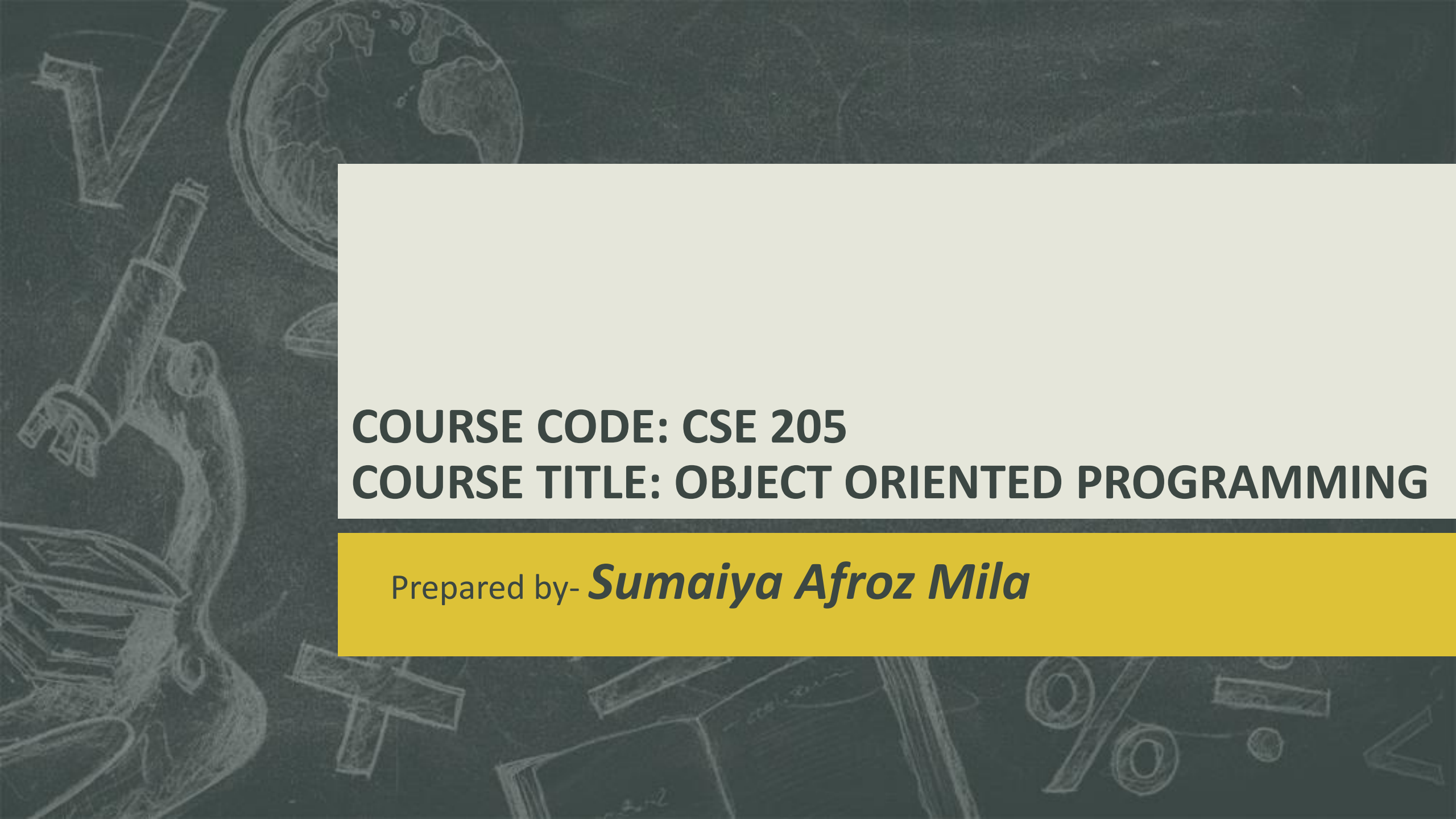


COURSE CODE: CSE 205
COURSE TITLE: OBJECT ORIENTED PROGRAMMING

Prepared by- ***Sumaiya Afroz Mila***

Constructor

- A ***constructor*** is a special member function which initializes the objects
  - Has the exact same name as its class
  - No return type
  - It has to be public
- Invoked implicitly/explicitly every time an instance/object of that class is created
- Can be overloaded
- Are called in the order of creation
- Involve memory allocation
- Once a constructor is defined, the default one stops working

