে call করতাম। কিন্তু এখন pointer দিয়ে call করতে হবে –

```
Base *b;
Derive *d;
b = &d;
b->print()
```

→ তখন Base এর class এর function কে call করবে।

S-32 Operator

→ a sign to do a finite task. ex. +, −

(a + b) → operand.

→ ① Unary → deals with one operand → a++;

② Binary → a+b;

③ trinary → conditional → a>b ? c : d ;

S-33 Operator overloading # Intro

```
int a = 5, b = 6, c;
c = a+b ⟶ 11
string sa = "ABC", sb = "DEF";
sc = sa + sb ⟶ ABCDEF
```

(+) is working for both integer and string.

We can only overload unary and binary operator,

The operators which can't be overloaded.

(1) . (dot) (2) .* (pointer) (3) : ? (ter.) (4) ::

(5) sizeof (6) # preprocessor

We can't create new operators we have to overload the existing operators.

And we can't overload all the operators we have.

we can't change the precedency of operator

unary operator will not work like binary and binary won't like unary

we change operator (but we should not)

operator can gain new characteristics but will not loose it's own characteristics

Operator overloading

→ Give additional meaning to operator.

\+ Info. from previous page.

Refining operator overloading.

```
Returntype classname :: operator op (Arg-list)
{
    body;
}
```

symbol ↑ (op) → which will be overloaded

* If we declare outside of the class → will be not have to write to declare inside the class

* operator function class member ও হতে পারে।
আবার friend function ও হতে পারে।

test operator – ( ) → Unary
test operator + (test) → Binary

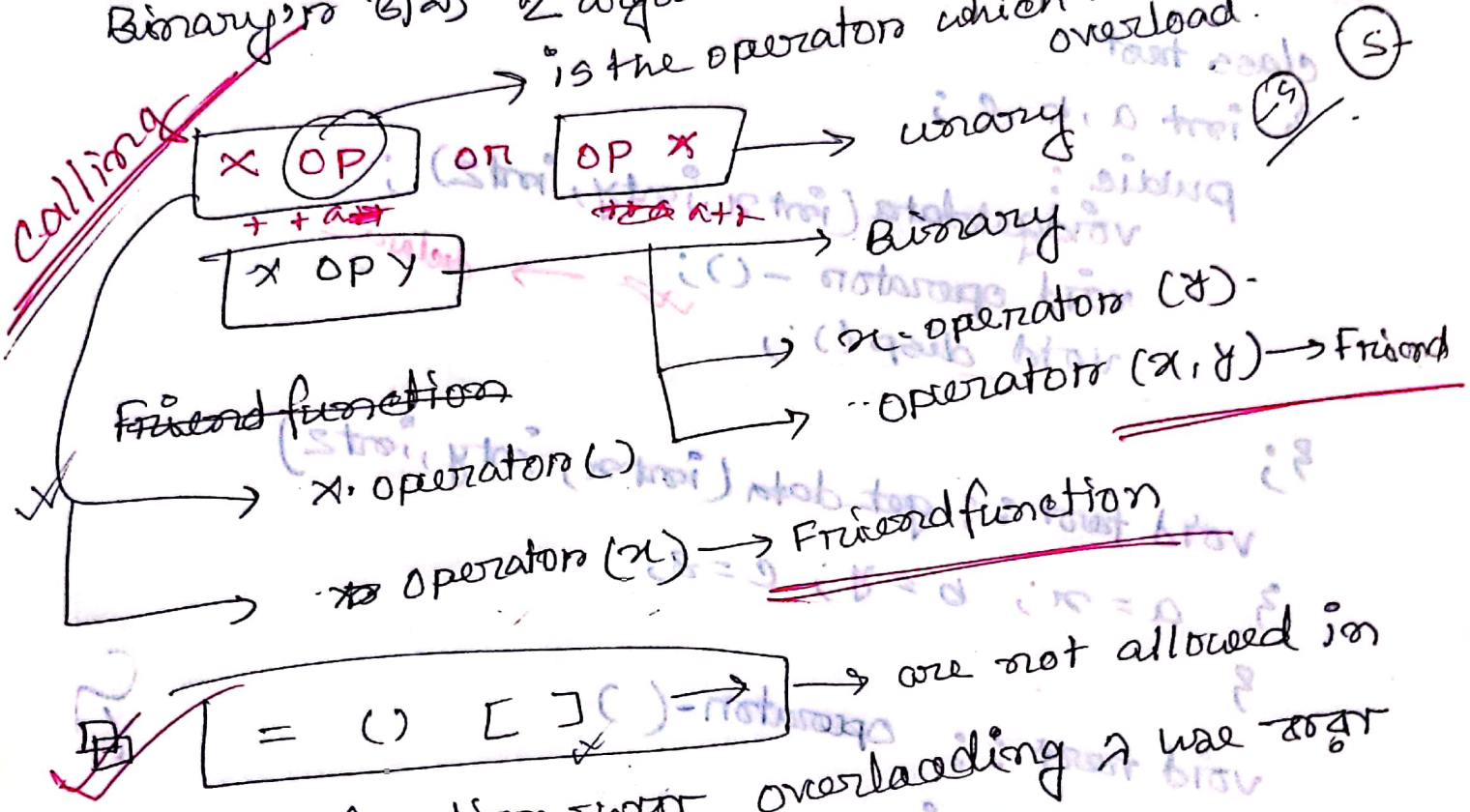
obj দ্বারা যখন call করা হবে Binary op overload
কে তখন object implicitly pass হয়ে যাবে।

Binary এর ক্ষেত্রে যে দুটি object যোগ করা হবে
তার একটি implicit ভাবে pass হবে আর
অন্যটি argument হিসেবে।

For friend function.

friend test operator = (test) → Unary
friend test operator + (test, test) → Binary

যেহেতু Friend function কোনো class: এর member নয় তাই unary'র ক্ষেত্রে 1 argument and Binary'র ক্ষেত্রে 2 argument.



= () [ ] ( ) → are not allowed in friend function দ্বারা overloading ও করা যায় না।

TANVEER AHMED

S-34 Operator overloading start

→ overload করার জন্য যে operator (unary বা binary) use করা হয়।

→ এক্ষেত্রে default কোন argument থাকবে না।

E-66 Unary operator overloading

```
class test
{ int a, b, c;
  public:
    void getdata(int x, int y, int z);
    void operator -();
    void disp();
};
void test :: get_data(int x, int y, int z)
{ a = x; b = y; c = z;
}
void test :: operator-()
{ a = -a;
  b = -b;
  c = -c;
};
void test :: disp();
{ cout << a << b << c;
}
```

```
int main()
{ test obj;
  obj.getdata(-10,20,30);
  obj.disp();  → -10, 20, 30
  -obj. → calling operator overloading
  obj.disp();  → 10, -20, 30
return 0;
}
```

operator (-)

t.a

E-67 : Binary operator overloading

```
class test
{ int a, int b;
  public :
    test() { a=0; b=0; }
    test(int x, int y) { a=x; b=y; }
    test operator + (test t);
    void disp();
};
test test :: operator + (test t)
{ test temp;
  temp.a = a + t.a;
  temp.b = b + t.b;
  return temp;
}
```

creating a object

new object এর a and b variable.

test class এর variable কে object দিয়ে বোঝানো

pass করা এর a ও b

```
void test :: disp()
{ cout << a << b;
};
```

```
int main()
{
    test t1, t2, t3;          → default constructor.
    t1 = test(10,20);         ┐→ Explicitly calling
    t2 = test(30, 40);        ┘  constructor
    t3 = t1 + t2;             → + overloading.
    t1.disp();                → 10, 20
    t2.disp();                → 30, 40
    t3.disp();                → 40, 60.
    return 0;
}
```

(−) হলে error দেখাবে কারণ আমরা minus overload করিনাই

to prev page এর t.a and t.b এর a and b t2 object এর.

8-35 Binary operator overloading

Binary operator overloading এর definition

Binary operator overloading এ class এর একটা object to যে class এ নিয়ে যাবে class এর একটি object implicitly pass হবে। এবং অন্যটি Argument এ pass করাতে হবে।

Number যদি একটি class হয়

Number operator + (class name obj);

কোনো obj. হতে পারে কিন্তু যে class আছে এর name হবে

$N_3 = N_1 + N_2$ → একটা বা ৩ class এর 3 হতে পারে।

N_1 → implicitly pass হয়

CODE SAME AS E-67

S-36 Relational operator overloading

— এ ক্ষেত্রে overloading এ আরেকটু সত্য বা মিথ্যা এ ক্ষেত্রে Answer চাই।

→ 2 ta object equal কিনা চেক করব।

```
class Number {
    int x, y;
public:
    Number () { x=0; y=0 }
    Number (int a, int b) { x=a; y=b; }
    void get (int &a, int &b) { a=x; b=y; }
    void set (int a, int b) { x=a; y=b; }
    void print () { cout<<x<<" "<<y<<endl; }

    Number operator + (Number ob);
    bool operator > (Number ob);
    bool operator < (Number ob);
    bool operator == (Number ob);
};

Number Number :: operator + (Number ob)
{
    Number temp;
    temp.x = x + ob.x;
    temp.y = y + ob.y;
    return temp;
}
```

```
bool Number :: operator > (Number ob) {
    return (x+y) > (ob.x + ob.y);
}
bool Number :: operator < (Number ob) {
    return (x+y) < (ob.x + ob.y);
}
bool Number :: operator == (Number ob) {
    return (x+y) == (ob.x + ob.y);
}
int main()
{
    int x1, y1, x2 y2
    cin >> x1 >> y1 >> x2 >> y2;
    Number n1( ), n2( ), n3;
    n3 = n1 + n2;
    n3.print();
    if (n1 > n2) cout << "N1 largest" << endl;
    else if (n1 == n2) cout << "N1 = N2" << endl;
    else cout << "N2 largest" << endl;
    return 0;
}
```

S-37 → Logical operator overloading

```
class --- {
    int x;
public:
    bool operator && (Number ob);
};
bool Number :: operator && (Number ob)
{
    return (x && y) && (ob.x && ob.y);
}
```

code একই থাকবে

S-38 unary overloading

```
class Number {
public:
    Number operator ++ ();
};
int main()
{
    Number
```

→ code remains the same

→ void 3 টাকার হবে।

```
Number Number :: operator ++ ()
{
    Number temp;
    temp.x++; y++;
    temp.x = x;
    temp.y = y;
    return temp;
}
```

void type আমি only n1 ~~n1~~ object এর value increase করতে পারব। কিন্তু এটা Assign করতে পারব না অর্থাৎ

n3 = ++n1; → return type number হবে (এক্ষেত্রে use করতে এক্ষেত্রে আবার

n3 = n1++; error dekhabe.

কারণ এ ক্ষেত্রে ++ pre increment হিসেবে আর default post increment হিসেবে use করার জন্য parameter নিতে হবে.

Number operator ++ (int a);

Number Number :: operator ++ (int a)

{

Number temp;

same code

no use of it

}

n3 = n1++;

কিন্তু এখানে n3 এর মান n1 এর previous value আসার কথা ছিল কিন্তু এখানে n1 এর increment value আছে.

So, এটা ছিল কথা যে definition হবে

```
Number Number :: operator ++ (int a)
{ Number tmp;
  tmp.x = x;
  tmp.y = y;
  x++; y++;
  return tmp;
}
```

→ এটা পরে হবে কথা এটা post increment

```
Number Number :: operator ++ ( );
{ Number temp;
  x++; y++;
  temp.x = x; temp.y = y;
  return temp;
}
```

এক্ষেত্রে function overloading হয় যদি x++ or ++x কাজ করে।

+ unary and binary both হিসাবে use করা যায়

a = -a → unary

a = b + c → binary.

