Object Oriented Programming

Copy Constructors & Pointers

Mr. Usman Wajid

usman.wajid@nu.edu.pk

March 23, 2023

National University
of Computer & Emerging Sciences

Copy Constructor

Copy Constructor

It is special type of constructor intended to copy one object (source) into another (target) of the same `class`

Copy Constructor

Copy Constructor

It is special type of constructor intended to copy one object (source) into another (target) of the same `class`

- When no copy constructor is defined, then a default **copy constructor** is **implicitly invoked** by the compiler. All data and function members will be copied.

Copy Constructor

Copy Constructor

It is special type of constructor intended to copy one object (source) into another (target) of the same `class`

- When no copy constructor is defined, then a default **copy constructor** is **implicitly invoked** by the compiler. All data and function members will be copied.
- **Basic Syntax 1:**

```
<class-name> <target-object>(<source-object>);
```

Copy Constructor

Copy Constructor

It is special type of constructor intended to copy one object (source) into another (target) of the same `class`

- When no copy constructor is defined, then a default **copy constructor** is **implicitly invoked** by the compiler. All data and function members will be copied.
- **Basic Syntax 1:**

```
<class-name> <target-object>(<source-object>);
```

- **Basic Syntax 2:**

```
<class-name> <target-object> = <source-object>;
```

Copy Constructor Example 1

```
class Student {
        private:
                char section;
        public:
                void setSection(char section){
                        this->section = section;
                }
                char getSection(){
                        return section;
                }
};
int main() {
        Student ali;
        ali.setSection('A');
        cout<<"ali sec: "<<ali.getSection()<<endl;

        Student mahad(ali);
        cout<<"mahad sec: "<<mahad.getSection()<<endl;

        Student zain = ali;
        cout<<"zain sec: "<<zain.getSection()<<endl;
}
```

Output

Copy Constructor Example 1

```
class Student {
        private:
                char section;
        public:
                void setSection(char section){
                        this->section = section;
                }
                char getSection(){
                        return section;
                }
};
int main() {
        Student ali;
        ali.setSection('A');
        cout<<"ali sec: "<<ali.getSection()<<endl;

        Student mahad(ali);
        cout<<"mahad sec: "<<mahad.getSection()<<endl;

        Student zain = ali;
        cout<<"zain sec: "<<zain.getSection()<<endl;
}
```

Output

```
ali sec: A
mahad sec: A
zain sec: A
```

Copy Constructor Example 2

```
class Student {
        private:
                char sec;
        public:
                Student(){
                        cout<<"Default constructor\n";
                }
                Student(Student &obj){
                        setSec(obj.sec);
                        cout<<"copy constructor\n";
                }
                void setSec(char sec){
                        this->sec = sec;
                }
                char getSec(){
                        return sec;
                }
};
```

```
int main() {
Student ali;
ali.setSec('A');
cout<<"ali sec: "<<ali.getSec()<<"\n\n";

Student mahad(ali);
cout<<"mahad sec: "<<mahad.getSec()<<"\n\n";

Student zain = ali;
cout<<"zain sec: "<<zain.getSec()<<"\n\n";

Student akhtar;
akhtar = ali;
cout<<"akhtar sec: "<<zain.getSec()<<endl;
}
```

Copy Constructor Example 2

```
class Student {
        private:
                char sec;
        public:
                Student(){
                        cout<<"Default constructor\n";
                }
                Student(Student &obj){
                        setSec(obj.sec);
                        cout<<"copy constructor\n";
                }
                void setSec(char sec){
                        this->sec = sec;
                }
                char getSec(){
                        return sec;
                }
};
```

```
int main() {
Student ali;
ali.setSec('A');
cout<<"ali sec: "<<ali.getSec()<<"\n\n";

Student mahad(ali);
cout<<"mahad sec: "<<mahad.getSec()<<"\n\n";

Student zain = ali;
cout<<"zain sec: "<<zain.getSec()<<"\n\n";

Student akhtar;
akhtar = ali;
cout<<"akhtar sec: "<<zain.getSec()<<endl;
}
```

```
Default constructor
ali sec: A

copy constructor
mahad sec: A

copy constructor
zain sec: A

Default constructor
akhtar sec: A
```

Copy Constructor continued …

- A default copy constructor is implicitly called in the following scenarios,

Copy Constructor continued ...

