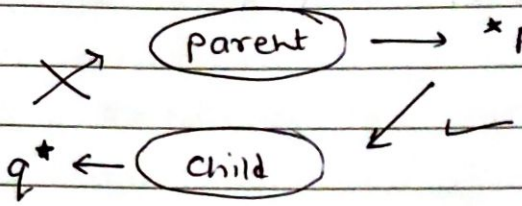
Day 1

DATE

Virtual Functions in C++.

- Base class pointer.
- It can point to the object of any of it's descendant class.
- Converse is not true.



Example.

```
Class A
{
   public :
virtual void f1 () {}
} ;
 Class B : public A
{
   public :
    void f1 () {}                    // function overriding.
    void f2 () {}
} ;
 Void main ()
{
   A *p, O1 ;
   B O2
   p = &O2 ;
   O2. f1 () ;                       // B wala f1
   p->f1 () ;                        // A wala f1   B wala f1
```

classmate

PAGE

→ after adding vi

DATE

when we use pointer of parent class for ~~member~~ object of child class and call member function of child class. We need to use virtual befor parent member function.

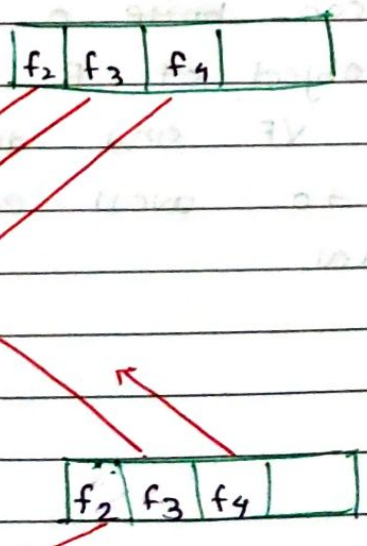
By writing virtual, compiler understands that ~~but~~ early binding of f1 should not happen i.e. binding should happen at run time.

- If any class has atleast one virtual function, an extra variable (* vptr) will be created by compiler. It will be created only in the parent class.

- Also a static array will be created for both parent & child class. It will be called vtable. It contains address of virtual functions present in that class.

Ex:

```
Class A
{ * vptr
public:
  void f1 () {..}
virtual void f2 () {..}
virtual void f3 () {...}
virtual void f4 (){...}
};
Class B : public A
{ void f1 () {...}
  void f2 () {....}
  void f4 (int x) {...}
};
```



classmate

PAGE

DATE

Note : Early binding → consider type of pointer.
Late binding → consider vtable of the class

Abstract Class

- Pure virtual function

A do nothing function is called as pure virtual function

A do nothing function has no body / definition

Syntax : virtual void fun () = 0 ;

Rules :

1) An ~~class~~ object of class containing pure VF can never be made. (Let say A)
2) If you want to use other functions in A. You can ~~write~~ a make a child class (B) and then call object of B. But you must override the pure VF and define it.
3) Also to avoid early binding, make the function virtual

DATE

- A class containing pure virtual function is an abstract class.
- We can not instantiate abstract class. i.e. can not make an object.

Template

- The keyword template is used to define function template and class template.
- It is a way to make your function or class generalize as far as data type is concerned.

1) Function template.

- Function template is also known as generic function.

Syntax:

```
template < Class X >  x Func-name ( X arg-1 )
                        ↳ you can write anything here.
```

Example.

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

DATE

```
int main ()
{
    cout << big(4,5);
    cout << big(4.5, 3.14);
}
```

While early binding X is replaced by appropriate data type.

2) Class template.

- Class template is also known as generic class.