S-39 Assignment operator overloading

→ যদি এটি এর data member গুলো copy না করতে চাই — Nothing else

S40 Subscript operator overloading [ ] → binary

```
class Number {
 int x, arr[10];
 public:
  number() { x = 1; for(int i=1; i<=10; i++)
                    arr[i-1] = i*x;
  }
 number(int a)
 { x = a;
    for (int i = 1; i<=10; i++)
        arr[i-1] = i*x;
 }
 int operator [] int (int a);
};
int Number :: operator [] (int a)
{ return arr[a-1];
}
main()
{ Number N1(4), N2(8), N3;
  cout << N3[0] << endl;
}
```

S42 output stream operator overloading.

cout << n1 → obj → will show data members.

```
class number {
  int x, y;
public:
  number() {x=0; y=0;}
  number(int a, int b) {x=a; y=b;}
  friend void operator <<(ostream &os, number obj);
};
void operator <<(ostream &os, number obj)
{ os<<obj.x <<" "<<obj.y <<endl;
}
int main()
{
  number n1(3,4), n2(4,0), n3;
  cout << n2;
  /// cout << n2 <<endl;
  return 0;
}
```

friend function → access ostream variable class & number variable class

must

error because obj << endl;

S-43 Type conversion

(1) build-in to class.
(2) class to built-in.
(3) class to class

→ construction use করেই করতে পারব

```
int a=5
n1 = a  ──→ Number(int a)
n1.print()   { x=a;
             }
```

→ object

→ operator int() { return xxy; }

```
int a;
a = n1;
cout << a << cout;
```

→ class having two data members

NEVER GIVE UP

S-63 Template intro

int add (int a, int b) { return a+b; }

~~double add~~

double add (doble a, double b) { return a+b; }

string (string a, string b) { return a+b; }

~~এক~~ 3 function overload করে add operation করা হয়েছে। Teemplat হচ্ছে একবারেই. যেকোনো ক্ষেত্রে অর্থাৎ যার একটি আর্গুমেন্ট রূপ ব্যবহারে এবং parameter অনুযায়ী value return করবে.

T add (T a , T b) { return a+b; }

object এর ক্ষেত্রেও Applicable.

S-64 Template function code

```
#include <iostream>
using namespace std;
template <class T>
T add (T a, T b) { return a+b; }
int main()
{ int num1 = 5, num2 = 6;
cout << add (num1, num2) << endl;
double num3 = 1.2, num4 = 2.3;
cout << add (num3, num4) << endl;
string s1 = "Tanmoy", s2 = "Likhon";
cout << add (s1, s2) << endl;
return 0;
}
```

→ add (int, float) → Error

→ add ("ABC", "DEF") → Error

→ In this case

S-65 → Type mix Template

```
template <class T>
void print(T a) { cout << a << cout ; }
```

-এখানে main() থেকে print(—) যে argument দিবো তাই print করবে।

S-66 Template – more generic type.

```
template <class T1, class T2>
void print (T1 a, T2 b)
{ cout << a << " " << b << endl;
}
int main()
{ print (1,2);
  print (1, 2.5);
  print (1, "Biggboss");
  return 0;
}
```

একাধিক argument -টি different data type হলেও এভাবে নিতে। S-65

S-67 overloaded template.

এখানে, template <class T1>
void print(T a) { cout << a << endl; }
for one argument.

S-68 : Template function override

- যদি কোন Template কোন নির্দিষ্ট datatype এর কাজ করতে না পারে, তাহলে সেক্ষেত্রে সেটার জন্য আলাদা Template function override.

```
template <class T>
void print (T a) { cout << a << endl; }
void print (double a) { cout << "Override" << endl;
```

এখানে double input দিলে এর input এবং output আলাদা।

S-69 → Template Example swapping.

```
template <class T>
void interchange (T &a, T &b)
{   T temp;
    temp = a;
    a = b;
    b = temp;
}
```

```
int main()
{ int a = 5, b = 6;
  cout << a << b;
  interchange (a, b);
  cout << a << " " << b;
  return 0;
}
```

S-70 Template class

```
template <class T>
class Abc {
  T a;
public: Abc() {}
  Abc(T x) {a = x;}
  void set(T x);  → used to change value.
  T get() {return a;}
  void print() {cout<<a<<endl;}
};
```

```
template <class T>
void Abc<T> :: set (T x) {a = x;};
```

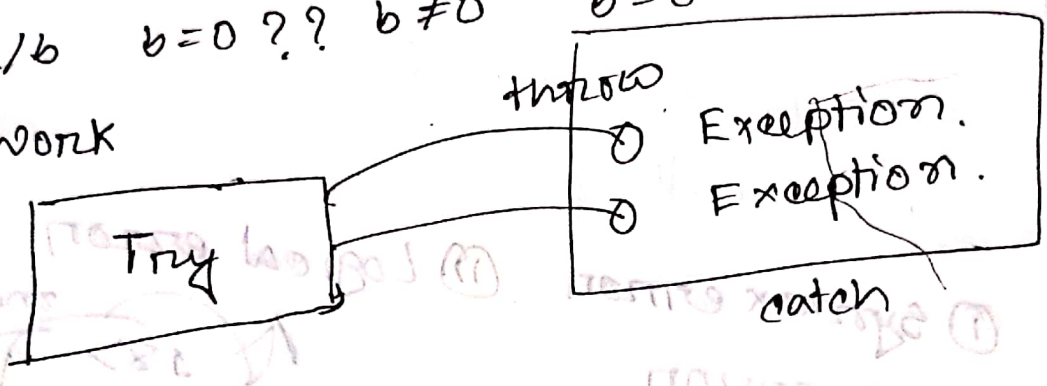
```
int main()
{ Abc<int> ob(5);
  ob.print();  → 5
  Abc<double> ob1(5.5);
  ob1.print();  → 5.5
  Abc<int> ob2(5.5) →
  ob2.print  → 5
  ob1.set(6.5);  ob1.print();  → 6.5
  return 0;
}
```

```
int main()
{  abc<int, 7> ob;
   ob.print();
   abc<char, 7>ob;
   ob.print();
   return 0;
}
```

Exception handling : S-74

Exception

a/b b=0 ?? b≠0 b=0 → exception.



- Try
- catch:
- Through

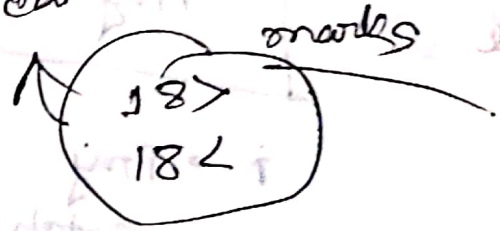
```
try
{ /// protected code
} catch
{ // et catch block
```

```
try
{ /// protected code
} catch ( ExceptionName e1 )
{ /// catch block
} catch ( ExceptionName e2 )
{ // catch block
} catch ( ExceptionName eN )
{ // catch block
}
```

~~S-75 Exception handling code~~

E-83 Exception handling

Error : (i) Syntax error (ii) Logical error



(iii) Runtime error

↳ Due to condition.

↳ ফলে input এর উপর program execution বন্ধ হয়ে যায়।

↳ Solve → Exception handling

Exceptions : (i) Synchronous → Array index out of bound, overflow

(ii) Asynchronous. → program control এর বাইরে কাজ like ctrl+c from keyboard.

C++ Exception handling

mechanism:

Ⓐ Hit the exception /Find the problem.
Ⓑ Throw the exception.
Ⓒ catch the exception.
Ⓓ Handle the exception.

try block
Detects and throw the Exception

Exception obj

catch blocks
Catches and handle the Exception.

try
{ throw exception;
}

catch (type arg) → match হলে program stop হয়ে যাবে
{ match হলে সেটা handle হবে
}

E-84 → Example.

```
int main()
{ int a,b;
  cin>>a>>b;
  int x = a/(a-b);
  cout<<"x"<<x;
  return 0;
}
```

a=20, b=10
x = 20/(20-10) = 2

a=10 b=10
x = 10/(10-10)
= 10/0 → Exception

```
int x = (a-b);
try { if(x!=0)
      cout<<(a/(a-b));
    else throw (x);
}
```

Exception:

```
int x = (a-b);
try {
    if (x != 0) cout << (a/(a-b));
    else   throw(x);
    }
catch (int i)
{   cout << "In Division By zero";
}
return 0;
}
```

এখানে catch/handling করার সুবিধা হলো program এর অনেক বড়বড় ~~handling~~ exception ~~থাকলেও~~ program এর বাকি অংশ execute হবে।

S-76 ~~...~~ - Same like

5-76

E-85 → Multiple catch statements.

→ যদি exception throw করার জন্য more than one condition needed.

```
try { if (n>0) throw (n0);
      else throw (ch);
    }
catch (type arg)
  { catch block1; }
catch (Type arg)
  { catch block2; }
catch (Type arg)
  { catch Blockn; }
```

type different

```
int main()
{ cout << "multiple catch statements";
  cout << "x>0";
  test(5);
  cout << "x<0";
  test(-3);
  return 0;
}
```

Program:

```
void test (int x)
{ try { if (x>0) throw x;
        else throw 'x';
      }
  catch (int n0)
  { cout << "caught an integer" << n0; }
catch (char ch)
  { cout << "caught a character" << ch;
    cout << "next statements" << endl; }
```

character

S-76 Exception handling Advance

```
int main()
{ try {
    cout << "Try" << endl;
    throw throw (1.0);
    cout << "Try end" << endl;
  }
  catch (int e) {
    cout << " catch " << e << endl;
  }
  return 0;
}
```

→ এখানে double value আসতে তাহলে program stop হয়ে যাবে catch এ int দিয়ে

/// It will not execute because it is after throw.

S-77 Exception handling and Function.

→ আমরা যদি try block এর function call করি এবং exception throw করি এবং সেটা আমরা পরবর্তীতে পাবো এবং এটা local variable এর মাঝে কাজ না করে Global variable এর মাঝে কাজ করবে।

Example:

```
void positive(int a)
{  cout << a << endl;
   if (a <= 1) throw a;
}
int main()
{
  try {
  positive (5);
   positive(-5);
   positive (6);
  }
  catch (int e) {
   cout << "Exception" << e << endl;
  }
  return 0;
}
```

output:
5
-5
Exception -5

positive(6) call হচ্ছে না কারণ (-5) এর জন্য exception throw হচ্ছে তাই পরের statement আর execute হচ্ছে না।

So catch user defined main এর value পাঠাবে।

So acts like global variable.

