→ Array of objects using pointers :-

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

class shopitem
{
    int id;
    float price;
  public:
    void getdata(int a, float b){
      id=a;
      price=b;
    }
    void getdata(void){
      cout<<"code of this item is"<<id<<endl;
      cout<<"price of this item is"<<price<<endl;
    }
};

int main()
{
    int size=3;

    shopitem *ptr=new shopitem[size];
    shopitem *ptrtemp=ptr;
```

```
int p, i;
float q;

for(i=0;i<size;i++)
{
    cout<<"input id and price"<<endl;
    cin>>p>>q;
    ptr->setdata(p,q);
    ptr++;
}

for(i=0;i<size;i++)
{
    ptrtemp->getdata();
    ptrtemp++;
}
return 0;
}
```

NOTE:- To iterate through the ~~objects~~ array of objects use loops.

→ This keyword :- It is a const pointer.
So lines this = some pointer cause comp. error.
This keyword is a pointer which points to the objects which invokes the given member function.

Since, it has the address of the object we can return it by any function too.

Also it will help in the case that our arguments and class function have same name.

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

class demo
{
    int a;
    public:
        void setdata(int a)
        {
            this->a=a;
        }
};
//Give line of code Put the value of argument(a) in
// class variable 'a', without 'this' it will show
// garbage value.
```

```
void getdata(){
    cout<<" The value of a is "<<a<<endl;
    }
};

int main()
{
    A a;

    a.setdata(5);
    a.getdata();
    return 0;

}
```

Another example to demostrate 'this' keyword

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

class demo{
    int a;
    public:
      demo& getdata(int a);
      {
          this->a=a;
          return *this;
      }
```

```
    void setdata()
    {
        cout<<" value of a is: "<<a<<endl;
    }

};
int main()
{
    demo d;
    d.getdata(10).setdata();

    return 0;
}
```

⇒ Polymorphism :-

Polymorphism means one thing having many forms. We have already done so by function overloading and object overloading, but these are not, only type of polymorphism. There are more type of polymorphism.

Basically polymorphism is of two types.

i) Compile time polymorphism
ii) Run time polymorphism

1) Comple time polymorphism:-

It is also called early biding/ satic binding.

In this type of polymorphism which function will be invoked by which object is binded by compiler at time of compiling.

It is achieved by two types by function overloading and by ~~object overloading/constructor overloading~~ operator overloading.

ii) Run time polymorphism:-

Which function will be invoked by object is decided by compiler at the time of running the program. There is no early biding in this cas.

Run time polymorphism is achieved by virtual functions.

→ Pointers to derived classes :-

We have seen or made pointers which points to base class and by dereferencing them we have invoked their member functions.

What if we make a pointer to base class but assign it the address of derived class. Does it show any error?

No, it will not show any error, we can do this. But when we ~~call~~ dereference that pointer and invoke a function which is in both base and derived class, it will invoke base class function as this pointer is of base class, it doesn't make difference that it has address of derived class.

If we want to invoke member function of derived class through that pointer, we can't do that, as that pointer is of base class type.

It means if we dereference a base class pointer to a member function, it will invoke member function of base class, also if it contains address of derived class as it is a base class pointer.

To reference to derived class member function you have to made pointer of derived class type.

In this process we have also done run time polymorphism if we have one base class and one derived class and both have ~~member~~ one same member function with diffe-
def

```
#include<iostream>
using std::cout;
using std::endl;

class base
{
    int database;
  public:
    void data(int a)
    {
        database=a;
        cout<<" value in database is:"<<
                                database<<endl;
    }
};

class derived : public base
{
    int dataderive;
  public:
    void data(int a)
    {
        dataderive=a;
        cout<<" value in variable is:"<<
                                dataderive<<endl;
    }
};
```

```
int main()
{
    base * ptrb;
    base objb;
    derived objd;
    ptrb = & objd;

    ptrb -> data(10);
    derived * ptrd;
     ptrd = & objd;
      ptrd -> data(95);

    return 0;
}
```

→ Virtual functions :-

When we have a pointer of base class type but having address of derived class ~~if we call/invoke any~~ then we can only invoke member function of base class not member function of derived class.

But if we want to invoke the same function which is in derived class

and also in base class but both have diffe-rent difination, then we have to do only one thing declair that base class function as virtual, then compilar will not invoke that but will invoke second defination of that function which is in derived class.

Also note that if there are more than one function which has different difination in base and derived class, then only that function will be invoked from derived class whose base class difination is declaired as virtual, other functions will be invoked from base class, who are not declaired as virtual, if we pass address of derived class to the pointer of base class and we are using that pointer to point to functions.

If you are inheiting base class as public to derived class then, it implies derived class will have copy of all public member and will have it's own defined functions. So, if we

make a pointer of derived class, then we can access to functions of both class if we want via (::) SRO.