Type of Constructors

- 3 types of constructors
  - Default / Zero argument – A constructor that accepts no parameter is called default constructor. Initializes the members of all the objects to a predefined value.
  - Parameterized – A constructor that accepts parameters is called parameterized constructor. Used when it is necessary to initialize different values to the members of different objects.
  - Copy Constructor

Default Constructor

```
class student {
        int id;
public:
        void setValues(){
        id = 0;
        }
        int getId(){
        return id;
        }
};
int main(){
student st;
st. setValues();
cout<<st.getId();
}
```

```
class student {
        int id;
public:
        student() {
        id=0;
        }
        int getId() {
        return id;
        }
};
int main(){
student st;
cout<<st.getId();
}
```

Constructor

Destructor

- A ***destructor*** is a special member function which destroys the objects
  - Has the exact same name as its class, preceded by the sign '~'
  - No return type
  - It has to be public
- Can not be overloaded
- Called when an object is destroyed
  - A function ends
  - Global objects destroyed when the program ends
  - a delete operator is called
  - A block containing local variable ends
- Are called in the reverse order of creation
- Cleans memory or deallocates memory

Why Defining Destructor is Necessary?

- If user defined ***destructor*** is not written in class, compiler calls the ***default destructor.*** Default destructor deallocates memory unless we have dynamically allocated memory. When a class contains a pointer to allocate memory dynamically, we should write a destructor to release memory before the class instance/object is destroyed.
- This has to be done to avoid memory leak.

Constructor and Destructor

```
class student {
        int id;
public:
        void setValues(){
        id = 0;
        }
        int getId(){
        return id;
        }
};
int main(){
student st;
st. setValues();
cout<<st.getId();
}
```

```
class student {
        int id;
public:
        student() {
        id=0;
        }
        ~student() {
        }
        int getId() {
        return id; }
};
int main(){
student st;
cout<<st.getId();
}
```

Constructor

Destructor

Default/ Zero Argument Constructor

```
class student {
    int id;
public:
    student() {
    id = 0;
    cout << "Constructing "<<id<<endl;
    }
    ~student(){
    cout << "Destructing "<<id<<endl;
    }
    int getId() {return id;}
};

int main(){
student st1,st2;
cout<<st1.getId()<<endl;
cout<<st2.getId()<<endl;
}
```

Constructor with zero parameter

```
"C:\Users\ASUS\Desktop\cse 205\Untitled1.exe"
Constructing  0
Constructing  0
0
0
Destroying  0
Destroying  0

Process returned 0 (0x0)   execution time : 0.088 s
Press any key to continue.
```

Object declaration should match with constructor

Parameterized Constructors

- Parameterized Constructor – When a constructor has been parameterized, passing the initial values as arguments to the constructor can be done in two ways while declaring the object:
  - By calling the constructor explicitly – student object = student(2);
  - By calling the constructor implicitly – student object(2);
- When a constructor is parameterized, appropriate arguments should be sent for the constructor.

