

Object Oriented Programming

const static class members

Lecture 9

Static, const class members
static Class Members

- Static member variables
 - All objects of class 'share' one copy

- One object changes it
- all see change
- Useful for ‘tracking’
 - How often a member function is called
 - How many objects exist at given time
 - Place keyword *static* before **data type**

`static` Class Members

- **static** class members
 - Shared by all objects of a class

- They do not belong to any particular instance, of a class
- Efficient when a single copy of data is enough
- Only the **static** variable has to be updated
- May seem like global variables, but have class scope
- only accessible to objects of same class
- Exist even if no instances (objects) of the class exist
- A static data member: **class variable**
- A non-static data member: **instance variable**
- Both variables and functions can be **static**

Declaring **static** data member syntax

- `static` keyword is used to make a data member static
- Static data member is declared inside the class
- But they are defined outside the class

```
class ClassName
{
    ...
public:
    static dataType variableName; //declaration
    ...
};
Classname::variableName = 0; //initialization
```

static Class Members

- **static** variables
 - Static variables are accessible through any object of the class
 - Example: `emp.count++`
- **public static** variables
 - Can also be accessed using scope resolution

`operator(: :) Employee::count`

Static Data Member class Rectangle

```
{  
};
```

```
private:
```

```
int width;
```

```
int length;
```

```
public:
```

```
static int count;
```

```
Rectangle r1;
```

```
Rectangle r2;
```

```
Rectangle r3;
```

```
r1
```

```
void set(int w,  
int l);
```

```
int area();
```

```
count
```

```
r2
```

```
width
```

```
length
```

width length

width

length

r3

Example: static data member

```
class Rectangle {  
public:
```

```
Rectangle(int w=5,int l=10) {  
width=w; length=l; count++; }  
void set(int w, int l);  
int area();
```

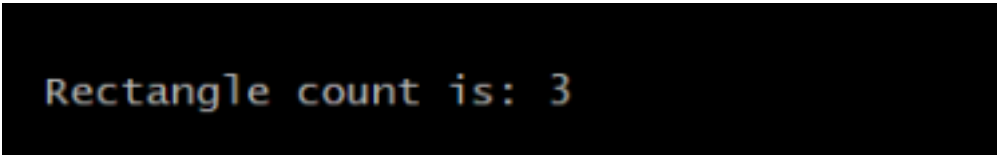
```
static int count;
```

```
private:
```

```
int width;  
int length;  
};
```

```
int Rectangle::count=0; //initialize static data member at  
global name space
```

```
int main() {
```



```
Rectangle count is: 3
```



```
Rectangle r1; //default constructor
Rectangle r2(2,4);
Rectangle r3(3,6);
cout<<"Rectangle count is:
"<<Rectangle::count<<endl; cout<<"Rectangle count
is: "<<r1.count<<endl;
```

Example: declaring static data member as Private

- If static data members are not explicitly initialized at the time of definition then they are initialized to 0.

```
int Student::noOfStudents; is equivalent to
int Student::noOfStudents = 0;
```

- Static keyword will not be used for definition.

```
class Student {
```

```
private:
    static int noOfStudents;
public:
    ...
};

int Student::noOfStudents = 0;

/*private static member cannot be accessed
outside the class except for
initialization*/
```

CS1004 - Objected Oriented Programming 9

static Data Member and Member Functions

- To access private `static` class member, provide a public `static` member function
- Call the `static` function by prefixing its name with the ***class name*** and scope resolution operator.

- A `static` member function is a service of the calls, not a of a specific object of the class.
- A `public static` member function can be defined as:

public:

```
static int getCount() { return count; }
```

- We can use the `public static` member function even if the object is not instantiated:

```
Rectangle::getCount();
```

- `static` member function cannot call non-static member function.

`class Rectangle {` **Example: static data member**

public:

```
Rectangle(int w=5,int l=10) {
```

```
Width=w; length=l; count++; }
```

```
void set(int w, int l);
```

```
static int getcount() { return is private };
```

```
count;} private:
```

```
int width; int length;
```

```
static int count; //note: count
```

Since getcount() is static, only static members can be referenced here.

```
int Rectangle::count=0; //initialize static data member at global name space
```

```
int main() {
```

```
cout<<"Rectangle count before objects instantiation:
```

```
"<<Rectangle::getcount()<<endl;
```

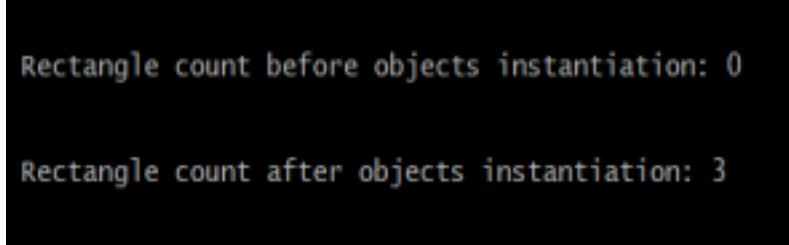
```
Rectangle r1; //default constructor
```

```
Rectangle r2(2,4);
```

```
Rectangle r3(3,6);
```

```
cout<<"Rectangle count after objects instantiation:
```

```
"<<Rectangle::getcount()<<endl;
```



```
Rectangle count before objects instantiation: 0
```

```
Rectangle count after objects instantiation: 3
```