```
#include <iostream>
using std::cout;
using std::endl;

class base{
    int a,b;
  public:
    base(){}
    base(int a1, int b1):a(a1), b(b1){
      cout<<"value in a and b in base
            class is: "<<a<<"and"<<b<<endl;
    }
    virtual void cal(){
      cout<<"value of a+b is:"<<a+b<<endl;
    }
    void print(){
      cout<<"print function of base"
          <<endl;
    }
};
```

```
class derived : public base {
    int a,b;
    public:
    derived (int a1, int b1, int c, int d): a(a1),
                    b(b1), base (c, d)
    {
      Cout<<" value of a and b in derived
            class is: "<<a <<"and" <<b<<endl;
    }

    void cal () {

      Cout<<" value of a*b is: "<<a*b <<endl;
    }
};

int main()
{
   base * ptrb;
   derived objb (54, 51, 764, 12);
   ptrb = &objd;
   ptrb -> cal();
   ptrb -> print();
   cout<<endl <<endl;
   derived * ptr = new derived (564, 65, 87, 8)
   ptr -> base :: cal();
   ptr -> cal();
   p+r -> print();
   return 0;
}
```

NOTE:-

Note that if you have a pointer of base class pointing to derived class (i.e, having address of derived class), and via that pointer you are invoking a function in derived class which is not not base class then it will throw an error that, that function is not in base class.

If you want to invoke a function in derive class, that function should be in base class and should be declaird as virtual. If you want to invoke a function in derived class via pointer of base class and you not want to define that function two time one in base and second in derived, you can simply left blank the function declais in base and then define it in derived as you wish, but that should be virtual.

Points needs to remember while creating virtual functions:-

- They can't be static.

- virtual functions can be friend functions
- A virtual function in base class can be empty.

```
#include<iostream>
#include<cstring>

Class CWH{
  protected:
    string title;
    float rating;
  Public:
    CWH(string s, float r){
      title=s;
      rating r;
    }
    virtual void display(){}
};

Class CWHvideo : Public CWH{
    float videolength;
    Pubic:
      CWHvideo(string s, float r, float vl):
                                CWH(s,r)
      {
        videolength=vl;
      }
```

```
Void display(){
    cout<<"This is amazing video with
        title"<<title<<endl;
    cout<<"Rating:"<<rating<<"out of 5"
    <<endl<<"Length of this video is"
    <<videolength<<"minutes"<<endl;
    }
};

Class CWHText: public CWH{
    int words;
    public:
        CWHText(string s, float r, int Wc): CWH(s,r)
        {
            words=Wc;
        }
        void display(){
        cout<<"This is amazing article with title"
            <<title<<endl<<"Rating"<<Rating
            <<"out of 5"<<endl<<No of
            words it has"<<words<<endl;
        }
};
int main()
{
    string title;
    float rating, vlen;
    int words;
```

```
title = "DJ tutu";
vlen = 4.56;
rating = 4.89;
CWHvideo divideo(title, rating, vlen);

title = "DJ tutu text";
coords = 433;
rating = 4.19;
CWHTEXT ditext(title, rating, coords);

CWH* tuts[2];
tuts[0] = &divideo;
tuts[1] = &ditext;

