# Topics

- Polymorphism
- Virtual Functions

## Polymorphism

- Code is said to be polymorphic if executing the code with different types of data produces different behavior
- Program in the general, rather than program in the specific
- Virtual functions make polymorphism possible

## Consider

```
#include <iostream>
using namespace std;
class Def1
{
  public:
    Def1() {cout << "Def1" << endl;}
    ~Def1 () {cout << "~Def1" << endl;}
    void Foo () {cout << "Def1 Foo" << endl;}
};
class Def2 : public Def1
{
  public:
    Def2 () {cout << "Def2" << endl;}
    ~Def2 () {cout << "~Def2" << endl;}
    void Foo () {cout << "Def2 Foo" << endl;}
};
```

## What is the output? Why?

```
int main ()
{
  Def1 *pcDef1 = new Def1;
  Def2 *pcDef2 = new Def2;
  pcDef1->Foo();
  pcDef2->Foo();
  delete pcDef1;
  delete pcDef2;
}
```

## What is the output? Why?

```
int main ()
{
  Def1 *pcDef1 = new Def1;
  Def1 *pcDef2 = new Def2; // Def2 to Def1
  pcDef1->Foo();
  pcDef2->Foo();
  delete pcDef1;
  delete pcDef2;
}
```

## Virtual Functions

- You can tell the compiler to select the more specialized version of a member function by declaring the member function to be a virtual function
- Declare a virtual function by prefixing its declaration with the word virtual

## What is the output? Why?

- If the following 2 changes are made to the previous program, what is the output? Why?

```
virtual void Foo () {cout << "Def1 Foo" << endl;}

virtual void Foo () {cout << "Def2 Foo" << endl;}

int main ()
{
  Def1 *pcDef1 = new Def1;
  Def1 *pcDef2 = new Def2;
  pcDef1->Foo();
  pcDef2->Foo();
  delete pcDef1;
  delete pcDef2;
}
```

## Virtual Destructor

- Any potential base class should have a virtual destructor
- Why? The compiler performs static binding on any destructor not declared virtual
- If the following changes are made to the original program, what is the output? Why?

```
virtual ~Def1 () {cout << "~Def1" << endl;}

virtual void Foo () {cout << "Def1 Foo" << endl;}

virtual void Foo () {cout << "Def2 Foo" << endl;}

int main ()
{
  Def1 *pcDef1 = new Def1;
  Def1 *pcDef2 = new Def2;
  pcDef1->Foo();
  pcDef2->Foo();
  delete pcDef1;
  delete pcDef2;
}
```

## Base class Person

```
class Person
{
public:
    Person() { setName(""); }
    Person(string pName) { setName(pName); }
    void setName(string pName) { name = pName; }
    string getName() { return name; }
 private:
    string name;
};
```

## Derived class Faculty

```
class Faculty : public Person
{
public:
    Faculty(string fname, Discipline d)
      {setName(fname); setDepartment(d); }
    void setDepartment(Discipline d)
      { department = d; }
    Discipline getDepartment()
      { return department; }
 private:
    Discipline department;
};
```

## Derived class TFaculty

```
class TFaculty : public Faculty
{
public:
    TFaculty(string fname, Discipline d, string title)   :
   Faculty(fname, d)
    {
      setTitle(title);
    }
  void setTitle(string title) { this->title = title; }
   string getName() { return title + " " +
                         Person::getName(); }
 private:
    string title;
};
```

## Polymorphism??

- Is this code polymorphic? If not, how could we make it polymorphic?

```
const int NUM_PEOPLE = 5;

Person *arr[NUM_PEOPLE] = {
   new Tfaculty(“Indiana Jones”, ARCHEOLOGY, “Dr.”),
   new Person(“Thomas Cruise”),
   new Faculty(“James Stock”, BIOLOGY),
   new Tfaculty(“Sharon Rock”, BIOLOGY, “Professor”),
   new TFaculty(“Nicole Eweman”, ARCHEOLOGY, “Dr,”)};

for(int k = 0; k < NUM_PEOPLE; k++)

{

  cout << arr[k]->getName() << endl;

}
```