Parameterized Constructors

```
class student {
    int id;
public:
    student(int i) {
    id = i;
    cout << "Constructing "<<id<<endl;
    }
    ~student(){
    cout << "Destructing "<<id<<endl;
    }
    int getId() {return id;}
};

int main(){
student st1(1),st2(2);
cout<<st1.getId()<<endl;
cout<<st2.getId()<<endl;
}
```

```
"C:\Users\MP & MC LAB\Desktop\Untitled1.exe"
Constructing 1
Constructing 2
1
2
Destructing 2
Destructing 1

                returned 0 (0x0)   execution time : 0.173 s
```

Parameterized constructor

Object declaration should match the constructor

Parameterized Constructors

```
class student {
    int id;
public:
    student(int i) {
    id = i;
    cout << "Constructing "<<id<<endl;
    }
    ~student(){
    cout << "Destructing "<<id<<endl;
    }
    int getId() {return id;}
};

int main(){
    student st(1);
    student st2 = student(2);
    cout<<st.getId()<<endl;
    cout<<st2.getId()<<endl;
    }
```

Parameterized constructor

Both the declaration is allowed

Parameterized Constructors

```
class student {
    int id;
public:
    student(int i) {
    id = i;
    cout << "Constructing "<<id<<endl;
    }
    ~student(){
    cout << "Destructing "<<id<<endl;
    }
    int getId() {return id;}
};

int main(){
    student st(1),st2;
    cout<<st.getId()<<endl;
    cout<<st2.getId()<<endl;
    }
```

What will happen now?

```
16  int main(){
17      student st(1),st2;
```

Blocks | Search results | Cccc | Build log | Build messages | CppCheck

	Line	Message
\SUS\...	6	note: candidate expects 1 argument, 0 provided
\SUS\...	3	note: candidate: constexpr student::student(const student&)
\SUS\...	3	note: candidate expects 1 argument, 0 provided

=== Build failed: 1 error(s), 0 warning(s) (0 minute(s), 0 second

Solution : Multiple Constructor in a Class/ Constructor Overloading

- Once a constructor is defined, the default one stops working.
- While defining constructor, we should take care of the fact that, there must be a constructor function for each way that an object of a class will be created.
- If a program attempts to create an object for which no matching constructor is found, compile-time error occurs.
- In the previous example, there was compilation error, because the declaration "student st2" did not match any constructor call. The only constructor function in the class accepts one parameter. But the declaration provides no argument.
- To avoid the error in the previous example, we could define 2 constructors.
  - **Default constructor** – this would match the declaration of student st2
  - **Parameterized constructor** – this would match the declaration of student st1(1)

Multiple Constructors

```
class student {
    int id;
public:
    student() {
    id = 0;
    cout << "Constructing "<<id<<endl;
    }
    student(int i) {
    id = i;
    cout << "Constructing "<<id<<endl;
    }
    ~student(){
    cout << "Destructing "<<id<<endl;
    }
    int getId() {return id;}
};
```

```
int main(){
    student st(1),st2;
    cout<<st.getId()<<endl;
    cout<<st2.getId()<<endl;
    }
```

Necessity of Constructor Overloading

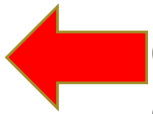
- To gain flexibility
- To support arrays
- To create copy constructors

Constructor-Destructor Call

```
class student {
    int id;
    char* name;
public:
    student(char* p , int q) {
    id=q;
    name = new char[strlen(p)];
    strcpy(name,p);
    cout << "Constructing: "<<name<<endl;
    }
    ~student(){
    cout << "Destructing "<<name<<endl;
    delete [] name;
    }
    int getId() {return id;}
};
```

```
void func(student st) {
cout << st.getId()<<endl;
}

int main(){
student st1("St1",1);
func(st1);
cout<<"Finish"<<endl;
}
```



Object declaration

OUTPUT??

Constructing st1

Constructor-Destructor Call

```
class student {
    int id;
    char* name;
public:
    student(char* p , int q) {
    id=q;
    name = new char[strlen(p)];
    strcpy(name,p);
    cout << "Constructing: "<<name<<endl;
    }
    ~student(){
    cout << "Destructing "<<name<<endl;
    delete [] name;
    }
    int getId() {return id;}
};
```

```
void func(student st) {
cout << st.getId()<<endl;
}
int main(){
student st1("St1",1);
func(st1);
cout<<"Finish"<<endl;
}
```

- This statement is not object declaration.
- This is an assignment stmt student st = st1;

OUTPUT??

