Object Oriented Programming

Lecture 8

Dr. Naveed Anwar Bhatti

Webpage: naveedanwarbhatti.github.io

Array of Objects

Before we start "Array of Objects"

```
class Rectangle
{
    int width, height;
    static int NoOfRectangles;
public:

    static int getTotalRectangles(Rectangle &a)
    {
        a.width=0;
        return a.width;
    }
};

int Rectangle::NoOfRectangles=0;

int main()
{
    Rectangle r1;
    cout << r1.getTotalRectangles (r1);
}
```

In last lecture:

Static functions cannot access **non-static members**

Question:

What if we pass **this** pointer (or self-pointer)?

Answer:

Yes! Then it will work

Array of Objects

- [**Dynamic**] Array of objects can only be created if an object can be created without supplying an explicit initializer
- There must always be a default constructor if we want to create array of objects

Example

```
class Test
{
   int i;
public:
};

int main(){
 Test array[2];  // OK
}
```

Example

```
class Test
{
   int i;
public:
 Test();
};

int main(){
 Test array[2]; // OK
}
```

Example

```
class Test
{
   int i;
public:
 Test(int x) {i=x;}
};

int main(){
 Test array[2]; // Error
}
```

Example

```
class Test
{
   int i;
public:
 Test(int x)  {i=x};
};

int main(){
 Test array[2]= {1,2};
}
```

Array[0].i = 1
Array[1].i = 2

Explicit initializer

Example

```
class Test
{
   int i,j;
public:
 Test(int x, int y) {i=x; j=y;};
};

int main(){
 Test array[2]= {{1,1},{2,2}};
}
```

Array[0].i = 1
Array[1].i = 2

Array[0].j = 1
Array[1].j = 2

Example

```
class Test
{
   int i,j;
public:
 Test(int x, int y) {i=x; j=y;};
};

int main(){
 Test a(1,1), b(2,2);
 Test array[2]= {a,b};
}
```

Array[0].i = 1
Array[1].i = 2

Array[0].j = 1
Array[1].j = 2

Pointer to Objects

Pointer to Objects

- Pointer to objects are similar as pointer to built-in types
- They can also be used to dynamically allocate objects

Example

```
class Rectangle
{
    int width, height;
public:
    Rectangle(int x=0, int y=0);
    int get_width();
    int get_height();
};

Rectangle::Rectangle(int x = 0, int y = 0)
{
    width = x;
    height = y;
}

int Rectangle::get_width()
{
    return width;
}

int Rectangle::get_height()
{
    return height;
}
```

```
int main()
{

    Rectangle obj;
    Rectangle* ptr;
    ptr = &obj;
    ptr->get_width;
    return 0;

}
```

Case Study

Case Study

Design a class ***date*** through which user must be able to perform following operations

- Get and set current day, month and year
- Increment by **X** number of days, months and year
- Set global ***Exam date***

Attributes

- Attributes that can be seen in this problem statement are
  - Day
  - Month
  - Year
  - Exam date

Attributes

- The Exam date is a feature shared by all objects
  - This attribute must be declared a static member

Structure of Files

Date.h

Date.cpp

main.cpp

Attributes in Date.h

```
class Date
{
 int day;
 int month;
 int year;
 static Date ExamDate;
…
};
```

Interfaces

- getDay
- getMonth
- getYear
- setDay
- setMonth
- setYear
- addDay
- addMonth
- addYear
- setExamDate
- getExamDate

Interfaces

- As the **Exam date** is a static member the interface **setExamDate** and **getExamDate** should also be declared static

Interfaces in Date.h

```
class Date{
…
public:
  void setDay(int aDay);
  int getDay() const;
  void addDay(int x);
  …
…
};
```

Interfaces in Date.h

```
class Date{
…
public:
  static void setExamDate(
     int aDay,int aMonth, int aYear);
…
};
```

Constructors and Destructors Interfaces in Date.h

```
  Date(int aDay = 0,
    int aMonth= 0, int aYear= 0);

  ~Date(); //Destructor
};
```

Implementation of Date Class

- The static member variables must be initialized

Date Date::ExamDate (07,3,2020);

Constructors

```
Date::Date(int aDay, int aMonth, int aYear)
{
        setDay(aDay);

      //similarly for other members
}
```