যদি 3 টি function এ call করতে চাই:

```
void positive (int a)
{  cout << a << endl;
   try { if (a < -1) throw a;
   }
   catch (int e)
   {  cout << "Exception" << e << endl;
   }
}
main()
{
 positive(5);
 positive(-6);
 positive(6);
}
```

S-78 - Exception Handling class Type Exception.

```
class exep {
    int id;
    string name;
public:
    exep (int i, string n) {id = i; name = n;}
    void print() {cout << "Error id:" << id;
                  cout << name << endl;}
};
void positive (int a)
{ cout << a << endl;
  try { if (a<1) throw exep (1, "Not a positive num");}
    catch (~~int~~ exep e) {
      e.print();
    }
int main()
{ positive (5);
  positive (-5);
  positive (6);
}
```

S-79 → Multiple catch statement

Info from E-85

program:

```
int main()
{ try { cout<<"TRY"<<endl;
          throw 1.5;
      }
  catch (int e) { cout<<"Exe int" <<e; }
  catch (double e) { cout<<"Exe double <<e; }
  return 0;
}
```

S-80 Exception ~~and~~ handling and Inheritance.

```
class Base { };
class Derive : public Base { }
int main()
{ Base b; Derive d;
  try{ throw b;
     /// throw d;
  catch (Derive e) { cout<<" Der Exe" <<endl; }
  catch (Base e) { cout<<" Base Excep" <<endl; }
  }
  return 0;
}
```

এই catch যদি exchange on first এ base than derive সেক্ষেত্রে হবে তাহলে কারণ অবশ্যই Base class catch করবে derive এর মধ্যে Base থাকে তাই উপর আগে call catch হবে

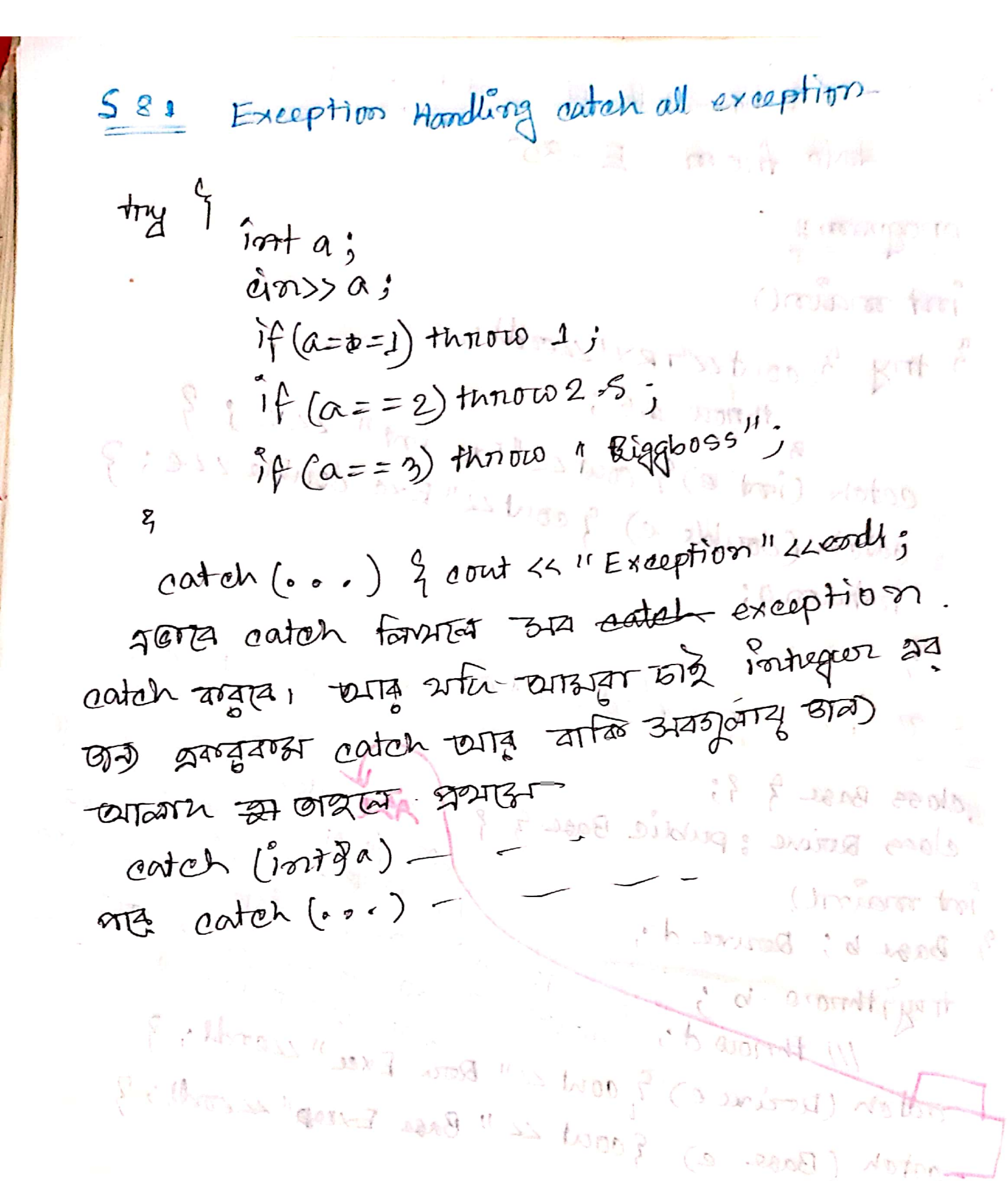
S 8 : Exception Handling catch all exception

```
try {
    int a;
    cin>> a;
    if(a==1) throw 1;
    if(a==2) throw 2.5;
    if(a==3) throw "Biggboss";
}
catch(...) { cout << "Exception" <<endl;
```

এভাবে catch লিখলে এর ~~catch~~ exception এ catch করবে। আর যদি আমরা চাই integer এর জন্য একরকম catch আর বাকি অবজুলোর জন্য আলাদা তাহলে প্রথমে

```
catch (int a) ——
```

আর catch(...) —



S-82 Exception Handling Restricting Exception

→ আমরা যদি কোন specific data type throw করতে চাই।

```
void excep(int a) throw(int)
{ if (a==1) throw 1.2;
  if (a==2) throw 1.2;
}
int main()
{ try { excep(2); }
  catch(...) { cout<<"Exception";
  return 0;
}
```

throw(int) → একাধিক দিলে (int, double)

→ না throw করলে ( )

→ যে গুলো দিয়েছি দিবো

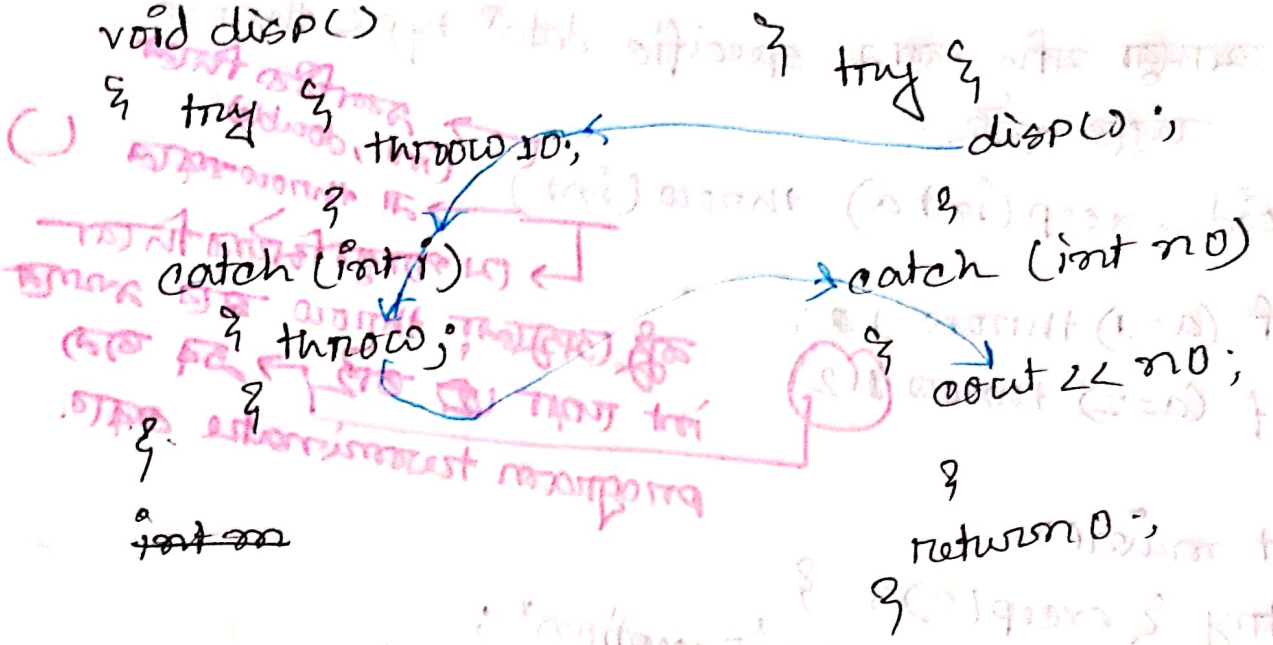
কিন্তু অন্যগুলো throw হবে এখানে int দেয়া তাই → এর জন্য program terminate করবে.

E-86: Re Throwing an Exception. (Nesting of try block)

```
try  → outer
{
  try  → inner
  { throw 10;
  }
  catch (int i)
  { throw;   → without any value.
  }
}
catch
```

Example:

```
void disp()
{ try {
    throw 10;
  }
catch (int i)
  { throw;
  }
}
int main()
{ try {
    disp();
  }
catch (int n0)
  { cout << n0;
  }
return 0;
}
```



(4)

```
void function (int a, int y)
{ cout << "\n\n Inside function";
  try { if (y=0) throw y;
        else cout << "\n\n Division : " << (x/y);
      }
  catch (int i) {
    cout << "\n\t Caught Integer Inside function" << i;
    throw;
  }
  cout << "\n\n End of the Function";
}
```

```
int main()
{ cout << "\n\n Inside main ";
  try { function (10,5)
        /// function (10,0);
      }
  catch (int no)
       { cout<< "caught int in main " << no;
       }
  cout << "End of main" <<endl;
return 0;
}
```

§ 5-83 → Re throw (optional)

§ 84 → Ex handling Terminate and unexpected

```
void excep() throw(double)
{ throw 1;
}
void T() { cout<< "Terminate"; }
void U() { cout<< "Unexpected "; }
int main()
{ set_terminate (T);
  set_unexpected (U);
  try { excep(); }
  catch (int a) { cout<<a ; }
  return 0;
}
```

throw করছে double কিন্তু catch করছে int so unexpected error. throw করছে double catch করছে terminate.

S-85 Exception handling Final Example

```
void divide(double num, double demo){
    try { if (demo == 0) throw 0;
        cout << "Result : " << num/demo << endl;
    }
    catch (int e){
        cout << " can't divide something by zero" << endl;
    }
}
int main()
{ double num, denom;
do { cout << endl; numerator (0 to stop): ";
    cout << " Enter denominator
    cin >> num;
    cout << " Enter Denominator: ";
    cin >> denom;
    divide(num, denom);
} while (num != 0);

return 0;
}
```

The STL → collection of predefined class

SS 32 Introduction to STL

→ standard template library

→ It's a powerful set of c++ template classes.

• At the the core of the c++ ~~standard~~ STL are following three well structured components.

- containers
- Algorithms
- iterators

Functions → ফাংশন

Template required

↳ generic programming concept.

⇒

⊞ Containers:

→ are used to manage collections of objects of a certain kind.

→ container library in STL provide containers that are used to create data structures like arrays, linklists, trees etc.

→ These container are generic, they can hold elements of any data type.

Example:

- vector can be used for creating dynamic arrays of char, integer, float and other types.

Algorithms

- Algorithm act on containers. They provide the means by which you will perform initialization, sorting, searching, and transforming of the contents of containers.

- Algorithms library contains built in functions that performs complex algorithm on the data structures

Example:

- One can reverse a range with reverse() function, sort a range with sort() function, search → binary_search()

- Algorith library provides abstraction, i.e you don't necessarily need to know how the algorithm works.

Iterators

- Used to step through the elements of collection of objects. These collections may be containers of subsets of containers.
- Iteration is STL are used to point the containers.
- Iterations actually acts as a bridge between containers and algorithms.

Example:

- sort() algorithm have two parameters, starting iterator and ending iterator, now sort() compare the elements pointed by each of these iterators and arrange them in sorted order, thus it does not matter what is the type of the container and same sort() can be used on different types of containers.

SS33 Containers C++

— a collection of classes
— are implemented as generic class templates.

— helps us to implement, replicate, simple and complex data structures very easily like arrays, ~~link~~ lists, trees associative arrays ...

