2 Virtual Destructors

2.1 Destructors

- A **destructor** is a member function that is called automatically when an object of the class goes out of scope
  - The destructor contains code to delete all dynamic variables created by the object
  - A class has only one destructor with no arguments
  - The name of the destructor is distinguished from the default constructor by the tilde symbol ~
    - Example: ~CyclicArray();

2.1.1 How constructor and destructor called? sample55.cpp

```
class pets{
public:
    // constructor
    pets(){
        cout << "constructor of pets" << endl;
    }
    // destructor
    ~pets(){
        cout << "destructor of pets" << endl;
    }
    // member functions
    void give_name(string name){
        pet_name = name;
    }
    void show_name(){
        cout << "pet name is " << pet_name << endl;
    }
private:
    string pet_name;
};
```

```
class dog: public pets{
public:
  // constructor
  dog(){
    cout << "constructor of dog" << endl;
  };
  // destructor
  ~dog(){
    cout << "destructor of dog" << endl;
  }
  void show_name(){
    cout << "dog name is " << pet_name << endl;
  }
};
```

```
int main(int argc, const char * argv[])
{
    pets mypet;
    mypet.give_name("GOO");

    mypet.show_name();

    dog mydog;
    mydog.give_name("POO");

    mydog.show_name();
    return 0;
}
```

Run results

```
constructor of pets
pet name is GOO
constructor of pets
constructor of dog
dog name is POO
destructor of dog
destructor of pets
destructor of pets
```

- At **“pets mypet;”,** the default constructor of **Pet** is called
- **show_name()** of **Pet** class is used.
- At **“dog mydog;”**, the default constructor of **Pet** is called then the default constructor of **Dog** is called
- **show_name()** of **Dog** class is used.
- At the end of program, when **mydog** is deleted, the destructor of **Dog** is called then the destructor of **Pet** is called
- At the end of program, when **mypet** is deleted, the destructor of **Pet** is called

2.1.2 Objects as a base class array

```
int main(int argc, const char * argv[])
{
    pets* mypet[3];

    mypet[0] = new dog;
    mypet[0]->give_name("GOO");

    mypet[1] = new cat;
    mypet[1]->give_name("FOO");

    mypet[2] = new dog;
    mypet[2]->give_name("BOO");

    for (int i = 0 ; i < 3 ; i++){
        cout << i << " ";
        mypet[i]->show_name();
    }
    // delete
    for (int i = 0 ; i < 3 ; i++){
        delete mypet[i];
    }
    return 0;
}
```

Run results

```
constructor of pets
constructor of dog
constructor of pets
constructor of cat
constructor of pets
constructor of dog
0 pet name is GOO
1 pet name is FOO
2 pet name is BOO
destructor of pets
destructor of pets
destructor of pets
```

- All objects calls **show_name()** of **Pet** class (base class)
  - We have to make **show_name()** of **Pet** class **virtual**
- All object calls **Pet** class destructor.
  - See below

2.2 Destructors should be made virtual

- Consider
  ```
  Base *pBase = new Derived;

  delete pBase;
  ```
  - If the destructor in **Base** is virtual, the destructor for **Derived** is invoked as **pBase** points to a **Derived** object, returning **Derived** members to the free store
    * The **Derived** destructor in turn calls the **Base** destructor

2.2.1 Non-Virtual Destructors

- If the Base destructor is not virtual, only the Base destructor is invoked
- This leaves Derived members, not part of Base, in memory

2.2.2 Modify Base Class : sample56.cpp

```
class pets{
public:
    // constructor
    pets(){
        cout << "constructor of pets" << endl;
    }
    // destructor
    virtual ~pets(){
        cout << "destructor of pets" << endl;
    }

    // member functions
    void give_name(string name){
        pet_name = name;
    }

    virtual void show_name(){
        cout << "pet name is " << pet_name << endl;
    }
protected:
    string pet_name;
};
```

Run results

```
constructor of pets
constructor of cat
constructor of pets
constructor of dog
0 dog name is GOO
1 cat name is FOO
2 dog name is BOO
destructor of dog
destructor of pets
destructor of cat
destructor of pets
destructor of dog
destructor of pets
```

- Each object calls its own show_name()
- Each object calls its own destructor first, then calls base class destructor.