- A default copy constructor is implicitly called in the following scenarios,
  1. When an object is declared and initialized by using the value of another object

Copy Constructor continued …

- A default copy constructor is implicitly called in the following scenarios,
  1. When an object is declared and initialized by using the value of another object
  2. When an object is passed by value to a function

Copy Constructor continued …

- A default copy constructor is implicitly called in the following scenarios,
  1. When an object is declared and initialized by using the value of another object
  2. When an object is passed by value to a function
  3. When the return value of a function is an object

Copy Constructor Example 3

```
class Student {
        private:
                char sec;
        public:
                Student(){
                        cout<<"Default constructor\n";
                }
                void setSec(char sec){
                        this->sec = sec;
                }
                char getSec(){
                        return sec;
                }
};
Student fun(Student obj){
        Student mahad(obj);
        cout<<"mahad sec: "<<mahad.getSec()<<"\n\n";
        return mahad;
}
```

```
int main() {
        Student ali;
        ali.setSec('A');
        cout<<"ali sec: "<<ali.getSec()<<"\n\n";

        Student imran(ali);
        cout<<"imran sec: "<<imran.getSec()<<"\n\n";

        Student zain = fun(ali);
        cout<<"zain sec: "<<zain.getSec()<<"\n\n";

        Student akhtar;
        akhtar = ali;
        cout<<"akhtar sec: "<<zain.getSec()<<endl;
}
```

Copy Constructor Example 3

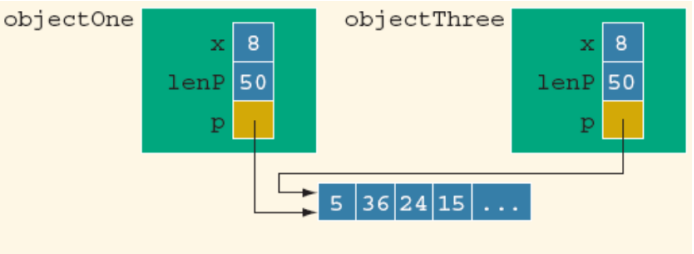
```
class Student {
        private:
                char sec;
        public:
                Student(){
                        cout<<"Default constructor\n";
                }
                void setSec(char sec){
                        this->sec = sec;
                }
                char getSec(){
                        return sec;
                }
};
Student fun(Student obj){
        Student mahad(obj);
        cout<<"mahad sec: "<<mahad.getSec()<<"\n\n";
        return mahad;
}
```

```
int main() {
        Student ali;
        ali.setSec('A');
        cout<<"ali sec: "<<ali.getSec()<<"\n\n";

        Student imran(ali);
        cout<<"imran sec: "<<imran.getSec()<<"\n\n";

        Student zain = fun(ali);
        cout<<"zain sec: "<<zain.getSec()<<"\n\n";

        Student akhtar;
        akhtar = ali;
        cout<<"akhtar sec: "<<zain.getSec()<<endl;
}
```

```
ali sec: A

imran sec: A

mahad sec: A

zain sec: A

Default constructor
akhtar sec: A
```

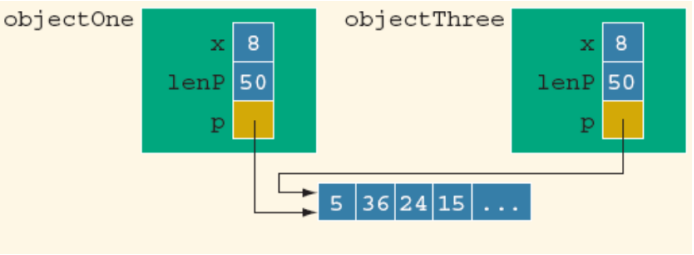
Copy Constructor with pointer member

```
pointerDataClass objectThree(objectOne);
```



Copy Constructor with pointer member

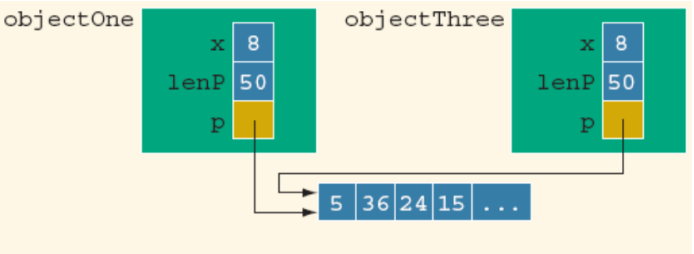
```
pointerDataClass objectThree(objectOne);
```