→ used to hold different kind of objects.

~~classes~~ — Common containers

- vector : replicate arrays
- queue : replicate ~~que~~ queues.
- stack : replicate stack
- priority_queue : replicate heaps.
- list : replicate linked list
- set : replicate trees.
- map : associative arrays.

classifications:

- Sequence containers
  - Like arrays, linked list etc.
- Associative ~~array~~ containers
  - Stored data structure like map, set.
- Unorder ~~ass~~ associative containers.
  - Unsorted data structures.
- Container adapters
  - Interfaces to sequence containers

```
#include<iostream>
using namespace std;
#include <list>
int main()
{ ~~list~~ list <int> mylist;
}
```

int → any type.

SS 34 Array is STL

- linear collection of similar elements.
- array container in STL provides wrapper implementation of static array
- header file #include <array>

```
template <class x, class y>
class array {
}
```

declare array <int, 4> obj;

array <object type, size> array name;
array size

↳ It creates an empty array of object type with maximum size of array size

→ compiler setting needed.

array <int, 4> odd = {1, 3, 5, 7}

most used array functions.

- at
- [] operator
- front()
- back()
- fill()
- swap()
- size()
- begin()
- end()

at() : • This method returns value in the array at the given range.

• If the given range is greater than the array size, out_of_range exception is thrown.

```
array <int, 5> data-array = {11, 22, 33, 44, 55};
                              0   1   2   3   4
cout << data-array.at(2);   → 33
cout << data-array.at(5);
```

→ terminate called after throwing an instance of 'std::out_of_range' what(): array::at (E)

[] operator : → use এ আগের মতোই

```
cout << data-array[3];
```

front() : → Returns first element

back() : → Return last element.

```
cout << data-array.front();
cout << data-array.back();
```

→ কিন্তু array <int, 8> data_array = {11, 12, 1, 3} অর্থাৎ বাকি element নেই তাহলে return করবে zero.

• fill() : assigns the given value to every element of the array.

data_array.fill(10); → আগের কোন value assign আছে বা না আছে অবস্থাতে 10 assign হবে।

• swap();

— swaps the contents of two array of same type and same size.

— It swaps index i of first array to the index of second array.

— first_array.swap(second_array);

size() → Returns number of element in the array.

cout << data_array.size();

আমি যদি size 5 করে 4টা element initialise করে রাখি তবুও size আসবে 5.

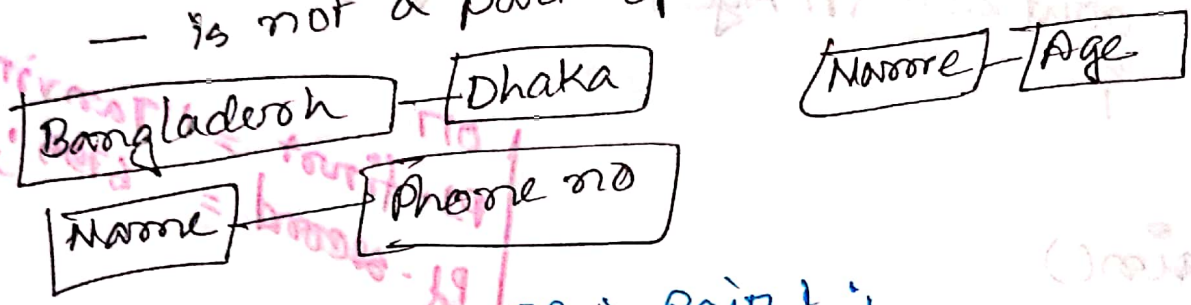
begin(): → returns the iterator pointing to the first element of the array.

end(): → returns an iterator pointing to an element ~~in the array~~ next to the last element in the array. —Later আলোচনা হবে।

#include <utility>

SS 35 Pair in STL

— is a template class in STL
— is not a part of container.



- pair <T1, T2> Pair1;

Example: pair <string, int> P1;

inserting value:

P1 = make_pair("Tamim", 6705);

printing: cout << P1.first << endl;
cout << P.second << endl;

Example:

```
class student{
    string name;
    int age;
public:
    void setStudent(string s, int a)
    { name = s; age = a;
    }
    void showStudent() {
    cout << "\n Name: " << name;
    cout << "\n Age: " << age;
    }
};
int main()
{ pair < string, int > P1;
  pair < string, string > P2;
  pair < string, float > P3;
  pair < int, student > P4;
  P1 = make_pair ("Tanvir", 16);
  P2 = make_pair ("Bangladesh", "Dhaka");
  P3 = make_pair ("ANSI C", 345.50f);

  student S1;
  S1.setStuden ("Tanvir", 22);
```

or

```
P1.first = "Tanvir";
P1.second = 6795;
```

```
P1.make_pair(_,_);
P1.make_pair(P2);
```

~~s1. set Student~~

```
p4 = make_pair (1, s1);
cout<< "in pair1"<< endl;
cout << p1.first << p1.second;
            pair 2
cout << p2.first << p2.second;
            pair 3
cout << p3.first << p3.second;
            pair 4
cout << p4.first << " ";
cout << p4.second;
student s2 = p4.second;
s2.showStudent();

return 0;
}
```

For objects

দুইটা pair কে আমরা compare ও possible ==, != - - - -

: swap() → pair1.swap(pair2)
or swap (pair1, pair2)

SS-36 Tuple [Not a container]

- Just like in pair, we can pair two heterogeneous objects; in tuple we can pair multiple objects.
  - tuple <T1, T2, T3> tuple1;

```
#include <tuple>

int main()
{
    tuple <string, int, int> t1;
    t1 = make_tuple("TutorialCup", 15, 30);
    cout << get<0>(t1) << endl;
    cout << get<1>(t1) << endl;
    cout << get<2>(t1) << endl;

    return 0;
}
```

position no

to access

SS-37 :→ Vector class

- The most general purpose container is the vector.
- supports a dynamic array.

capacity increases in double ☹

different from linked list

- Array size is fixed, no flexibility to grow or shrink during execution.
- vector can provide flexibility.
- #include <vector>

vetor <int> v1; → not mandatory to mention the size. at declaration

└→ here the initial size is 2020.

→ code will create a blank vector.

vector <int> v1 {10, 20, 30}; এভাবে initialize

3 নম্বর যাবে

└→ allowed in ctt 11

→ setting ~~ela~~ change required.

→ by default capacity 1.

→ initially capacity 3 here

→ আমি যদি অনেকগুলো value রাখতে চাই তাহলে বহু capacity double হয়ে যাবে মানে 6 টা।

- vector <int> v1 ; → zero length
- ~~vec~~ vector <char> cv(5); → 5 elements of char type

: ~~vector <char> v~~
vector <int> v3 (5, 10) ;

→ ~~इस~~ आज 5 ~~के~~ index ~~में~~ 10 ~~को~~ store ~~करो~~

v3

10	10	10	10	10

, vector <string> v4 (3) ;

→ 3 string blocks ✓

→ If I want to make 3 block with by default string vector <string> v4 (3, "Hello") ;

(*) Relational operators are allowed.

cout << v4[0] << endl ; → hello
v4[1] → hello
v4[2] → hello

1	2	3	4	5	

vec.begin()

vec.end()

- **push_back():** is a member function, which can be used to add value to the vector at the end.

v1.capacity() returns

```
int main()
{
    vector<int> v1;
    cout << v1.capacity() << endl;   → 0
    v1.push_back(10);
    cout << v1.capacity() << endl;   → 1
    v1.push_back(8);
    cout << v1.capacity() << endl;   → 2
    v1.push_back(89);
    cout << v1.capacity() << endl;   → 4
    v1.push_back(5);
    cout << v1.capacity() << endl;   → 4
}
```

- **pop_back():** → It removes the last element.

```
vector<int> v1;   → print capacity → 0
for (int i=0; i<=9; i++)
    v1.push_back(10*(i+1));
cout << "capacity :" << v1.capacity();   → 16
```

```
cout << "After pop" << endl;
for (int i = 0; i < 10; i++)
{
    v1.pop_back();
    cout << "capacity :" << v1.capacity;
}
```

→ capacity will remain the same

So capacity doesn't reduce.

size() : → size returns the number of elements currently in the vector

size and capacity are different.

* **clear()**: removes all the elements.

v1.clear();

↳ size → 0
↳ capacity ⟶ 16

at(): v1.at(3) → return the value of index 3 → position not element

• erase( ) → only position দেওয়া যাবে, or position range দেওয়া যাবে

front() / back() : returns first and last value respectively.

iterator : → like pointers

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

here it points the first element → value

To insert v1.insert(it+2, 35);
→ index no

• remove()

→ remove(first index, last index, element);

SS-38 List class

- List class supports a bidirectional list.
- vector supports random access but a list can be accessed sequentially only.
- List can be accessed front to back and back to front.

List <int> l1; → empty list

List <int> l1 {1, 2, 3};

List <string> l2 {"Dhaka", "Dinajpur"};

cout << l1[0]; → can't be used in List

* We have to use iterator

```
List<int>::iterator p = l1.begin();
while(p != l1.end())
{ cout << *p << "  ";
  p++;
}
cout << "Total value" << l1.size();
```

l2 → same process.

```
l2.push_back("Dinajpur");
l2.push_back("HSTU");
```

This process.

pop_front

We can also use pop back also.

Codes → 5538 ~~+0 150+ lines~~

- sort();

```
List<int> l1 {6, 9, 2, 4, 10, 2, 8};
l1.sort();
```

print the elements.

~~l1.begin~~ l1.reverse();

• remove( ) :→ to remove particular element of a list. ⁂ → element not index

ll. remove(44) → যখন element টি list এ না থাকে তাহলে কিছুই remove হবে না।

যদি same element more than one time থাকে তাহলে সবার 3 টি remove হবে।

• clear() → removes all the elements.

ll.clear();

MAP

• are used to replicate associative arrays

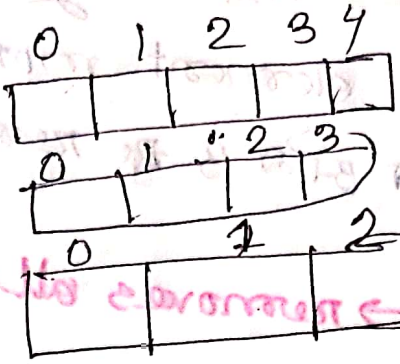
• maps contain sorted key-valued pair, in which each key is unique and can not be changed, and it can be inserted or deleted but cannot be altered.

Associative arrays :

আমরা আগের array গুলোর সবগুলো হচ্ছে numeric array.

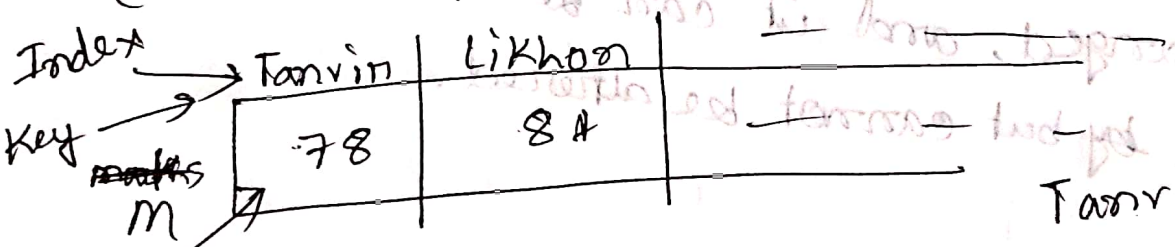
Numeric :

```
int a[5];
float b[4];
string s[3];
```



এখানে প্রতিটি element এর আগে একটি index থাকে a[0] = 2 . . . .