}₁₁

Life of data **static** members

- They are created even when there is no object of a class

- They remain in memory even when all objects of a class are

destroyed

CS1004 - Objected Oriented Programming 12

`const` member function

- There are functions that are meant to be *read only*

- Keyword **const** is placed at the end of the parameter list CS1004 -

const member function

- There must exist a mechanism to detect error if such functions accidentally change the data member
- They are just “*read-only*”
- Errors due to typing are also caught at comp

Example

- Usually Getter functions are declared const to avoid any unintentional modification to the data members.

```
class Student {  
public:
```

```
int getRollNo() const {  
    return rollNo;  
}  
};
```

Example

```
bool Student::isRollNo(int aNo) {  
    if (rollNo == aNo) {  
        return true;  
    }  
}
```

```
    return false;
}
```

- The above utility function can sometimes be mistakenly written as:

```
bool Student::isRollNo(int aNo) {
    /*undetected typing mistake*/
    if (rollNo = aNo) {
        return true;
    }
    return false;
}
```

- With **const** this error is detected by compiler

```
bool Student::isRollNo(int aNo) const {
    /*compiler error*/
    if (rollNo = aNo) {
        return true;
    }
    return false;
}
```

const function

- Constant member function cannot change data member
- Constant member function cannot access non-constant member functions, even if the latter does not modify any data member
- In addition, constructor and destructor are used to modify the object to a well defined state
- Constructors and Destructors cannot be **const**

Constant data member

- Change the class Student such that a student is given a roll number when the object is created and cannot be changed afterwards

```
class Student {  
    ...  
    int rollNo;  
public:  
    Student(int aNo);  
    int getRollNo();  
    void setRollNo(int aNo);  
    ...  
};
```

Modified **Student** class

- WE MAY DECLARE IT AS CONST, BUT THEN WE CANNOT EVEN INITIALIZE

```
IT.class Student {
```

```
    ...
    const int rollNo;
public:
    Student(int aNo);
    int getRollNo();
    void setRollNo(int aNo);
    ...
};
Student::Student(int aRollNo)
{
    rollNo = aRollNo;
    /*error: cannot modify a
```

```
    constant data member*/
}
```

Solution: Member initializer list

```
void Student::SetRollNo(int i)
{
    rollNo = i;
    /*error: cannot modify a constant
    data member*/
}
```


Member initializer list

- Constructors can use *member initializer list* to initialize const data members (and non-const) of a class.
- It is given after closing parenthesis of parameter list of constructor • In case of more then one member, use comma

separated list

CS1004 - Objected Oriented Programming 27

Class with constant data member and member initializer list

```
class Student {  
    const int rollNo;  
    string name;
```

```
    double CGPA;  
public:  
    Student(int aRollNo):rollNo(aRollNo), name(NULL), CGPA(0.0)  
    { ...  
    }  
    ...  
};
```

- Const as well as non-const data members can be initialized using Member initializer list

Class exercise

- Create a dayofyear classs with three data members, day, month, year.

Also provide member functions such as getter, setter and display()

- In main() create two objects, today and birthday.
- Since day/month/year of birthday is fixed make necessary changes in the class accordingly.

const object

- Objects can be declared constant with the use of **const** keyword •

Constant objects cannot change their state

- Example of constant objects:

- Time noon(12:00);
- Date birthday(1,1,2000);
- Date independenceDay(14,08,1947);

- The const property of an object goes into effect *after* the constructor finishes executing and ends *before* the class's destructor executes.

- Only constant member functions can be called for a constant object CS1004 - Objected

Oriented Programming 33

const object and non-const member function

```
int Student::getRollNo() {  
    return RollNo;  
}
```

```
int main() {
    const Student aStudent(20);
    int val = aStudent.getRollNo();
    //Error as const object cannot access a non const function
    return 0;
}
```

- Even though `getRollNo()` does not modify the `const aStudent` object, however it is not sufficient to indicate that `getRollNo()` is a constant function. • `getRollNo()` must be explicitly declared `const`:

```
int getRollNo() const {
    return RollNo;
}
```

const object

- `const` objects cannot access “non const” member function
- Chances of unintentional modification are eliminated

CLASS EXERCISE

- Create dayofyear class with member functions setyear() and getyear().
- In main() make two const objects, named, birthday and independence day.

- Now invoke `getyear()` function using the above two objects. CS1004 -

Initializing const objects

- A constructor must be `non-const` member function
- Invoking a `non-const` member function from the constructor call as part of the initialization of a `const` object is allowed.
- The **“constness”** of the `const` object is enforced from the time the constructor completes initialization of the object until that object destructor is called.

Constructor for the `const` object

```
class Student {  
    int rollNo;
```

```

public:
    Student(int aNo):rollNo(aNo){};
    int getRollNo() const;
    void setRollNo(int aNo);
};

int Student::getRollNo() const
{
    return rollNo;
}

int main()
{
    const Student aStudent(20);
    int val = aStudent.getRollNo();
    return 0;
}

```

- Invoking a non-const member function from the constructor call as part of the initialization of a const object is allowed.
- The “constness” of the const object is enforced from the time the constructor completes initialization of the object

until that object destructor is called.