tuts[0] -> display();
tuts[1] -> display();
return 0;
}
```

→ Abstract base class and pure virtual functions :-

Abstract base class is a special type of class which is ~~often used~~ created to be inherited by other classes. It should must have at least one pure virtual

function.
Also we can't make it's object directly but we can make it's pointer.

Like in privious example class CWH can be abstract base class if virtual function display() changes to pure virtual function.

Pure virtual functions:-

Pure virtual functions are special type of virtual functions, they declared in abstract base class.

If a function is pure virtual function, it means you ~~have to~~ must have to define it in derived classes again.

If you made a pure virtual function in abstract class and you not defined it, again in derived class, compilar will show an error, unlike virtual functions, in case of virtual function compilar will use ~~data~~ and exute

virtual function.

Syntax:-

Virtual function name() = 0;

→ Working with files ←

Till now, in C++ we have written programms in which we have given input to the program and get the out.

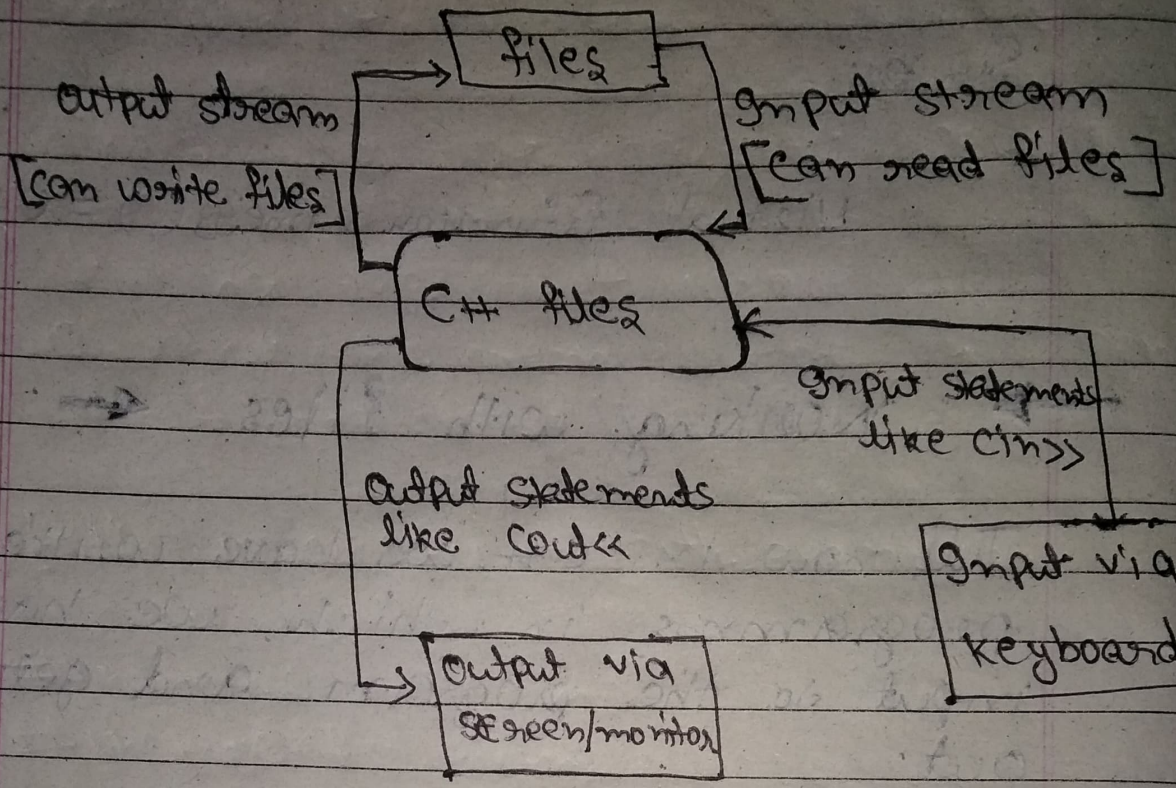
But now, we will learn how to work with files via C++ programms

If bits of information flows to program from files it is called input stream.

And if bits of information flows to files from program it is called output stream.

Via input stream we read these files and via output stream we write

to these files.



There are 3 header files used for I/O in C++

i> iostream → contains objects like cin, cout, cerr, clog etc.

ii> iomanip → I/O manipulators. Contains setw, setprecision etc.

iii> fstream → contains operators for I/O of file stream.

ALSO NOTE THAT <bits/stdc++.h> IS A SPECIAL HEADER FILE, THAT CONTAIN ALL STANDARD LIBRARIES. USING IT IN ANY CONTEST MIGHT BE USEFUL TO SAVE TIME ON WRITING EACH ONE BY ONE.

BUT IT HAS SOME DISADVATAGES. IT IS NOT STANDERD HEADER FILE SO, IT WILL WORK ONLY ON GCC COMPILAR, ALSO USING IT WILL AFFECT YOUR RUN TIME AS IT IS VARY BIG.

To read or write to a file we use a special header file <fstream>.

This headerfile provide 3 useful classes which helps us to work on files.

i) fstreambase :- It is base class as name suggests.

ii) ifstream :- derived from fstreambase, used for input (read)

iii) ofstream :- derived from fstreambase, used for output (write)

In order to work with files, you have to open it.

There are two ways to open a file:

1. using the constructor
2. using the member function open() of the class.

Let's understand it with example.

```
#include<iostream>
#include<fstream>

using namespace std;

int main(){

    string st="Harry bhai";

    ofstream out("Simplebo.txt");

// In this line we have open a file
// Simplebo.txt by making object out
// of class ofstream, so now, we can
// write to this file.
 out<<st;
```

// By this line of code, we have written
// whatever is written in st string to
// that file.

String st2; // a variable of type string.

ifstream in("sample.bob.txt");

//open a file ~~sample.bob.b~~ sample.bob.txt to
// read in to it via object in.

in >> st2; // reading to that file.

//Also note that if you are reading like
//this use will only get first word of that
// file. To read whole line we uses
//getline() function.

getline(in, st2);
cout << st2;

return 0;
}

Also note that getline function read till
encounter enter or sees delimiter
provided by user.
So, it think/ count blank line as a line.

to overcome this problem of getline we can use loop to read until next line with cout not come.

ex. - (for getline)

```
#include<iostream>
#include<cstring>

using namespace std;

int main()
{
    string str;
    int t=4;
    while(t--)
    {
      getline(cin, str);

      while(str.length()==0)
       {
          getline(cin, str);
       }
       cout<< str <<" : newline"<<endl;
    }
    return 0;
}
```