Associative : এই Array তে index এর বদলে string/text ব্যবহার করা যায়।



Tanvin 78
Likhon 84

m["Tanvin"] = 78;

m["Key"] = value;

NB : Keys must be unique and can't be changed

- Value associated with keys can be alter.

m["Likhon"] = 88;

- Keys can be of any type.
- Automatically keys will be sorted.

- #include <map>

int main()
{
map<int, string> customer;
customer[100] = "Gajendra";
customer[123] = "Dilip";
customer[145] = "Aditya";
customer[171] = "Shahid";
customer[200] = "Rajesh";

or,
map<int, string> c
{ {100, "Gajendra"},
{123, "Dilip"} ---
};

↓ at declaration time.

Customer Number (Key)	Customer name (Value)
100	Gajendra
123	Dilip
145	Aditya
171	Shahid
200	Rajesh

/* যেহেতু সরাসরি index
প্রদান করে ব্যবহার করা যাই
তাহলে (iterator) use
করতে হবে value print
করার জন্য */

cout << customer[100]; → Rajendra

cout << customer[101]; → No output

cout << customer.at(145) → doesn't exist.

↳ यहाँ 3 होता।

```
map<int, string>::iterator p = customer.begin();
while (p != customer.end())
{
    cout << p->second << endl;
    p++;
}
```

key sorted होता, print होता।

value print होता।

First index print होता।

```
cout << customer.size();
cout << customer.empty();
```

→ returns 0 if the map is not empty else returns 1.

~~customer.insert(205, "Tanveer")~~

```
customer.insert(pair<int, string>
                (205, "Tanveer"));
return 0;
```

or,

```
customer.insert(make_pair
                (205, "Tanveer"));
```

}

iterator ⟶ #include <iterator>
sort ⟶ #include <algorithm>

vector sort (vec.begin(), vec.end());

↳ ascending

descending order ⇒ print করতে হবে

```
bool myfunc (int a, int b)
{ return (a>b);
}
int main()
{ vector <int> vec;
  vect <int>
  sort (vect.begin(), vec.end(), myfunc);
}
```

no need parenthesis

যদি যদি প্রথম 3 টি element sort করতে চাই,

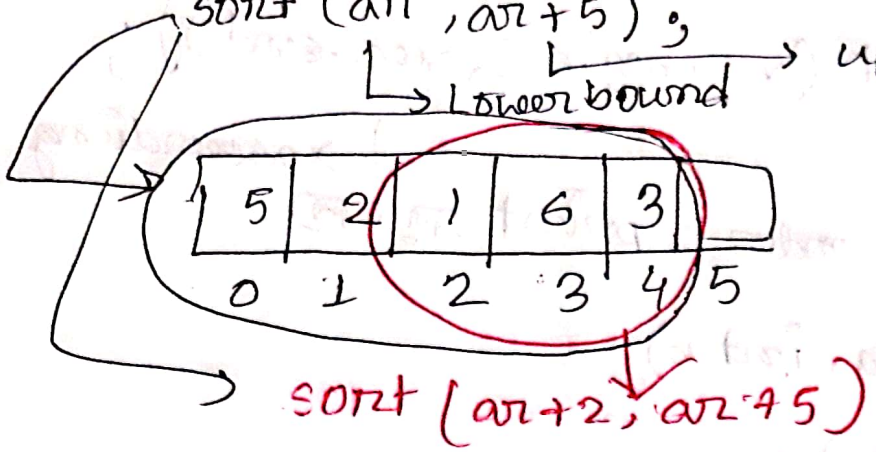
~~sort (vec.begin(), vec.begi~~

sort (vec.begin(), vec.begin()+3);

- `int ar[5] = { 5, 2, 1, 6 3 };`

```
sort (ar , ar+5);
```

ar → lower bound, ar+5 → upper bound.



VECTOR

learn ctrl + shift + space

- push_back()
- find() #include<algorithm>
- reverse();
- clear()
- sort() #include<algorithm>
- ~~erase()~~
- ~~remove~~
- erase()
- remove() → algorithm.

LIST

- push_back()
- push_front()
- pop_front()
- pop_back()
- find()
- reverse()
- clear()
- sort()
- erase()
- insert()

remove( ) ⇒

T-15

SET → #include <set>

* automatically sorted
* can't insert duplicate value

```
int main()
{
  set<int> s;
  set<int>::iterator it;
  s.insert(10);
  s.insert(3);
  s.insert(5);
  for(it = s.begin(); it != s.end(); ++it){
    cout << *it << " ";
  }
```

→ 3, 5, 10 ascending order

```
  it1 = s.find(3);
  s.erase(it1);
  s.erase(s.find(5));

  return 0;
}
```

element → to remove

insert() function → pair return करता
first → iterator, second → boolean return करता..

CODE →

```
int main()
{
    set<string> s;
    set<string>::iterator it;
    s.insert("dipta");
    s.insert("rezal");
    s.insert("Nabil");
    s.insert("Arnob");
    pair<set<string>::iterator, bool> p;
                 iterator
    p = s.insert("arnob");
    if (p.second == false)
        cout << "Can't insert" << endl;
    else
        cout << "Inserted" << endl;
    return 0;
}
```

STL -16 STACK (pop operation)

→ #include <stack>

iteration can't be applied in stack.

```
int main() {
    stack <string> s;
    s.push("Tanveer");
    s.push("Ahmed");
    s.push("Likhon");
// cout<<s.top()<<endl;
// s.pop();
  while(!s.empty())
  {  string x;
     x=s.top(); cout<<x<<endl;
     s.pop();
  }

return 0;
}
```

not push back

top element দেখাবে।

remove top element

যতক্ষণ stack empty না হবে ততক্ষণ।

STL - 17 priority queue #include<queue>

— যে element এর priority বেশি সে element টি উপরে চলে আসে।

```
int main()
{ priority_queue <int> q;
    q.push(400);
    q.pus(100);
        300
        10
        7
        9;

while (!q.empty()) {
  int x; x = q.top();
  cout << x << endl; → 400, 300, 10, 9, 7
  q.pop();
}

return 0;
}
```

q.front use করলে error দেয়

Actually descending order for int and lexicographical for string descen...

STL - 18 multiset/map.

map এর queue এ key ≠ unique value রাখতে হয়। এই অসুবিধা দূর করতে multimap ব্যবহার করা হয়।

```
int main()
{
    multimap<string, int> m;
    multimap<string, int>::iterator it;
    m.insert(make_pair("Tanvir", 6));
    m.insert(make_pair("Tanveer", 8));
    m.insert(make_pair("Tanvir", 9));
    m.insert(make_pair("Tanvir", 5));

    for(it=m.begin(); it!=m.end(); it++) {
    cout<<it->first<<" "<<it->second<<endl;
    }
    return 0;
}
```

```
int main()
{ multiset<int> s;
  multiset<int>::iterator it;
  s.insert(100);
  4 times same line
  s.insert(1);
for(it=s.begin(); it!=s.end(); it++) {
  cout<< *it << endl;
  }
return 0;
}
```

→ ~~10~~ 1, 100, 100, 100, 100, 100

Tuto complete ☺

The STL is a set of c++ templates classes to provide common programming data structures and functions such as list, stack, array etc.

Sequence containers.
Associative containers.
Derived containers.

STRING

Saurav Sukla

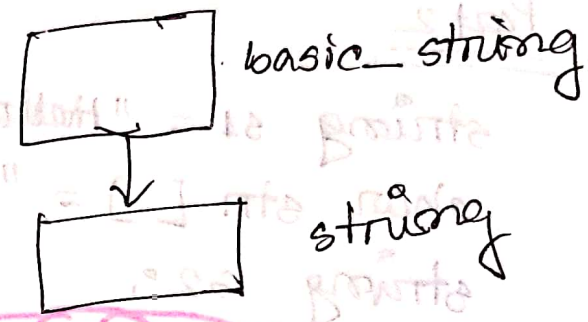
LECTURE - 40

char s1[10] = "Hello" এভাবে করলে C++ এর সব সুবিধা applicable হবে না।

s1 = "Likhon" // wrong

The string class is a specialization of a more general template class called basic_string.

string s1; → object



Character array is faster than string

string is better for easy operation & and safety

String is safe than character array.

#include<string>

string → container

- String class supports many constructors.
  - string()
  - string (const char *str)
  - string (const string & str)

string s1 = "Tanveer";

char str[] = "Hello";

string s2 = str; ✓ correct

string s2(str); ✓ " } constructor calling

string s3 = s2; ✓

nothing more

Part 2

```
string s1 = "Hello";
char str[] = " students";
string s2;
s2 = s1 + str;
cout << s2 << endl;
```

→ at least one must be string type and other any c style string or character.

• **assign() :**

```
string s1;
s1.assign("Hello");
cout << s1;
```

• **append() :**

```
s1.append(" students"); // s1 = s1 + " students";
cout << s1  → Hello students.
```

• **insert():** string s1 = "Hello";
0 1 2 3 4 → index no:

```
s1.insert(2, "123");
cout << s1  → He123llo
```

• **replace()**

```
s1.replace(2, 2, "A");
```

Hello → 'A'

2 → index

2 → length আগে যতগুলো replace করতে চাই,

→ He(ll)o → 2-টা থেকে 1-টা A হবে।
A

```
s1.replace(1, 3, "Tanveer");
```

✓ Correct

Scanned by CamScanner

- erase()

s1.erase() → এর erase হয়ে যাবে

s1.erase(4,9)

- find();

string s1 = "Tanveer Ahmed Likhon";

int i = s1.find("Ahmed");

→ returns index

cout << i; → 7

যদি not found হয় তাহলে -1 return করে

একই element/string যদি দুই বার থাকে তাহলে first এর index print হবে।

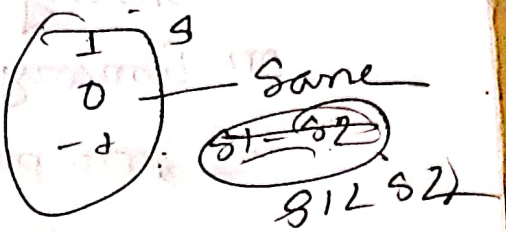
আরেকটি function • rfind(); এটা Last থেকে reverse দিকে search করবে এবং Last occurance এর index return করবে।

• compare ( ):

```
string s1 = "Tanveer";
string s2 = "LIKHITHA";
int i = s1.compare(s2);
cout<< i ;
```

→ difference

0 → for the same



• c_str( ) ;

```
string s1 = "Hello";
char str[20];
strcpy(str, s1.str());
```

• size() cout<< s1.size( ) ;

• begin( ) / .end( ) → returns iterator.

```
string s1 = "Bigg Boss";
string :: iterator it=s1.begin( ) ; or end( ):
cout <<*it ;
```

→ (B)

• rbegin( ) / .rend( ) → last to first start

```
string s1 = "Tanveer";
string :: reverse_iterator it = s1.begin( );
cout<<*it ;
```

→ r ;

- max_size():

Returns the maximum number of elements the string is able to hold due to system or library implementation limitations

→ no parameters.

→ returns maximum number of characters

→ no exception.

```
string s;
cout << s.max_size();
```

- resize

- resizes the string to a length of n characters

→ If n is smaller than the current string length, the current value is ~~sort shorted to~~ shortened to its first n character, removing the characters beyond n.

→ If n is greater than the current string length, the current content is extended by inserting at the end as many

characters as needed to reach a size of n. If c is specified, the new elements are initialized as copies of c, otherwise they are value-initialized characters (\0).

