Class: Constructors Overloading

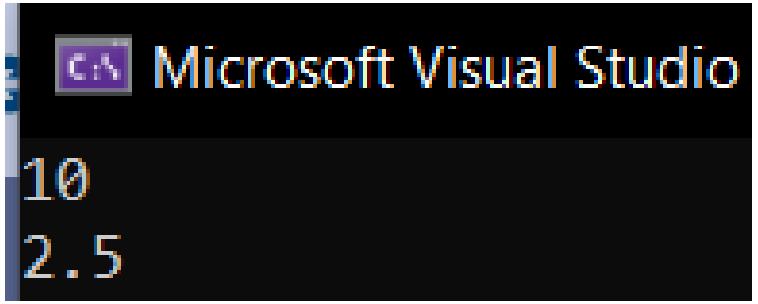
- Constructors Overloading is derived from ***Function Overloading***
- What is ***Function Overloading?***
  - Two functions can have the same name if their parameters are different;
    - ❖ either because they have a different number of parameters
    - ❖ or because any of their parameters are of a different type

Function Overloading

- Example

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

int operate(int a, int b)
{
    return (a * b);
}

double operate(double a, double b)
{
    return (a / b);
}
```

```
int main()
{
    int x = 5, y = 2;
    double n = 5.0, m = 2.0;
    cout << operate(x, y) << '\n';
    cout << operate(n, m) << '\n';
    return 0;
}
```

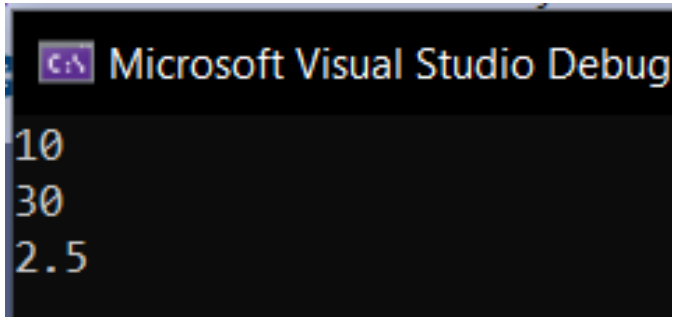


Function Overloading

- Another example

Function cannot be overloaded only by its return type. At least one of its parameters must have a different type.

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

int operate(int a, int b)
{
    return (a * b);
}

int operate(int a, int b, int c)
{
    return (a * b * c);
}

double operate(double a, double b)
{
    return (a / b);
}

int main()
{
    int x = 5, y = 2, z = 3;
    double n = 5.0, m = 2.0;
    cout << operate(x, y) << '\n';
    cout << operate(x, y, z) << '\n';
    cout << operate(n, m) << '\n';
    return 0;
}
```



Class: Constructors Overloading

- Back to Constructor Overloading;
  - Like function, constructor can also be overloaded with different versions taking different parameters

```
Rectangle::Rectangle() {
    width = 5;
    height = 5;
}

Rectangle::Rectangle(int a, int b) {
    width = a;
    height = b;
}
```

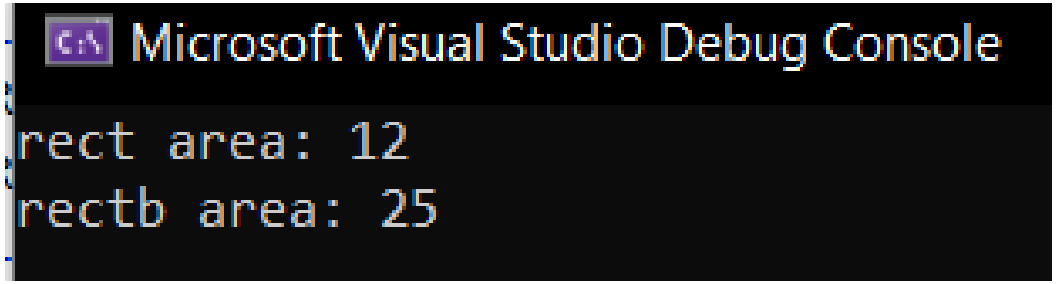
Class: Constructors Overloading

- Complete Example

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

Rectangle::Rectangle() {
    width = 5;
    height = 5;
}

Rectangle::Rectangle(int a, int b) {
    width = a;
    height = b;
}
```

Is called "*default constructor*".

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

int main() {
    Rectangle rect(3, 4);
    Rectangle rectb;
    cout << "rect area: " << rect.area() << endl;
    cout << "rectb area: " << rectb.area() << endl;
    return 0;
}
```



Class: Destructor

- Automatically called when class object passes **out of scope** or is **explicitly deleted**
- Mainly used to de-allocate the memory that has been allocated for the object by the constructor (or any other member function).
- Syntax is same as constructor except preceded by the tilde sign

```
~class_name() { };    //syntax of destructor
```

- **Neither takes any arguments nor does it returns value**
- **Can’t be overloaded**

Class: Destructor

- Example (out-of-scope)

```
class Rectangle {
    int width, height;
public:
    Rectangle();
    ~Rectangle();
};

Rectangle::Rectangle() {
    cout << "Hey look I am in constructor" << endl;
}

Rectangle::~Rectangle() {
    cout << "Hey look I am in destructor" << endl;
}
```

```
int main() {
    Rectangle rect;
    return 0;
}
```

```
Hey look I am in constructor
Hey look I am in destructor
```

Class: Destructor

- Example (out-of-scope)

```
class Rectangle {
    int width, height;
public:
    Rectangle();
    ~Rectangle();
};

Rectangle::Rectangle() {
    cout << "Hey look I am in constructor" << endl;
}

Rectangle::~Rectangle() {
    cout << "Hey look I am in destructor" << endl;
}
```

```
int main() {
    Rectangle *rect;
    return 0;
}
```

Class: Destructor

- Example (out-of-scope)

```
class Rectangle {
    int width, height;
public:
    Rectangle();
    ~Rectangle();
};

Rectangle::Rectangle() {
    cout << "Hey look I am in constructor" << endl;
}

Rectangle::~Rectangle() {
    cout << "Hey look I am in destructor" << endl;
}
```

```
int main() {
    Rectangle *rect= new Rectangle;
    return 0;
}
```

```
Hey look I am in constructor
```

Class: Destructor

- Example (delete)

```
class Rectangle {
    int width, height;
public:
    Rectangle();
    ~Rectangle();
};

Rectangle::Rectangle() {
    cout << "Hey look I am in constructor" << endl;
}

Rectangle::~Rectangle() {
    cout << "Hey look I am in destructor" << endl;
}
```

```
int main() {
  Rectangle *rect;
  rect = new Rectangle;
  delete rect;
  return 0;
}
```

```
Hey look I am in constructor
Hey look I am in destructor
```

Class: Destructor

- Example (when its useful)

```
class Rectangle {
    int *width, *height;
public:
    Rectangle();
    ~Rectangle();
};

Rectangle::Rectangle() {
    cout << "Hey look I am in constructor" << endl;
    width = new int[10];
    height = new int[10];
}

Rectangle::~Rectangle() {
    cout << "Hey look I am in destructor" << endl;
    delete [] width;
    delete [] height;
}
```

```
int main() {
    Rectangle rect;
    return 0;
}
```

```
Hey look I am in constructor
Hey look I am in destructor
```

Thanks a lot

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