

Virtual function

Runtime polymorphism – virtual functions (through dynamic binding)

Need for virtual functions :

when objects of different classes in a class hierarchy, reacts to the name ménage in their own unique ways.

```
#include<iostream.h>
#include<string.h>
Class Father
{
    char name [20] ;
    public :
        Father (char * fname)
        {
            strcpy (name, fname)
        }
        void show ()
        {
            cout << "Father name :" << name << endl ;
        }
};
```

```
Class Son : public Father
{ char name [20] ;
  public :
    Son (char * sname, char * fname) : Father (fname)
    {
        strcpy (name, sname) ;
    }
  void show ()
  {
      cout << "son name :" << name << endl ;
  }
};

void main ()
{
  Father * fp ;
  Father f1 ("David") ;
  fp = & f1 ;
  fp→show ()    // display father show () function
  Son s1 ("John", "David") ;
  fp = & s1 ;
  fp→show ()    // guess the output ?
}
```

Output :

Father name : David

Father name : David

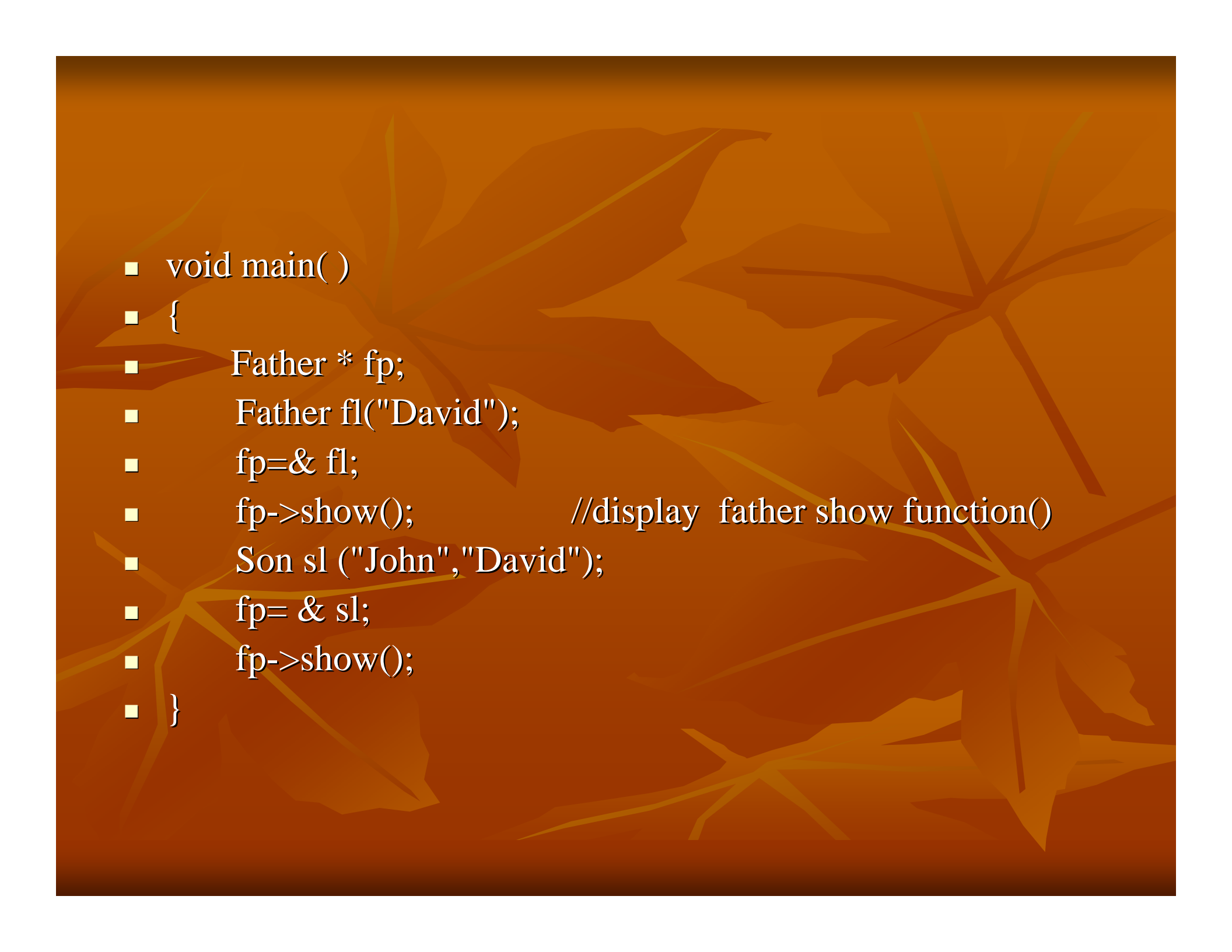
The 2nd stmt fp → show () still invokes the member function show () defined in class Father !

There must be a provision to use the member function show () to display the state of objects of both Father & Son.

In C++ this is achieved by virtual functions.

Virtual functions allow programmers to declare functions in the base class, which can be defined in each derived class.

```
■ #include<iostream.h>
■ #include<string.h>
■ class Father
■ {
■     char name[20];
■     public:
■         Father(char * fname)
■         {
■             strcpy(name,fname);
■         }
■         virtual void show()
■         {cout<<"Father name:"<<name<<endl;
■         }
■     };
■ class Son:public Father
■ {
■     char name[20];
■     public :
■         Son(char * sname,char * fname):Father(fname)
■         {
■             strcpy(name,sname);
■         }
■         void show()
■         {
■             cout<<"Son name:"<<name<<endl;
■         }
■     };
■     };
```



- void main()
- {
- Father * fp;
- Father fl("David");
- fp=& fl;
- fp->show(); //display father show function()
- Son sl ("John","David");
- fp= & sl;
- fp->show();
- }

- Output:
- Father name: David
- Son name: John
- Here we have defined in the class Father virtual void show().
- It indicates that the member function show() is virtual and binding of a call to this function must be postponed until runtime.

- Definition of a virtual function:

- Class Myclass

- {

- public:

- -----

- -----

- Virtual Returntype Functionname(arguments)

- {

- -----

- -----

- }

- };

-

- Virtual function should be defined in the public section of a class.
- Thus c++ provides a solution to invoke the exact version of the member function , which has to be decided at runtime using virtual functions. They are the means by which functions in the base class can be overridden by functions in the derived class.

Rules for virtual functions:

- They can't be static members.
- They can't be friend functions to another class.
- They are accessed using object pointers.
- A class can't have virtual constructors but can contain virtual destructors.
- It is possible to have virtual operator overloading.

Pure virtual functions

Syntax

Virtual Return Type Function Name(arguments)=0

It has null-body \equiv dummy function.

It is expected a desired class will override the pure virtual function.

- ✓ Serves as a framework for future design of the class hierarchy .