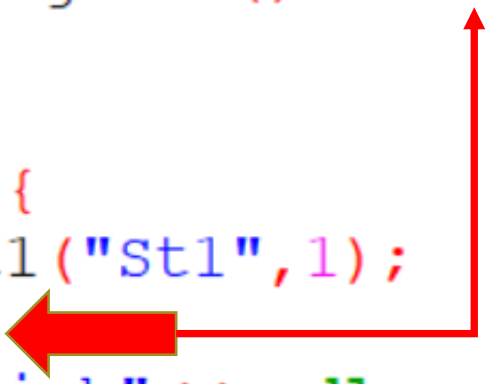
Constructing st1

Constructor-Destructor Call

```
class student {
    int id;
    char* name;
public:
    student(char* p , int q) {
    id=q;
    name = new char[strlen(p)];
    strcpy(name,p);
    cout << "Constructing: "<<name<<endl;
    }
    ~student(){
    cout << "Destructing "<<name<<endl;
    delete [] name;
    }
    int getId() {return id;}
};
```

```
void func(student st) {
cout << st.getId()<<endl;
}

int main(){
student st1("St1",1);
func(st1);
cout<<"Finish"<<endl;
}
```



OUTPUT??

Constructing st1
1

Constructor-Destructor Call

```
class student {
    int id;
    char* name;
public:
    student(char* p , int q) {
    id=q;
    name = new char[strlen(p)];
    strcpy(name,p);
    cout << "Constructing: "<<name<<endl;
    }
    ~student(){
    cout << "Destructing "<<name<<endl;
    delete [] name;
    }
    int getId() {return id;}
};
```

```
void func(student st) {
cout << st.getId()<<endl;
}

int main(){
student st1(
func(st1);
cout<<"Finish"<<endl;
}
```

- func() ends here
- The object "st" is a local variable of the function.
- So, destructor for object "st" will be called

OUTPUT??

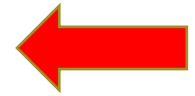
Constructing st1
1
Destructing st1

Constructor-Destructor Call

```
class student {
    int id;
    char* name;
public:
    student(char* p , int q) {
    id=q;
    name = new char[strlen(p)];
    strcpy(name,p);
    cout << "Constructing: "<<name<<endl;
    }
    ~student(){
    cout << "Destructing "<<name<<endl;
    delete [] name;
    }
    int getId() {return id;}
};
```

```
void func(student st) {
cout << st.getId()<<endl;
}

int main(){
student st1("St1",1);
func(st1);
cout<<"Finish"<<endl;
}
```



OUTPUT??

Constructing st1
1
Destructing st1
Finish

Constructor-Destructor Call

```
class student {
    int id;
    char* name;
public:
    student(char* p , int q) {
    id=q;
    name = new char[strlen(p)];
    strcpy(name,p);
    cout << "Constructing: "<<name<<endl;
    }
    ~student(){
    cout << "Destructing "<<name<<endl;
    delete [] name;
    }
    int getId() {return id;}
};
```

```
void func(student st) {
cout << st.getId()<<endl;
}

int main(){
student st1("St1",1);
func(st1);
cout<<"Finish"<<endl;
}
```

- Int main() ends here
- The object "st1" is a local variable of the function.
- So, destructor for object "st1" will be called

OUTPUT??

Constructing st1
1
Destructing st1
Finish
Destructing st1

Constructor-Destructor Call

```
class student {
    int id;
    char* name;
public:
    student(char* p , int q) {
    id=q;
    name = new char[strlen(p)];
    strcpy(name,p);
    cout << "Constructing: "<<name<<endl;
    }
    ~student(){
    cout << "Destructing "<<name<<endl;
    delete [] name;
    }
    int getId() {return id;}
};
```

```
void func(student st) {
cout << st.getId()<<endl;
}

int main(){
student st1("St1",1);
func(st1);
cout<<"Finish"<<endl;
```

- Names of the objects character pointer
- While calling the function ***func(st1)***, value of st1 is stored in st(a bitwise copy made)
- ***st.name = st1.name & st.id = st1.id***
- Both the objects names are pointing to the same memory. ***st.name & st1.name*** are pointing to the same memory location
- When the destructor is called 1st time, it destroys the memory containing the name

OUTPUT??