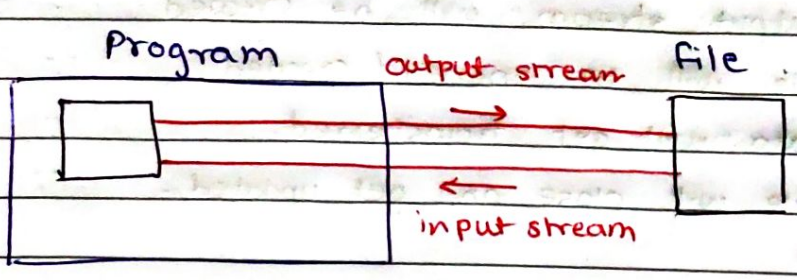
Syntax:

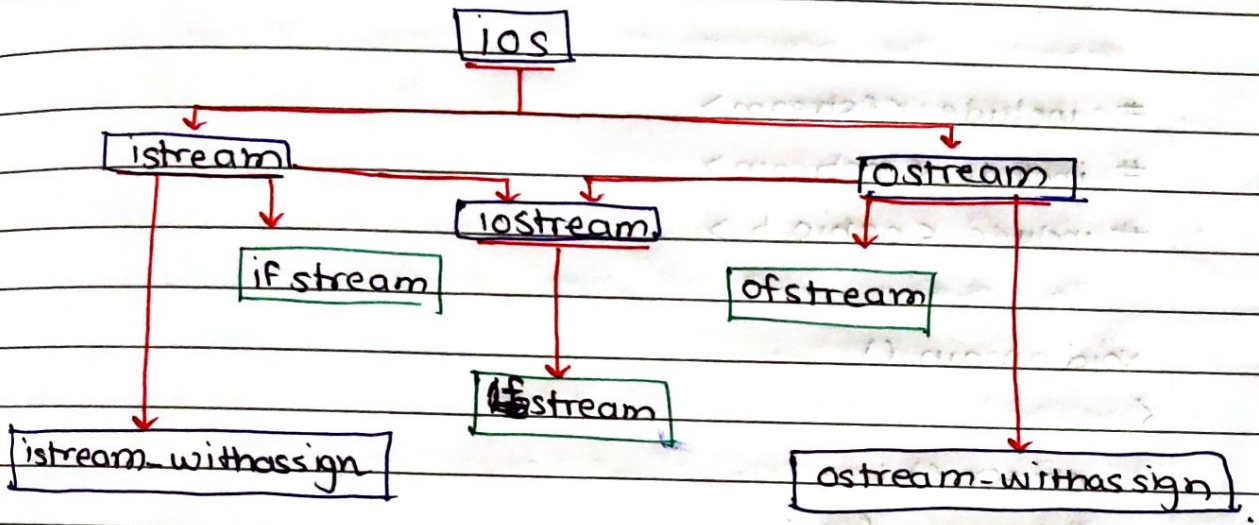
```
template < class X >  class class-name {....};
```

~~Examples~~

DATE

File handling





headerfile : # include <fstream>

Example :

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

class.
object
member function

DATE

Steps :

1) Output stream will be created : ofstream fout
2) File will be opened / created : fout.open.
3) Data will get transferred. : fout << ....
4) File will close and get updated. : fout.close.

Example : inputstream.

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

void main ()
{
  clrscr ();          → class
  fstream fin;        → object
  char ch
  fin.open ("myfile.dat");
  fin >> ch;
  while ( ! fin.eof() )   → Function to find end of file.
  {
    cout << ch
    fin >> ch;
  }
  fin.close ();
  getch ();
}
```

DATE

Steps :

1) ~~out~~ inputstream
2) file will be opened
3) extract data ~~in~~ from file using >>
4) Close the file.

Note : '>>' operator treats space " " as data separator so it will not read ~~—~~ it. To solve this instead of '>>' use get()

eg: fin >> ch; ⟹ ch = fin.get();

- File opening modes.

1)	ios :: in	input / read
2)	ios :: out	output / write
3)	ios :: app	append
4)	ios :: ate	update
5)	ios :: binary	binary.

Syntax:

fin.open ("mytile.dat", ios :: app | ios :: binary)