Object Oriented Programming

Lecture

Separation of ***interface*** and ***implementation***

Separation of interface and implementation

- Public member function exposed by a class is called interface

- Separation of implementation from the interface is good software engineering

Separation of interface and implementation

- Usually functions are defined in implementation files **(.cpp)** while the class definition is given in header file **(.h)**

- Some authors also consider this as separation of interface and implementation

Rectangle.h

```
class Rectangle
{
   int width, height;
public:
   Rectangle& set_width(int width);
   Rectangle& set_height(int height);
   int area();
};
```

Rectangle.cpp

```
#include “Rectangle.h”

Rectangle& Rectangle::set_width(int width)
{
    this->width = width;
    return *this;
}

Rectangle& Rectangle::set_height(int height)
{
    this->height = height;
    return *this;
}

int Rectangle::area()
{
    return width * height;
}
```

main.cpp

```
#include <iostream>
using namespace std;

#include “Rectangle.h”


int main()
{

    Rectangle r1;
    r1.set_width(10).set_height(10);
    cout << r1.area();
    return 0;
}
```

Overall Structure of the program

Rectangle.h

Structure of the class
Interface

Rectangle.cpp

Working of the class
Implementation

main.cpp

The main program

`const` Member Functions

`const` Member Functions

- There are functions that are meant to be read only
- There must exist a mechanism to detect error if such functions accidentally change the data member

`const` Member Functions

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

const Member Functions

Declaration:

```
class ClassName
{
 ReturnVal Function() const;
};
```

Definition:

```
ReturnVal ClassName::Function() const
{
 …
}
```

Example

```cpp
class Rectangle
{
   int width, height;
public:

   int get_width() const
   {
      return width;
   }

   int get_height() const
   {
      return height;
   }
};
```

`const` Functions

What a const Member Function Cannot Do
A const member function CANNOT modify member variables:

- Constant member functions cannot modify the state of any object
- They are just ***“read-only”***
- Errors due to typing are also caught at compile time

Example

```
bool Rectangle::isWidth(int W){
 if(Width == W){
     return true;
 }
 return false;
}
```

Example

```
bool Rectangle::isWidth(int W){
 /*undetected typing mistake*/

 if(Width= W){
    return true;
 }
 return false;
}
```

Example

```
bool Rectangle::isWidth(int W) const {
 /*compiler error*/

 if(Width= W){
    return true;
 }
 return false;
}
```

const Functions

- Constructors and Destructors cannot be **const**

- Constructor and destructor are used to modify the object to a well defined state

Example

```
class Rectangle{
public:
 Rectangle() const {}    //error…
 ~Rectangle() const {}   //error…
};
```

`const` Functions

- Also, **constant member function** cannot access **non-constant member functions**

Example

```
class Rectangle
{
   int width, height;
public:
   int set_width(int a)
   {
      width=a;
   }
   int get_width() const
   {
      set_width(1);
   }

};
```

Error

this Pointer and const Member Function

- ***this*** pointer is passed as constant pointer to const data in case of constant member functions

```
instead of Rectangle * const this;
           const Rectangle *const this;
```

this Pointer and const Member Function

```
Rectangle* const this;
```

Constant Pointer

```
const Rectangle* this;
```

Pointer to Constant Data

```
const Rectangle* const this;
```

Constant Pointer to Constant Data

Thanks a lot