Constructing st1
1
Destructing st1
Finish
Destructing st1

Constructor-Destructor Call

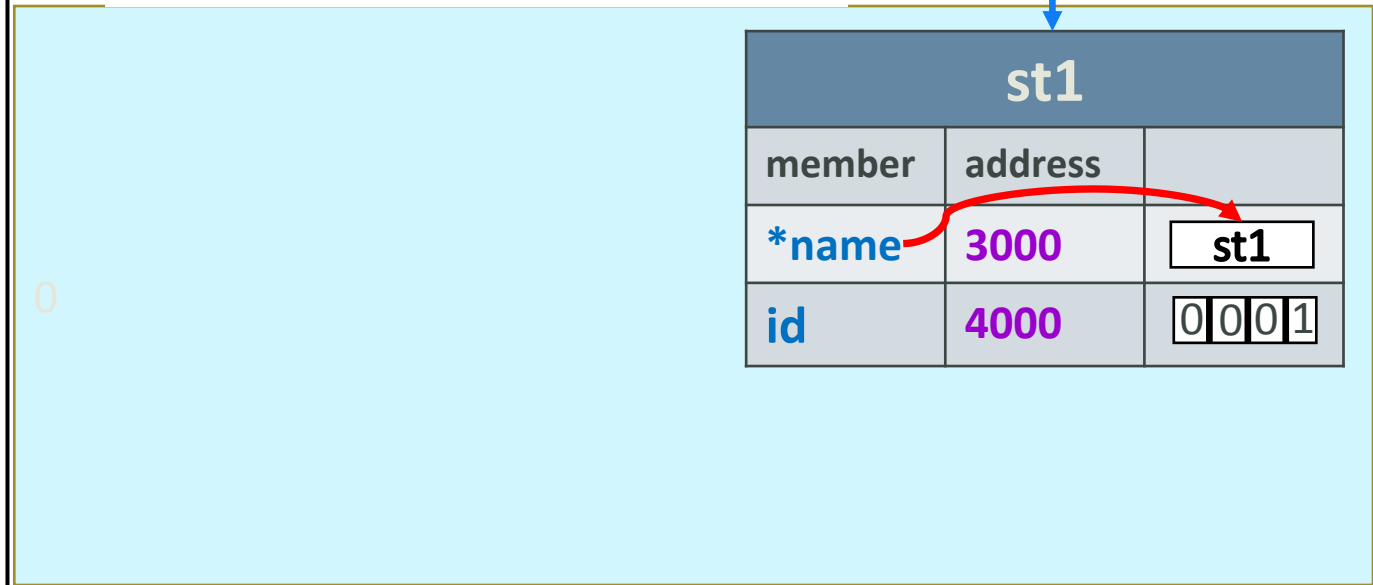
```
class student {
    int id;
    char* name;
public:
    student(char* p , int q) {
    id=q;
    name = new char[strlen(p)];
    strcpy(name,p);
    cout << "Constructing: "<<name<<endl;
    }
    ~student(){
    cout << "Destructing "<<name<<endl;
    delete [] name;
    }
    int getId() {return id;}
};
```

```
void func(student st) {
cout << st.getId()<<endl;
}

int main(){
student st1("St1",1);
func(st1);
cout<<"Finish"<<endl;
}
```

OUTPUT??