Destructor

- We are not required to do any house keeping chores in destructor

```
Date::~Date
{
}
```

Getter and Setter

```
void Date::setMonth(int a)
{
 if(a > 0 && a <= 12){
     month = a;
}
int getMonth() const{
 return month;
}
```

addYear

```
void Date::addYear(int x){
 year += x;
 if(day == 29 && month == 2 && !leapyear(year)){
     day = 1;
     month = 3;
 }
}
```

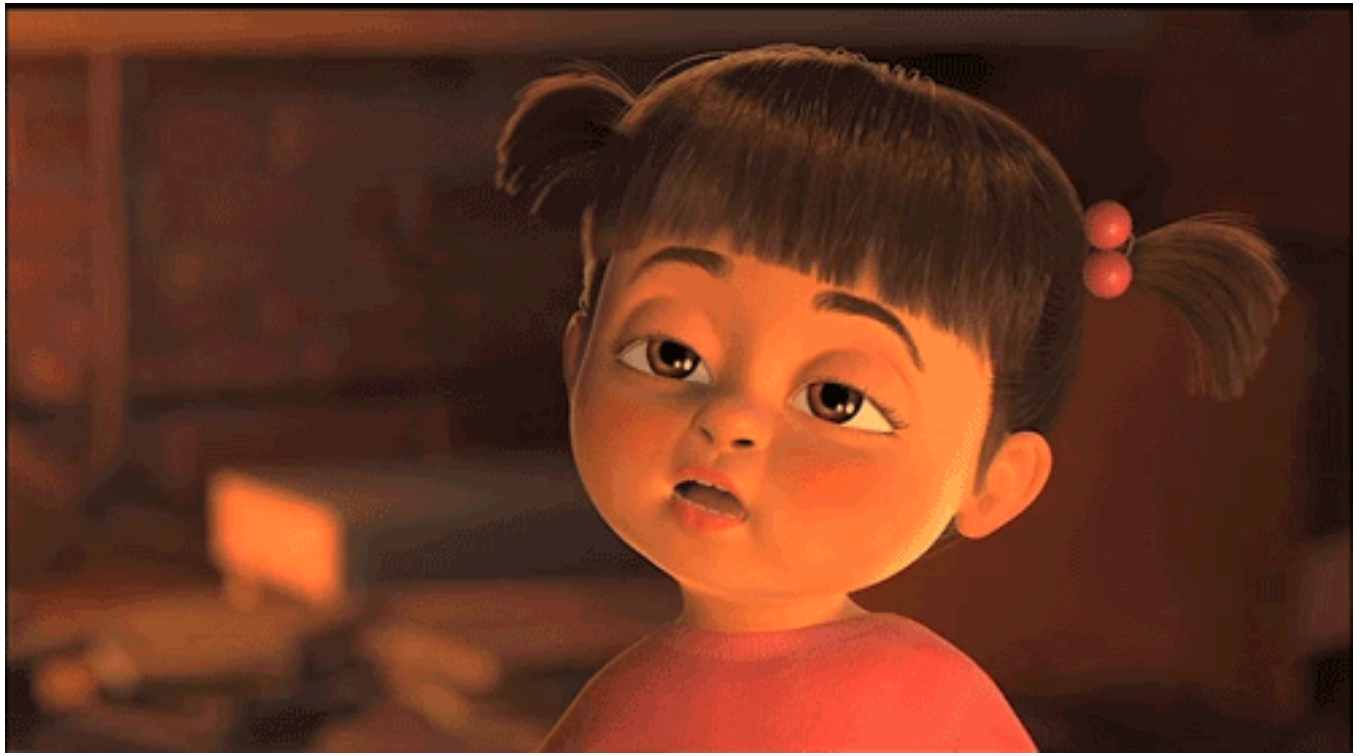
Helper Function

```
class Date{
…
private:
 bool leapYear(int x) const;
…
};
```

Helper Function

```
bool Date::leapYear(int x) const{
 if((x%4 == 0 && x%100 != 0)||(x%400==0)){
     return true;
 }
 return false;
}
```

Thanks a lot



If you are taking a Nap, **wake up**.....Lecture Over