26/10

Date / / Page no

Inheritance

→ Main motive: Reusability of code

→ Child classes borrow characteristics of parents with some additional features

→ Better design of code

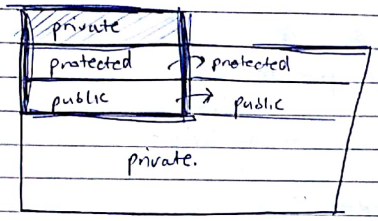
```
class student {
protected:
    int rollno;
    float height;
public:
    void getdata () {
        cout << " Enter roll no ";
        cin >> rollno;
        cout
        cin >> height;
    }
};

class studentdisplay : public class student {
public:
    void display () {
        cout << rollno;
        cout << height;
    }
};
```

→ Anything that is private, it cannot be inherited.

★ Therefore we have another access type modifier 'protected'

↳ It has all the characteristics of private but it can be inherited.

student



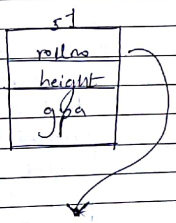
student display : public student.

```
int main() {
    studentdisplay s1;        ✓
    s1.getdata();             ✓
    s1.display();             ✓

    student s2;               ✓
    s2.getdata();             ✓
    s2.display();             X
```

Overriding

If we include a ~~exact~~ function with same signatures of a function which has been inherited. The ~~ds~~ child class's function will override the inherited func.

```
class studentdisplay : public student {
    float gpa;
    public: void display() {

    }

    void getdata() {                    ← This func will override the getdata from student.
        cout << "hello";
    }
    void entergpa() {
    }
};
int main() {
    student display s1;
```



First space is allocated for the parent class datamembers.

Write: student & studentdisplay with constructors.

```
class student{
};

class studentdisplay {
    float gpa;
public:
    studentdisplay(): student() { }
```

constructor of base class

```
studentdisplay(int x, float y, float z): student(x,y)
{
    gpa = z;
};
```

```
class student {
    int rollno;
    float height;
protected
    float gpa;
public:
};
```

```
class student.iiti : public student{
    int roomno;
protected:
public:
};
```

When we declare a student.iiti

rollno
height
gpa
roomno

General Class — Hierarchical Inheritance

More specific classes

Multilevel Inheritance

(Q)
```
class student {
protected:
    int rollno;
public:
    student() {};
};
class student.iith : public student {
protected
int roomno;
public:
    student.iiti(): student() {};
};
class studentug : public student.iiti{
protected: char branch;
public: studentug(): student.iiti(){};
};
```

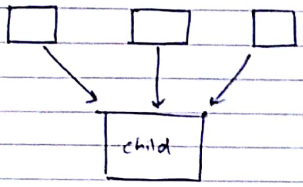
studentug (int x, int y, char c) : student (x, y)

{

}

};

Multiple Inheritance



For multiple reasons should not be used.

class student class mechanical

class student_mech : public student, public mechanical {

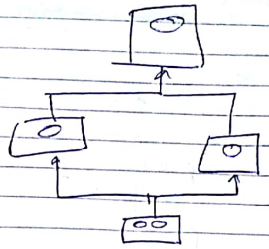
public:

student_mech() : student(), mechanical() {

}

comma

Virtual Base Class



class student {

protected:

int rollno;

public:

};

class studentug : public virtual student {

};

class studentmech : public virtual student {

};

class teamrobocon : public studentug, studentmech {

};

Private inheritance

```
class A {
    int x;
protected:
    int b;
public:
    void abc() { }
};

class B : protected A {
    int a
protected
    int b;
public:
    void xyz() { }
};
```

Pointers

Now funcs cannot be used in main
Thats why this is not very useful.

```
int main() {
    A obja;
    B objb;
    A *ptr;
    ptr = &objb;   ✓
```

This is permitted if B is a child class of A.

```
    B *p;
    p = &obja;   X
```

This is not permitted. as A is parent of B.

parent class pointer with child object.

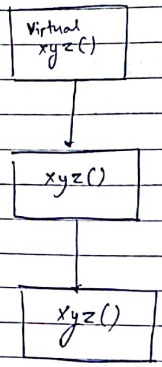
`(*ptr).abc();` ✓ Parent class func can be called.

`(*ptr).xyz();` ✗ Child class function cannot be called.

while `objb.abc();` ✓
`objb.xyz();` ✓

Virtual Functions

```
class A {
public:
    void abc() { }
    virtual void xyz() { }
};

class B : public A {
public:
    void abc() { }
    void xyz() { }
};

int main() {
    A obja;
    B objb;
    objb.abc();   → (2)
    obja.abc();   → (1)
    A *ptr;
    ptr = &objb;
    (*ptr).abc();   → (1)  based on type of pointer
    (*ptr).xyz();   → (2)  based on type of object if func is virtual
}
```



→ Polymorphism in C++

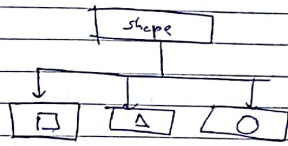
- → Operator overloading
- → Function overloading

```
class shape {
    protected:
        int height;
        int width;
    public:
    virtual void draw(){
        }
};
```

```
class square : public shape {
public:
    void draw() {
    }
};
```

```
class triangle : ——— {
    void draw() {
    }
};
```

```
int main() {
    shape *ptr
    ptr = new square;
    (*ptr).draw();
```



abstract class

→ A class of which a object cannot be created. (pointer is allowed.)