Constructing st1



Constructor-Destructor Call

```
class student {
    int id;
    char* name;
public:
    student(char* p , int q) {
    id=q;
    name = new char[strlen(p)];
    strcpy(name,p);
    cout << "Constructing: "<<name<<endl;
    }
    ~student(){
    cout << "Destructing "<<name<<endl;
    delete [] name;
    }
    int getId() {return id;}
};
```

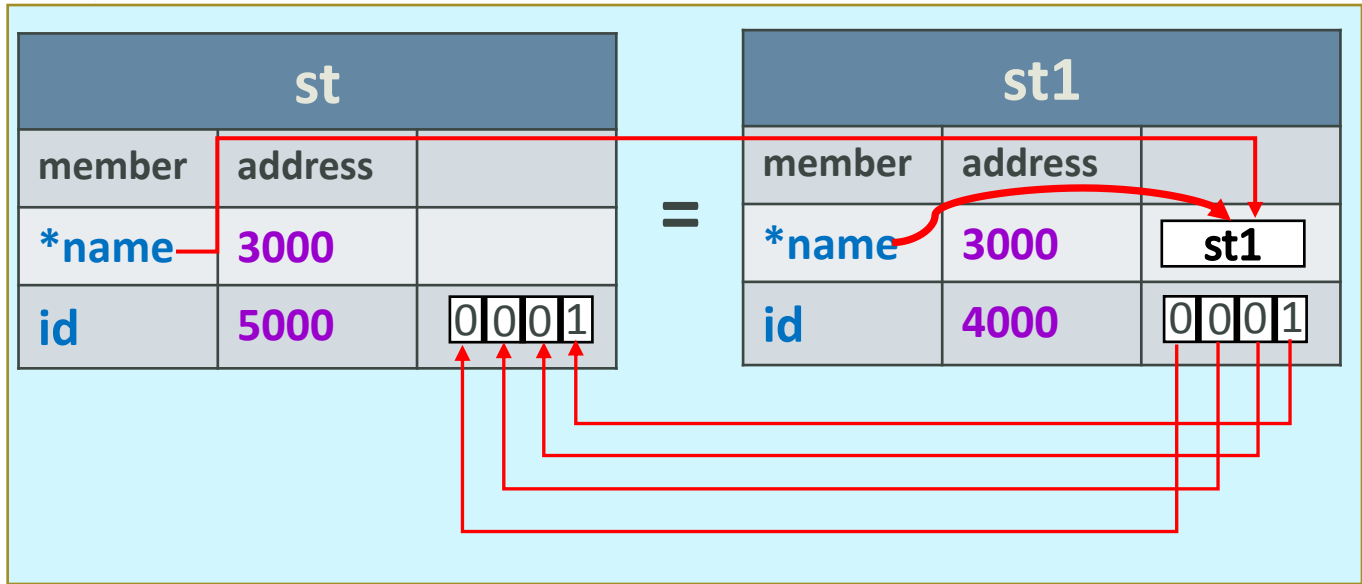
```
void func(student st) {
cout << st.getId()<<endl;
}

int main(){
student st1("St1",1);
func(st1);
cout<<"Finish"<<endl;
}
```

OUTPUT??

Constructing st1

st = st1(bitwise copy)



Constructor-Destructor Call

```
class student {
    int id;
    char* name;
public:
    student(char* p , int q) {
    id=q;
    name = new char[strlen(p)];
    strcpy(name,p);
    cout << "Constructing: "<<name<<endl;
    }
    ~student(){
    cout << "Destructing "<<name<<endl;
    delete [] name;
    }
    int getId() {return id;}
};
```

```
void func(student st) {
cout << st.getId()<<endl;
}

int main(){
student st1("St1",1);
func(st1);
cout<<"Finish"<<endl;
}
```

OUTPUT??

Constructing st1
1

st		
member	address	
*name	3000	
id	5000	0 0 0 1

=

st1		
member	address	
*name	3000	st1
id	4000	0 0 0 1

Constructor-Destructor Call

```
class student {
    int id;
    char* name;
public:
    student(char* p , int q) {
    id=q;
    name = new char[strlen(p)];
    strcpy(name,p);
    cout << "Constructing: "<<name<<endl;
    }
    ~student(){
    cout << "Destructing "<<name<<endl;
    delete [] name;
    }
    int getId() {return id;}
};
```

```
void func(student st) {
cout << st.getId()<<endl;
}

int main(){
student st1("St1",1);
func(st1);
cout<<"Finish"<<endl;
}
```

OUTPUT??