→ parameters.

n → new string length, expressed in number of characters.

c → character used to fill the new character space added to the string.

```
int main()
{  string str = "I like to code in C";
   cout<<str;
   unsigned sz = str.size();
   str.resize(sz+2,'+');
   cout<<str;
   str.resize(6);
   cout<<str;
   str.resize(100);
   cout<<str;
   return 0;
}
```

"I like to code in C" → I like to code in C++

→ first 6 character.

→ full accordance → last remained elements will be '\0'.

- clear()
- capacity() → আগের same
- empty()
- reverse();

```
string str = "Tanmoy";
reverse(str.begin(), str.end());
cout<<str;
```

- shrink_to_fit();

আমরা আগের vector কমাতে capacity reduce করতে পারিনি কিন্তু এ function দ্বারা আমরা Reduce করতে পারব।

```
int main()
{ vector<string> str = {100, "x"};   → 100 টি "x" string character
  vector<string>::iterator it = st
  cout<< str.capacity() <<endl;   → 100
  str.resize(10);
  cout<< str.capacity() <<endl;   → 100
  str.shrink_to_fit();
  cout<< str.capacity();   → 10
  return 0;
}
```

- push_back () → same
- back() → last character ~~change~~ return too.

string str = " hello world."";
str.back() = '!';
cout << str.back();; → !
~~str~~ cout<< str << endl → hello world!

- front() like back().
- at() → same
- pop_back() → same

- **erase():**

Erases part of the string, reducing its length.

① Sequence:
Erases the portion of the string values that begins at the character position pos and spans len character on until the end of the string.

string str = " Tanner";
str.erase(4) → it will erase all the ~~ch~~ characters from index 4.
str.erase() → erase all.
str.erase(4,7) → erase index from 4 ~~to~~ to ~~7 char~~ next 7 characters including index 4.

Scanned by CamScanner

we can erase the all characters of string iterators.

str.erase(str.begin(), str.end());

- find_first_of (' ')
- find_last_of (' ')

parameter characters returns index

Substring:

substr()

This function takes two values pos and len as an argument and returns a newly constructed substring object with its value initialized to a copy of a sub-string of this object. copying of string start from pos and done till pos+len means.

[pos, pos+len]

→ if pos = string length returns empty string

→ <pos> string length throws out_of_range and no changes in the string.

→ if requested sub string len is greater than size of string, then returned sub-string [pos, size()).

⇒ string str = "TANVER";
string st = str.substr(1,3);
cout << st; ⟶ ANV;

NEVER GIVE UP
CODE CODE CODE

E-78 → Virtual Function

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

pointer দিয়ে disp() call করলে base class এর virtual disp() ফাংশনকে call করে।

virtual function use করে runtime polymorphism support করে।

```
class Base {
    public:
        void disp(void) { cout << "In Base class"; }
    virtual void show (void) { cout << "In Bas class show"; }
};
class D :Derived : public Base
{ void disp() { cout << " Derived class disp()"; }
void show() { cout << " Derived class show()"; }
};
```

```
int main()
{
    Base *p
    Base objb;
    Derived objd;
    p = &objb;
    p -> disp();
    p -> show();
    p = &objd;
    p -> disp();
    p -> show();
    return 0;
}
```

→ Base class disp
→ Base class show
→ Base " "
→ Derived show()

শেষ হলো কারণ show() ফাংশনটি virtual keyword দিয়া Base class > declare করা হয়েছে।

* non-~~virtual~~ virtual function আমলে তার memory compile করার সময় নির্ধারিত হয়। আর virtual ~~fun~~ function আমলে তার memory runtime > নির্ধারিত হয়। এর নাম dynamic binding — S 61

* constructor virtual হতে পারে না কিন্তু destructor হতে।

E-74 → Abstract Classes and Pure virtual Functions

আমরা যখন কোনো class বানাই তখন আমাকে at least one pure virtual function থাকতে হবে। ঐ class কে তখন Abstract class বলা হবে।

→ যে virtual function এর কোন body বা কোন definition থাকে না তাকে pure virtual function বলে।

Base ঐ বানাতে হবে।

```
virtual void disp() = 0;
```

এবং যে class এ আমরা এটা বানাবো তাকে Abstract class বলা হয়।

Abstract base class

→ যখন inherit করা হবে তখন derive class এর responsibility হবে pure virtual function এর definition দেয়া। এবং যদি definition না দেওয়া হয় তাহলে ঐ class ও Abstract class হয়ে যাবে।

Abstract class এর object বানানো যায় না। কিন্তু pointer or ref create করা যায়।

```
class Base
{ public:
      virtual void disp() = 0;
  };
class D : public Base
{ void disp() { cout << "Derived class"; }
};
int main()
{   D obj;
    obj.disp();
    return 0;
}
```

Base *obj; → error.

output
Derived class.

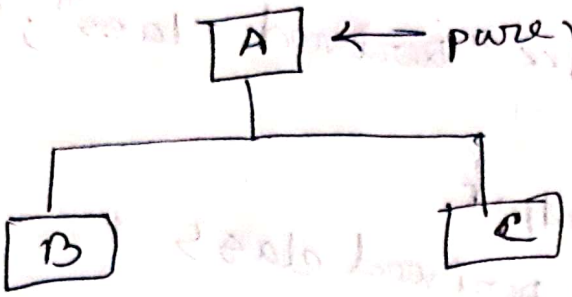
```
int main()
{   B *ptr;
    D obj;
    ptr = &obj;
    ptr -> disp()
    return 0;
}
```

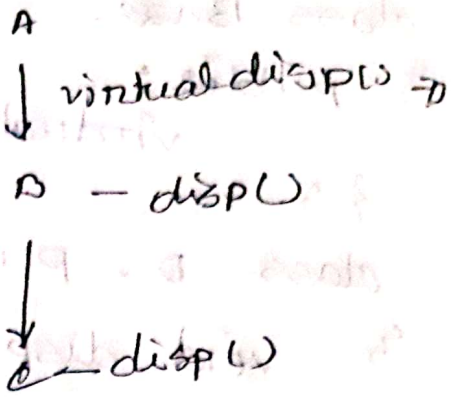
Derived class.

S-62 - nothing extra.

E-75 polymorphism

runtime polymorphism.






```
class A
{   public: virtual void disp() = 0;
};
class B : public A
{   int val1;
    public: B(int x) { val1 = x; }
    void disp() { cout << " Value is class <<val1; }
};
class C : public A
{   int val2;
    public: C(int x) { val2 = x; }
    void disp() { cout << " val2 is " << val2; }
};
```

```
int main()
{   A *bp;
    B objb(100);
    C objc(200);
    bp = &objb;
    bp -> disp();
    bp = &objc;
    bp -> disp();
    return 0;
}
```

output

val 1 in class B 100
val 2 in class C 200

Function overloading

↓

compile time overloading

```
int main()
{   A *bp[2];
    B objb(100);
    C objc(200);
    bp[0] = &objb;
    bp[1] = &objc;
    bp[0] -> disp();
    bp[1] -> disp();
    return 0;
}
```

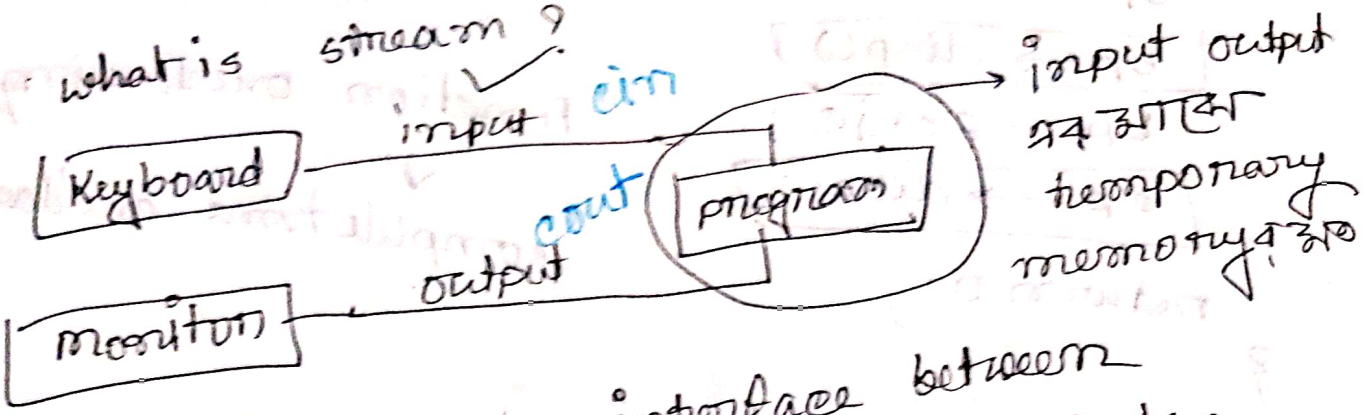
3.

E-76 File handling and Statement

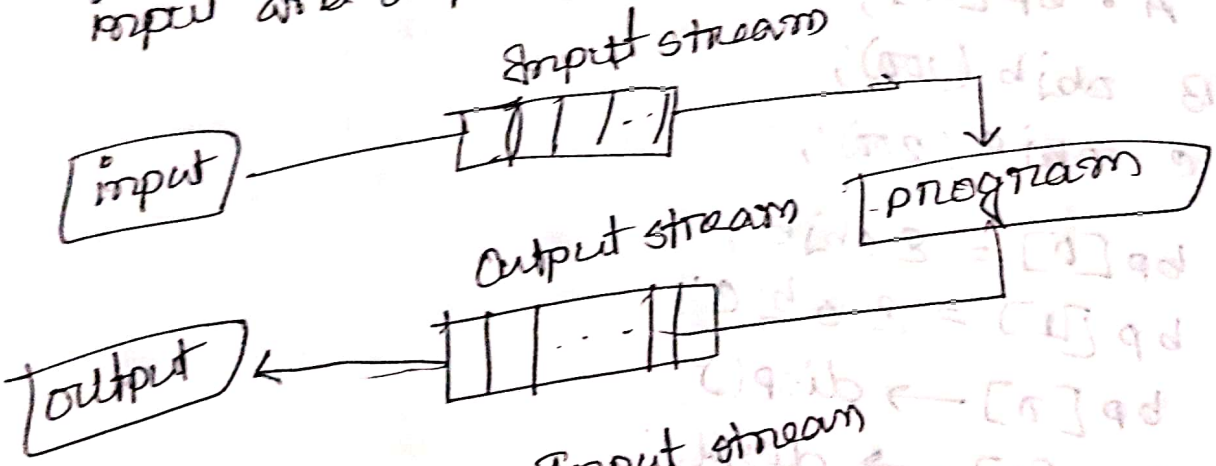
to store data ~~on~~ on hdd.

<iostream> → istream cin
→ ostream cout

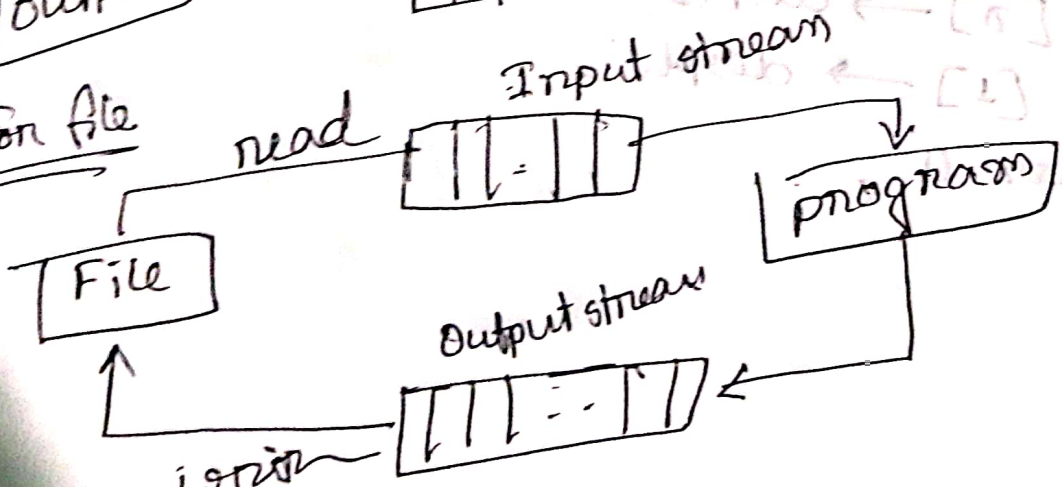
what is stream ?



stream works as interface between input and output. → sequence of bytes.