- This initialization is called the default member-wise initialization

Copy Constructor with pointer member

```
pointerDataClass objectThree(objectOne);
```



- This initialization is called the default member-wise initialization
- It implicitly invokes the default copy constructor

Copy Constructor Example 4

```
class List {
        private:
                int * data;
                int size;
        public:
                List (int size){
                        this->size = size;
                        data = new int[size];
                }
                void fill(){
                        for(int i=0; i<size; i++)
                                data[i] = i*i;
                }
                void update(int index, int val){
                        data[index] = val;

                }
                void print(){
                        for (int i=0; i<size; i++)
                                cout<<data[i]<<" ";
                }
                ~List(){
                        delete [] data;
                }
};
```

```
int main() {

        List objectOne(5);
        objectOne.fill();
        cout<<"obj1 : ";
        objectOne.print();

        List objectThree(objectOne);
        objectThree.update(0,3);


        cout<<"\n\nobj1 : ";
        objectOne.print();
}
```

Copy Constructor Example 4

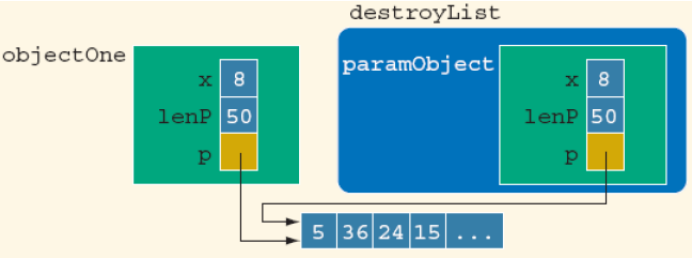
```
class List {
        private:
                int * data;
                int size;
        public:
                List (int size){
                        this->size = size;
                        data = new int[size];
                }
                void fill(){
                        for(int i=0; i<size; i++)
                                data[i] = i*i;
                }
                void update(int index, int val){
                        data[index] = val;

                }
                void print(){
                        for (int i=0; i<size; i++)
                                cout<<data[i]<<" ";
                }
                ~List(){
                        delete [] data;
                }
};
```

```
int main() {

        List objectOne(5);
        objectOne.fill();
        cout<<"obj1 : ";
        objectOne.print();

        List objectThree(objectOne);
        objectThree.update(0,3);


        cout<<"\n\nobj1 : ";
        objectOne.print();
}
```

```
obj1 : 0 1 4 9 16

obj1 : 3 1 4 9 16
```

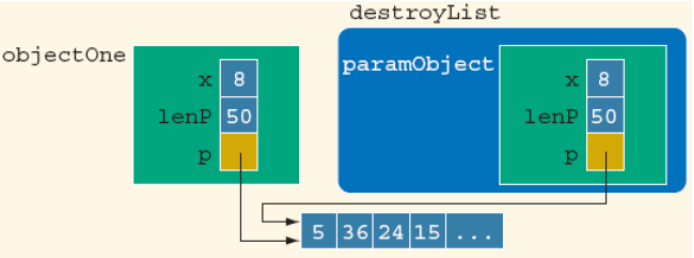
Copy Constructor with pointer member continued ...

```
void destroyList(pointerDataClass paramObject);
```



Copy Constructor with pointer member continued ...

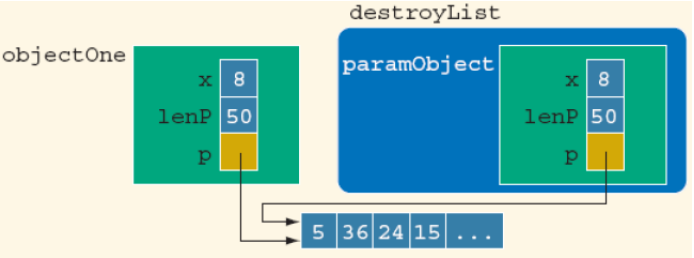
```
void destroyList(pointerDataClass paramObject);
```



- Default Initialization leads to shallow copying of data

Copy Constructor with pointer member continued ...