Constructing st1
1
Destructing st1

st		
member	address	
*name	3000	
id	5000	0 0 0 1

=

st1		
member	address	
*name	3000	
id	4000	0 0 0 1

Constructor-Destructor Call

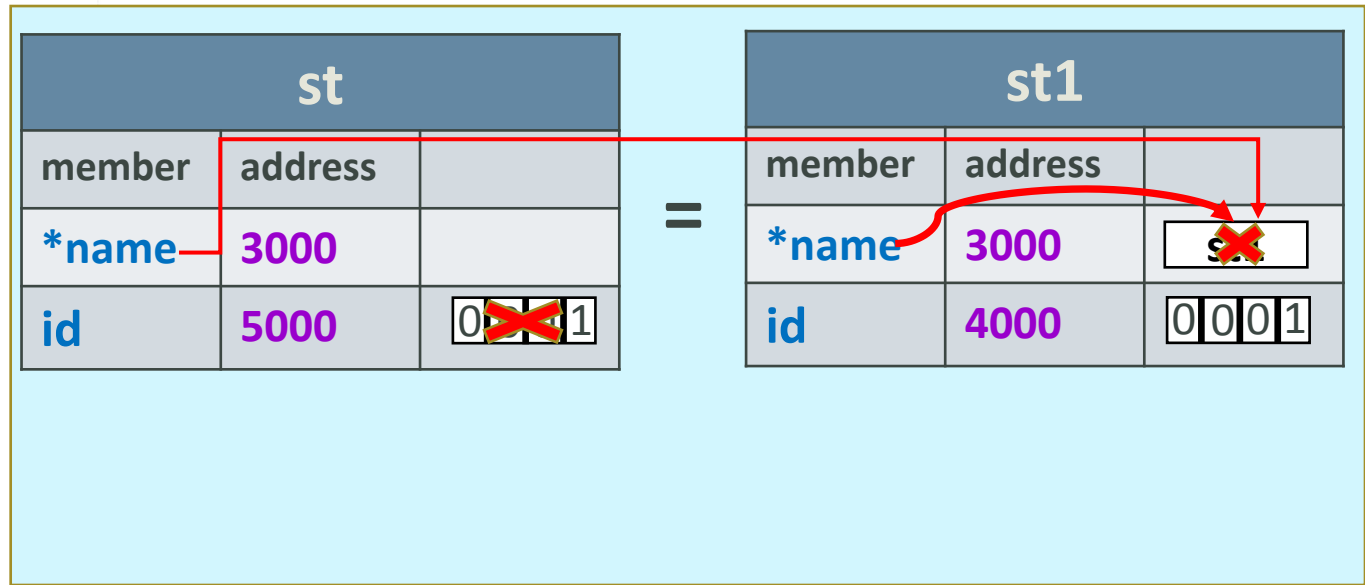
```
class student {
    int id;
    char* name;
public:
    student(char* p , int q) {
    id=q;
    name = new char[strlen(p)];
    strcpy(name,p);
    cout << "Constructing: "<<name<<endl;
    }
    ~student(){
    cout << "Destructing "<<name<<endl;
    delete [] name;
    }
    int getId() {return id;}
};
```

```
void func(student st) {
cout << st.getId()<<endl;
}

int main(){
student st1("St1",1);
func(st1);
cout<<"Finish"<<endl;
}
```

OUTPUT??