For file



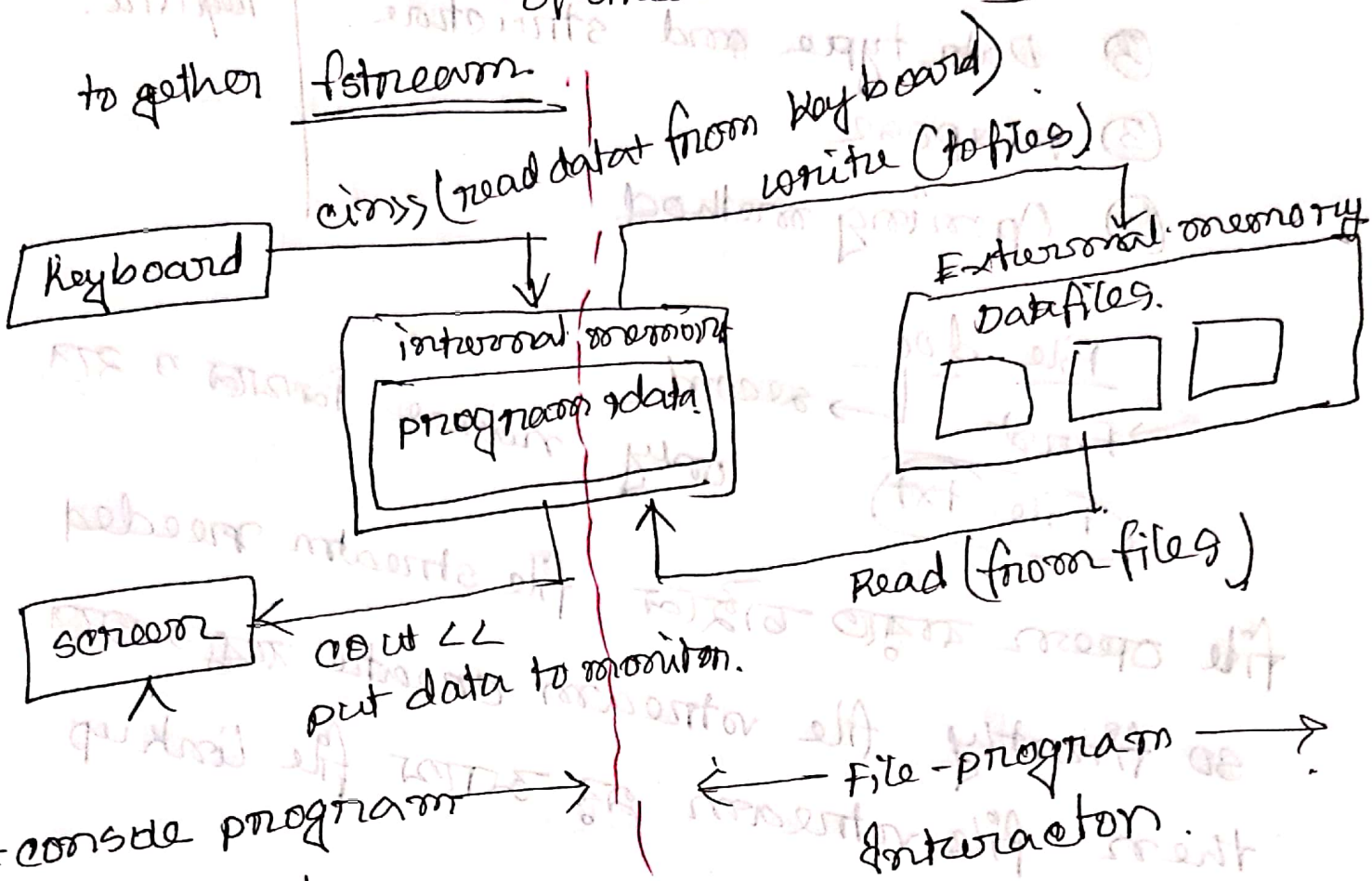
File → input → ifstream

File ← write ← output

ofstream

to gether fstream

S →



← console program → ← file-program →

interaction interaction

Scanned by CamScanner

E-77 ~~Eb~~ File open close.

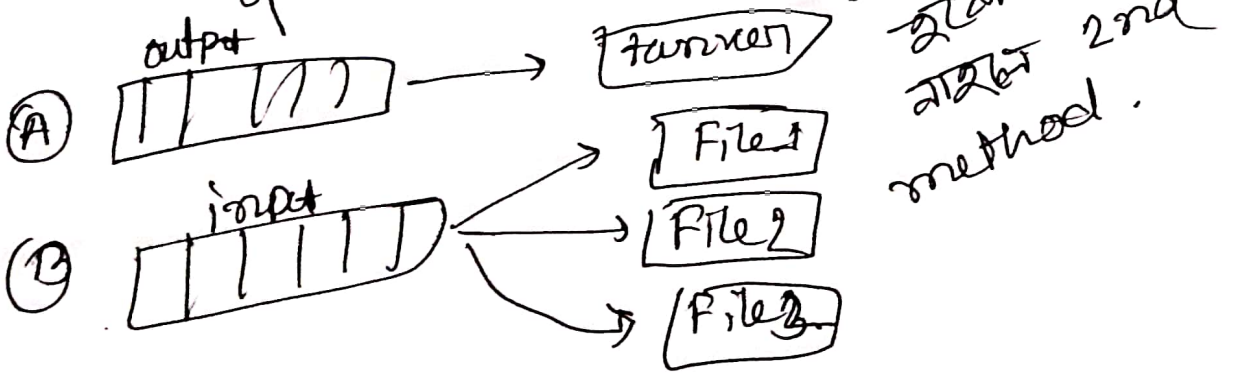
* ① Suitable name for file.
  ② Data type and structure.
  ③ purpose
  ④ Opening method.

require.

File-doc
First
second
File.txt
only name

file open করতে চাইলে file stream needed
so firstly file ostream create করতে হবে
then fileostream এর মাধ্যমে file link up
করতে হবে।

(File can be opened in two ways
① using constructor function of the class
② using member Function open()
of the class



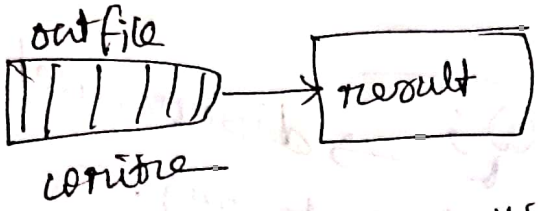
(A)

(1) ifstream, ofstream

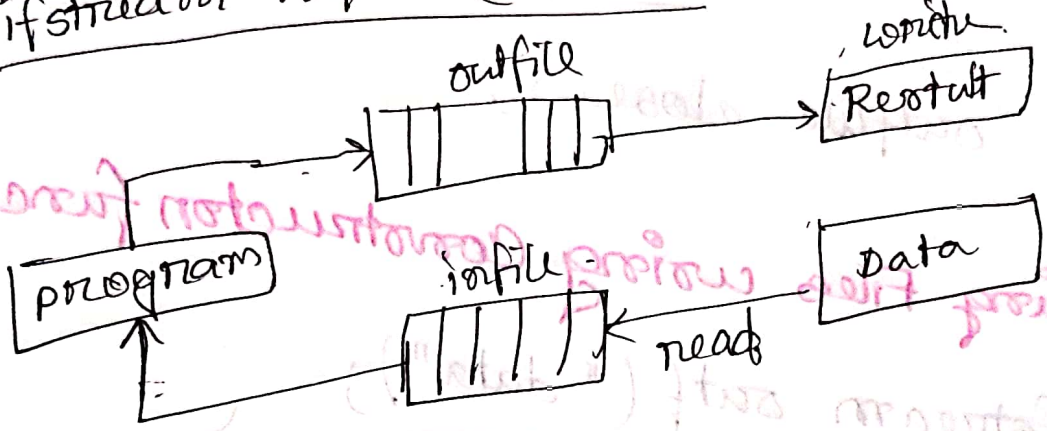
(2) File name ✓ → obj → creating file → open + connect it to outputstream

ofstream outfile ("result");

→ write file.



ifstream infile ("Data");



outfile.close();

infile.close();

→ closing file.

(B)

fileStream_class stream_object;

stream_object.open("file_name");

ofstream outfile;
outfile.open("Data 1");
---
outfile.close(); → disconnect mandatory
outfile.open("Data 2");
---
outfile.close();

E.g. Opening Files using Constructor function.

ofstream outf("data");
↳ যদি data নামের কোনো ফাইল আগে থেকে থাকে তাহলে তা open হবে এবং আগের data delete হবে।

ifstream inf("data"); to read.

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{ char name[20];
  float cost;
  ofstream outf("data");
  cout << "Enter item name: \n";
  cin >> name;
  cout << "Enter item cost";
  cin >> cost;
  outf << name << '\n';
  outf << cost << '\n';
  outf.close();
  ifstream inf("data");
  inf >> name;
  inf >> cost;
  cout << "\n item name" << name;
  cout << "\n item cost" << cost;
  inf.close();
  return 0;
}
```

E-g Opening Files using

```
int main()
{
    char name[20];
    int cost;
    ofstream fout;
    fout.open("data");
    fout << "Keyboard \n";
    fout << 350 << "\n";

    ifstream fin;
    fin.open("data");
    fin>> name;
    fin>> cost;
    cout << "\n Item name " << name;
    cout << "\n Item cost " << cost;
    fin.close();

    return 0;
}
```

E-80 open() Function File modes.

Parameters.

ios::in → ifstream এর default mode, এর কোনো ব্যতিক্রম না
↳ to only read from file.

ios::out → ofstream এর default
↳ to write only on file.
→ open এর আগে File এর beginning এ যাবে

ios::app → append
↳ to add something last of the file.

ios::ate
↳ open এর আগে file এর last এ চলে যাবে।
↳ end e content add করে + previous data modify করে।

ios::nocreate
↳ file open করতে হবে আগে যদি আগে কোন file না থাকে তাহলে কোনো file create হবে না এবং failed হবে.

ios::noreplace
↳ open এর আগে file exists থাকলে failed হয়ে যায়।

ios::binary
↳ If we want to create binary file

ios::trunc
↳ open করলে আগের existing file থাকলে তার data erase হয়ে যাবে।

ios::out ios::trunc আলাদা আলাদা by default open হয়।

যে বা যেকোনো operation এ use করা যাবে।

```
ofstream outf;
outf.open("data", ios::ate | ios::noreplace);
```

→ এমনে আগের যদি data থাকবে file exist করে তাহলে outf fail হবে।

E-81 Get() and Put()

get() → ifstream → get one character
put() → ofstream → print one character

① get(*char)
② get(void)

```
char ch;
cin>>ch;
```

Tanver Likhon

(A)

```
char ch;
cin.get(ch);
```

obj — member function

(B)

```
ch = cin.get();
```

↑ return character

Put

```
cout.put('E');
char ch = 'y';
cout.put(ch);
int main()
{
int
```

S-113 – Writing text.