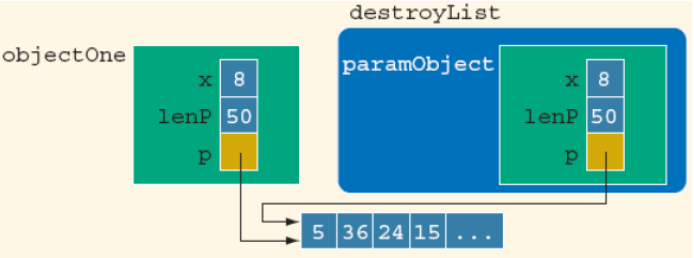
```
void destroyList(pointerDataClass paramObject);
```



- Default Initialization leads to shallow copying of data
- Similar problem occurs when passing objects by value

Copy Constructor with pointer member continued ...

```
void destroyList(pointerDataClass paramObject);
```



- Default Initialization leads to shallow copying of data
- Similar problem occurs when passing objects by value
- **Solution:** Write a user defined copy constructor to create a deep copy

Copy Constructor Example 5

```
class List {
        private:
        int * data;
        int size;
        public:
        List (int size){
                this->size = size;
                data = new int[size];
        }
        void fill(){
                for(int i=0; i<size; i++)
                data[i] = i*i;
        }
        void print(){
                for (int i=0; i<size; i++)
                cout<<data[i]<<" ";
        }
        ~List(){
                delete [] data;
        }
};

void destroyList (List ObjectThree){}
```

```
int main() {

        List objectOne(5);
        objectOne.fill();
        cout<<"obj1 : ";
        objectOne.print();

        destroyList(objectOne);

        cout<<"\n\nobj1 : ";
        objectOne.print();
}
```

Copy Constructor Example 5

```
class List {
        private:
        int * data;
        int size;
        public:
        List (int size){
                this->size = size;
                data = new int[size];
        }
        void fill(){
                for(int i=0; i<size; i++)
                data[i] = i*i;
        }
        void print(){
                for (int i=0; i<size; i++)
                cout<<data[i]<<" ";
        }
        ~List(){
                delete [] data;
        }
};

void destroyList (List ObjectThree){}
```

```
int main() {

        List objectOne(5);
        objectOne.fill();
        cout<<"obj1 : ";
        objectOne.print();

        destroyList(objectOne);

        cout<<"\n\nobj1 : ";
        objectOne.print();
}
```

```
obj1 : 0 1 4 9 16

obj1 : 7958456 7930048 4 9 16
```

Copy Constructor Example 6

```
class List {
        private:
        int * data;
        int size;
        public:
        List (int size){
                this->size = size;
                data = new int[size];
        }
        List(List &obj){
                size = obj.size;
                data = new int[size];
                for (int i=0; i<size; i++)
                        data[i] = obj.data[i];
        }
        void fill(){
                for(int i=0; i<size; i++)
                data[i] = i*i;
        }
        void print(){
                for (int i=0; i<size; i++)
                cout<<data[i]<<" ";
        }
        ~List(){
                delete [] data;
        }
};

void destroyList (List ObjectThree){}
```

```
int main() {

        List objectOne(5);
        objectOne.fill();
        cout<<"obj1 : ";
        objectOne.print();

        destroyList(objectOne);

        cout<<"\n\nobj1 : ";
        objectOne.print();
}
```

Copy Constructor Example 6

```
class List {
        private:
        int * data;
        int size;
        public:
        List (int size){
                this->size = size;
                data = new int[size];
        }
        List(List &obj){
                size = obj.size;
                data = new int[size];
                for (int i=0; i<size; i++)
                        data[i] = obj.data[i];
        }
        void fill(){
                for(int i=0; i<size; i++)
                data[i] = i*i;
        }
        void print(){
                for (int i=0; i<size; i++)
                cout<<data[i]<<" ";
        }
        ~List(){
                delete [] data;
        }
};

void destroyList (List ObjectThree){}
```

```
int main() {

        List objectOne(5);
        objectOne.fill();
        cout<<"obj1 : ";
        objectOne.print();

        destroyList(objectOne);

        cout<<"\n\nobj1 : ";
        objectOne.print();
}
```

```
obj1 : 0 1 4 9 16

obj1 : 0 1 4 9 16
```