Constructing st1
1
Destructing st1
Finish



Constructor-Destructor Call

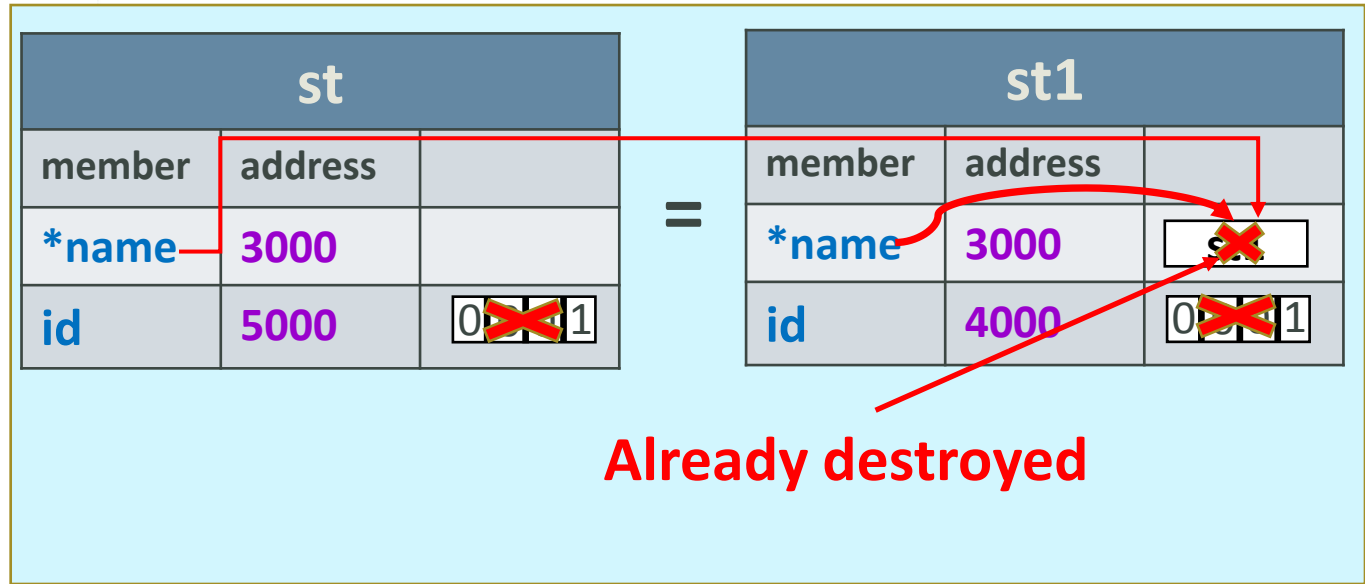
```
class student {
    int id;
    char* name;
public:
    student(char* p , int q) {
    id=q;
    name = new char[strlen(p)];
    strcpy(name,p);
    cout << "Constructing: "<<name<<endl;
    }
    ~student(){
    cout << "Destructing "<<name<<endl;
    delete [] name;
    }
    int getId() {return id;}
};
```

```
void func(student st) {
cout << st.getId()<<endl;
}

int main(){
student st1("St1",1);
func(st1);
cout<<"Finish"<<endl;
}
```

OUTPUT??

Constructing st1
1
Destructing st1
Finish
Destructing st1



Solution??

- Assignment
- Find out the solution

Constructor-Destructor Call

```
class student {
    int id;
    char* name;
public:
    student(char* p , int q) {
    id=q;
    name = new char[strlen(p)];
    strcpy(name,p);
    cout << "Constructing: "<<name<<endl;
    }
    ~student(){
    cout << "Destructing "<<name<<endl;
    delete [] name;
    }
    int getId() {return id;}
};
```

```
void func(student &st) {
cout << st.getId()<<endl;
}

int main(){
student st1("St1",1);
func(st1);
cout<<"Finish"<<endl;
}
```

3.3.20

Constructor-Destructor Call

```
class student {
    int id;
    char* name;
public:
    student(char* p , int q) {
    id=q;
    name = new char[strlen(p)];
    strcpy(name,p);
    cout << "Constructing: "<<name<<endl;
    }
    ~student(){
    cout << "Destructing "<<name<<endl;
    delete [] name;
    }
    int getId() {return id;}
};
```

```
void func(student *st) {
cout << st->getId()<<endl;
}

int main(){
student st1("St1",1);
func(&st1);
cout<<"Finish"<<endl;
}
```


Thank
you!