```
int main()
{ ofstream out("text.txt");
  if (!out) cout << "can't open file" << endl;
  else
  { out << "Biggboss";
    out.close;
  }
return 0;
}
```

→ fn create करने के लिए

→ text file में data लिखने के लिए

S-114 Reading text

```
int main()
{   ifstream in("test.text");
    if(!in) cout<<"can't open file";
    else { string s;
           in>>s;
           cout<<s;
           in.close()
    }
    return 0;
}
```

→ File থেকে input নিবে।

→ screen এ output প্রদর্শন করবে।

S-115 put, get;

→ file এ 1 টি character print করবে।

→ file থেকে 1 টি character read করবে।

```
int main()
{   ifstream in("test.txt");
    if(!in) cout<<"can't ....";
    else{ while(in) {
            char ch;
            in.get(ch);
            if(in) cout<<ch;
        }
        in.close();
        return 0; }
```

for put

~~while~~ t

else

Tanver Likhon

For put

```
int main()
{ ofstream out("test.txt");
  if(!out) cout<<"can't ____ ";
  else { int i=0;
         char ch[100] = "Biggboss.com";
        while(ch[i] != '.')
          { out.put(ch[i]);
            i++;
          }
        out.close();
  return 0;
}
```

S-116 write, read → are used to read or write a full structure

```
struct student {
   char name[20];
   int std;
};
int main()
{ struct student a,b;
```

```
int main()
{ struct student a, b;
  strcpy(a.name, "Tanveer");
  a.sid = 6795;

  ofstream out("info.txt");
  if(!out) cout<<"can't ---- ";
  else {
       out.write((char*)&a, sizeof(struct student));
       out.close();
  }
  return 0; return 0;

}
```

to read:

```
ifstream in("info.txt");
if(!in) cout<<"can't ---- ";
else {
     in.read((char*)&b, sizeof(struct student));
  cout<<"Name: "<<b.name<<"\nID: "
                <<b.sid<<endl;
     in.close();
}

return 0
```

211-2

S-177 getline

```
int main()
{ ifstream in("test.txt");
  in.open("test.txt");
  if(!in) cout<<"can't ---;
  else
  { char str[100];
    in.getline(str,100);
    cout<<str;
    in.close();
  }
  return 0;
}
```

in.getline(str,100,'.'); → dot পর্যন্ত পড়বে।

→ 100 character পর্যন্ত input নিবে space with নিবে।

only while(in) use করলে 3 বার চলে।

S-178 EOF

```
while(!in.eof()){
  in.getline(str,100);
  cout<<str<<endl;
}
```

S-119 Flush

আমরা file এ যখন কোন কিছু লিখি তখন যতক্ষণ পর্যন্ত file close না করি ততক্ষণ পর্যন্ত Data source এ যায় না, তখন flush use করা হয়।

```
int main()
{
    ofstream out;
    ifstream in;
    out.open("test.txt");
    in.open("test.txt");
    out<<"hello";
    out.flush();
    char str[100];
    in>>str;
    cout<<str;
    out.close();
    in.close()
    return 0;
}
```

out.flush(); use করলে output দেখাবে।

এখানে garbage value আসবে কারণ file close করার আগে file থেকে read করছি।

এই কারণে file এখানে output দেখাবে।

S-120 File seekg() → for get
seekp() → for write

→ File এর position change করার জন্য use করা হয়।

→ File থেকে কোন কিছু নিতে চাইলে কোন position থেকে নিবো তার জন্য seekg এবং write করতে চাইলে কোন position থেকে write করব তার জন্য seekp

```
int main()
{
    ofstream out("text.txt");
    out.seekp(4);
    out<<"World";
    return 0;
}
```

→ প্রথমে first 4টা space ফাকা রেখে write হবে।

```
int main()
{
    ofstream out("text.txt");
    out<<"Hello World";
    return 0;
}
```

seekg

```
int main()
{
    char str[100];
    ifstream in("text.txt");
    in.seekg(3);
    in>>str;
    cout<<str;
}
```

S-121 File tellg, tellp → for read, → for write

→ return করে এই মুহূর্তে read/write করার অবস্থান সে পরিমাণ বাইট।

→ tellg return করলে তার value সেখান থেকে data read করবে

→ tellp return করলে সেখান থেকে আরম্ভ data ও তার write করলে তার value হবে

```
ofstream out("test.txt");
if(!out) cout<<"Can't ____ ";
else
{ cout<< out.tellp() <<endl; → 0
  out<< "Hello World";
  cout<< out.tellp() <<endl; → 11
  out<< "Hello World";
  cout<< out.tellp() <<endl; → 22
  out.close();
```

→ seekp() এটার পর বলা হবে সেখানে

For tellg

Suppose "Hello World".

else,

```
char str[100];
cout << in.tellg() << endl;        → 0
in.seekg(3);
cout << in.tellg();                → 3
in >> str;
cout << str;
cout << in.tellg() << endl;
in >> str;
cout << str << endl;
cout << in.tellg() << endl;
in.close();
```

S-122 File mode

→ out, in, app

→ আগে check করবে যে File টি আছে কিনা, যদি থাকে তাহলে তার Data মুছে দিবে। File না থাকলে নতুন করে create করবে।

→ check করবে File টি আছে কিনা। যদি থাকে তাহলে open করতে পারবে না।

→ check করবে File আছে কিনা। যদি কোনো তার মধ্যে থাকে তাহলে তার ভিতর থেকে Data write করবে।

ofstream out("test.txt", ios::app)

↳ আরো parameter.

E - 80 (আগে শুরুতে)

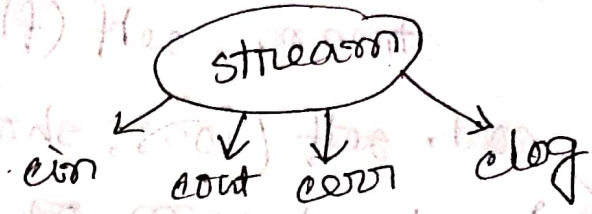
Ctrl file.cpp

S86 – IO intro

Input
* collect data from
* keyboard.

output
show output

string s;
cin >> s;
cout << s;
Tanveer Ahmed



S-87 → I/O Formatted output

```
char str[100];
cin.getline(str, sizeof(str));
cout << str
```

↳ by default cin.getline(str, sizeof(str), '\n');

যেকোন ভাবে আমরা new line পর্যন্ত নিতে

```
cin.getline(str, sizeof(str), '.');
```

↳ মধ্যে dot পাইলেই input নেওয়া off হবে যাবে।

Another way;

```
getline(cin, str);
```

S-88 Formatted Output

* ios class member → input output stream
* manipulator

→ class member ফিচার এবং ব্যবহার করলে flag set ব্যবহার হবে

```
stream.setf(flag)
cout.setf(ios::showpos);
```

→ cout stream এর এবং flag use ব্যবহারের showpos → number এর positive হলে তাহলে ব্যবহার positive sign দেখাবে।

manipulators

```
#include <iomanip>
cout<<hex<<100;
```

Flags

boolalpha
showbase
showpoint
showpos
skipws
unitbuf

uppercase
numerical
hex
oct
fixed
scientific
left
right

S-80 IOS member use

cout<< 12.34 << endl; → 12.34

cout.setf(ios::showpoint);

cout<<12.34<<endl; → 12.3400

4 digit পর্যন্ত দশমিক সংখ্যা দেখাবে

cout.setf(ios::scientific);

cout<<12.34; → scientific method এ দেখাবে

cout.setf(ios::uppercase);

↳ যদি আগে থেকে scientific notation এ e upper case এ E দেখাবে। এটা only scientific এ কাজ করে not for string.

cout.setf(ios::showpos);

↳ positive number এর আগে + sign দিবে।

S-90 ios মেম্বার unsetcheck.

```
cout<<12.34<<endl;
cout.setf(ios::scientific);
cout<<12.34<<endl;
cout.unsetf(ios::scientific);
cout<<12.34<<endl;
```

→ non scientific number দেখাবে

```
cout<< cout.unsetf(ios::dec);
      cout.setf(ios::hex);
       cout<<100<<endl;
```

→ basefield দিলেও হয়।

→ এই line টা না দিলে → এখানে hexadecimal না print করে decimal ই print করতো।

```
cout.setf(ios::showbase);
cout<<100<<endl;
```

কোন flag use করা হয়েছে কিনা আমরা যদি এটা জানতে চাই

```
ios::fmtflags flag;
flag = cout.flags();
if (flag & ios::showbase)
    cout<<"found";
```

fmtflags → format; flag → object

flag = cout.flags(); → এখানে অনেক flags include করা থাকবে

cout<<"found"; — showbase use হলে এটা দেখাবে

S-91 ios member setf Advance

→ আমরা যদি একসাথে অনেক flag use করতে চাই

```
cout.setf(ios::showpos | ios::showpoint);
cout << 1.23 << endl;
```
—— [illegible]

S-92 ios member Flags sets Type 2

→ setf function use না করে flag setup করা

```
int main()
{ ios::fmtflags f;
  f = ios::showpos | ios::showpoint;
  cout.flags(f);
  cout << 100.45 << endl;
}
```
→ [illegible] Flag [illegible]

~~1000~~

n = R 3 9.

R = n % 10

n = 123

~~R = 9 % 10~~

R =

n =

Sum = Sum + R, 10 × R. n %.

= 410 = 390

S-93 width, precision, fill.

```
cout.width(20);
cout<<"Tanveer"; —20
```

আমরা যদি width বলে দিই আর output ছোট থাকে তাহলে অবশিষ্ট জায়গা ফাঁকা থাকবে।

↳ 20 ঘর জায়গা নিয়ে প্রিন্ট করবে এবং বামদিকের ঘরগুলো ফাঁকা থাকবে।

আমরা যদি ফাঁকা জায়গা গুলো কোনো character দিয়ে fill করতে চাই তাহলে

```
cout.width(20);
cout.fill('.');
cout<<"Tanveer"<<endl;
cout<<"Tanveer"<<endl;
```

→ 20 ঘরের ফাঁকা ঘরগুলো dot দ্বারা পূর্ণ হবে।

↳ একবার দিলে width and fill কাজ করবে না। width and fill immediate পরের টাতে কাজ করে।

```
cout.precision(4);
cout<<12.456;
```

(4) → significant digit

→ এটা float and double এর ক্ষেত্রে কাজ করে only.

5-94 Left, right, adjust field

```
cout.width(20);
cout.setf(ios::left)
cout<<"Tanveer";
cout<<"Boss";
```

→ প্রথমে 20 ঘরের বামদিকে Tanveer অনুসরণ করবে এবং তারপর Boss লেখা আসবে.

5-95 Boolean Input/Output

bool a; a = 5 cout<<a<<endl → 1

otherwise 0

a = true / false

5-96 manipulators (ws) skipws, noskipws

ws → white space

আমরা যখন cin নিই তখন default ভাবে space গুলোকে skip করে

যদি চাই space skip না করতে তাহলে noskipws use করতে হবে।

```
char s;
cin >> noskipws >> s;
cout<<s;
cout<<(int)s;
```

→ এটার কারণে আর white space skip করবে না

ws দিতে হবে input এর আগে

cin>>ws>>k;

cout<<(int)s; → 32 print করবে অর্থাৎ space এর ascii value.

S07 → boolalpha , noboolalpha

cout<< true <<endl; → 1

আমরা যদি চাই true যেন true হিসেবে print হবে, তখন boolalpha flag use করতে হবে,

cout<< boolalph << true <<endl;

```
bool a;
cin>> boolalph>>a;
cout<< boolalpha << a;
```

noboolalpha দিলে 0 and 1 আসবে