Pure virtual function

→ virtual void draw() = 0;

If draw is not defined in the child classes the child class will became an abstract class.

strings

2 type of strings in C++

1) C string

2) standard string library class

#include <string.h>

string a;
getline (cin, a);

* Array name → pointer constant

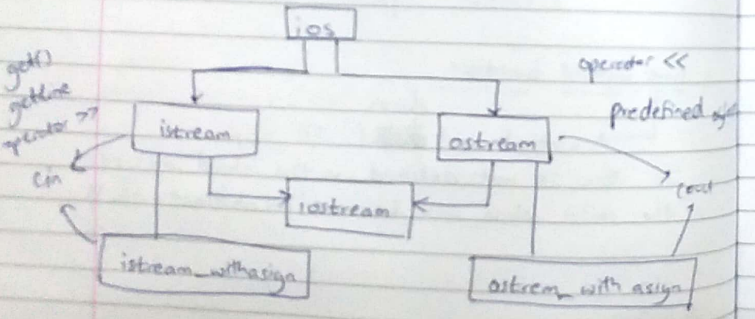
int a[4] = {1, 2, 4, 5}

& → pointer constant

char a[100];
get (a, 100);

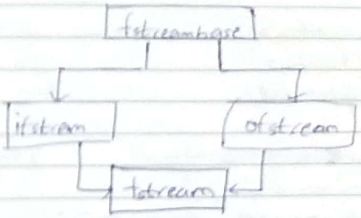
Input/output

objects { cout <<
cin >>



Fstream

#include <fstream>



All this is interconnected with ios.

```
ofstream obj1;          creates a new stream
obj1.open("abc");       Opens or creates a file abc
obj1 << "Hello";        writes "Hello" to abc
obj1.close();           Connection to abc gets closed.
```

Alternate way

```
ofstream obj1("abc");
obj1.open();
obj1.write("---");

ifstream bb("abc")
bb.open();
bb >>
```

User inputs int data
Write it to a file
display what the user had written

```
#       <iostream>
#       <fstream>
using namespace std;
int main(){
    ofstream aa("abc");
    aa.open();
    aa.write();
    aa.close();

    fstream bb("abc");
    bb.open();
    bb.
```

```
int x;
cout << "Enter an int: ";
cin >> x;
ofstream outf("abc");  outf.open();
outf << x;
outf.close();

int y;
istream inf("abc");
inf.open();
inf >> y;
cout << y;
inf.close();
```

Pointer p controls where we input something into the file
↓
put

Pointer g controls from where we read the file.
↓
get

seekp()
seekg()
} will give positions of pointers

seekp(10, ios beg);
↑
10 bytes → 5 ints

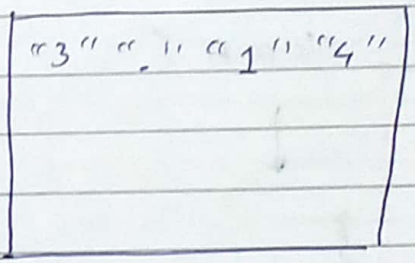
In istream mode
the get pointer gives position of the pointer from where it will start giving out info.

```
seekg (10, ios::beg)
  ↑      ↑
  |      10 bytes from beginning.
  used for manipulating position
```

In ostream mode
put pointer
seekp ( )

tellg() will tell you current position.
tellp()

```
float x = 3.14;
ostream fout("file1");
fout << x;
```




```
float x = 3.142986;
ostream fout ("file1", ios::binary);
fout << x;
```

```
int x[] {1, 3, 6, 9, 11};
ostream fout("file1", ios::binary);
fout.write ((char*) x , 5* sizeof(x))
```

↳ reinterpret_cast<char*>

Write a program to create a class and store contents of the objects in a file.

```
class A {
  int a;
  string name;
public:
  store() {
    ostream fout ("file1", ios::binary | ios::app)
    fout << a << " " << name << "\n";
  }
```

append

Alternate

ios::ate
↳ goes to the end of the file.

```
class A {

    ↓

}

int main() {

  A obja;
  ofstream fout("file1", ios::binary)
  fout.write((char*)(&obja), sizeof(A))
}
```

(1) WAP to read from the file

```
A objb;
ifstream fin ("file1", ios::binary);
fin.read((char*)(&objb), sizeof(A));
objb.show();
}
```