

PROGRAMMING LANGUAGES 1

Structures & Classes



STRUCTURES

```
struct rectangle {  
    float length;  
    float width;  
};  
  
int main() {  
    rectangle rect;  
    cin >> rect.length;  
    cin >> rect.width;  
...  
}
```

STRUCTURES & FUNCTIONS

```
void setWidth (rectangle rect, float width) {  
    rect.width = width;  
}  
  
void setLength (rectangle rect, float length) {  
    rect.length = length;  
}  
  
float calculateArea(rectangle rect) {  
    return rect.length * rect.width;  
}
```

CLASSES

- A class consists of member variables and member functions
- Member functions include:
 - Constructor(s)
 - Destructor
 - Setters & Getters
 - Other member functions
- Each class has:
 - Declaration (typically in header file)
 - Definition (typically in source file)
 - Instantiation (in another source file)
- An object is an instance of a class

CLASS DECLARATION

```
class rectangle {  
private:  
    float width;  
    float length;
```

```
public:  
    rectangle(float, float); //  
    constructor  
    void setWidth(float);  
    void setLength(float);  
    float getLength();  
    float getWidth();  
    float area( );  
    ~rectangle(); // destructor  
};
```

CLASS DEFINITION

```
rectangle::rectangle(float l, float w) {  
    length = l;  
    width = w;  
}  
void rectangle::setWidth(float w) {  
    width = w;  
}  
void rectangle::setLength(float l) {  
    length = l;  
}  
  
float rectangle::getLength() {  
    return length;  
}  
float rectangle::getWidth() {  
    return length;  
}  
float rectangle::area( ) {  
    return length * width;  
}  
rectangle::~~rectangle() { }
```

OBJECT INSTANTIATION

```
int main( ) {  
    rectangle rect(5,3); // Object Instantiation  
    cout << rect.area( ); // Call member function  
}
```

TYPICALLY

rectangle.h

```
#ifndef RECT_H
#define RECT_H
class rectangle {
private:
    float length;
    float width;
public:
    rectangle(float, float);
    void setWidth(float);
    void setLength(float);
    ...
};
#endif
```

rectangle.cpp

```
#include "rectangle.h"

rectangle::rectangle(float l, float w) {
    length = l;
    width = w;
}

void rectangle::setWidth(float w) {
    width = w;
}

void rectangle::setLength(float l) {
    length = l;
}
```

main.cpp

```
#include "rectangle.h"

int main( )
{
    rectangle rect(5,3);
    cout << rect.area();
}
```

CONSTRUCTOR

- A member function that has the exact same name as the class
- Has no return or return type
- Called automatically on object construction
- Many constructors can be defined (different arguments)
- A default trivial constructor is provided if no constructor is defined
 - Takes no arguments
 - Does not set any values
- No default constructor is provided if the user defines any

COPY CONSTRUCTOR

- A constructor that takes another object of the class as input
- Copies given object into this object
- Each class has a default copy constructor
 - copies all member variables into the current object
- The default copy constructor is not suitable if array members exist
 - Does not copy the array
- No default copy constructor is provided if one is defined
- If a member is an object of another class
 - Copy constructor of the other class is called

COPY CONSTRUCTOR - EXAMPLE

```
rectangle::rectangle(const rectangle& other) {  
    length = other.length;  
    width = other.width;  
}
```

INITIALIZER LIST

```
rectangle::rectangle(float l, float w) : length(l), width(w) {}  
rectangle::rectangle(const rectangle& o) : length(o.length), width(o.width) {}
```

DESTRUCTOR

- A member function that has the same name of class preceded by ~
- Takes no arguments and has no return
- Called automatically when an object goes out of scope
- Clean up tasks should be implemented here, e.g. delete a pointer
- A default trivial destructor is provided if none is defined
 - Does nothing

ACCESS MODIFIERS

- Each member (function/variable) is defined as private, protected, or public
- A private member cannot be accessed from a non-member function
- A public member is accessible by any function
- Variables should be defined as private members
- Default is private if no modifier is used
- In C++, struct is a class whose members are public by default

ACCESS MODIFIERS - EXAMPLE

```
int main ( ) {  
    rectangle rect(3,5)  
    cout << rect.length; X // length is private  
    cout << rect.area( ); ✓ // area( ) is public  
}
```

```
class rectangle {  
    private:  
        float length;  
        float width;  
    public:  
        rectangle(float, float);  
        void setWidth(float);  
        void setLength(float);  
        float area( );  
};
```

FRIEND

- Another class or a class member function can be defined as a *friend*
- A friend class or function can access private members of the class

FRIEND - EXAMPLE

```
class A {  
    private:  
        int x;  
    public:  
        A() : x(0) {}  
        friend class B;  
};
```

```
class B {  
    private:  
        int y;  
    public:  
        void showA(const A& a) {  
            cout << a.x;  
        }  
};
```

GETTERS & SETTERS

- Member functions to allow setting and getting private member variables

```
int main ( ) {  
    rectangle rect(3,5)  
    cout << rect.length; X  
    cout << rect.getLength(); ✓  
    rect.length = 7; X  
    rect.setLength(7); ✓  
}
```

CONSTANT FUNCTION

- Function that should not make any changes
- Declared by adding **const** at end of function declaration
- Example: A getter function should be constant

```
class rectangle {  
public:  
    ...  
    float area() const;  
    float getLength() const;  
    float getWidth() const;  
    ...  
};
```

:: OPERATOR

- Defines scope of function (member of which class)
- :: operator with no class means global function
 - Used if same function name is used inside and outside class

OBJECTS & FUNCTIONS

- Objects should be passed by reference
 - To avoid copying the whole information of the object
 - To update members of the same object
- If object should not be modified, pass as a constant reference (e.g, const rectangle&)
- A function can return an object
 - Should be returned by reference if we need to access the same object (not a copy)

POINTER TO OBJECTS

```
rectangle * rect = new rectangle(l,w);  
cout << rect->area( );  
...  
delete rect;
```

this OPERATOR

- Pointer to the current object
- Used in definition of member functions only

```
rectangle& rectangle::setLength(float length) {  
    this -> length = length;  
    return *this;  
}
```

OPERATOR OVERLOADING

- A way to define a new behavior for an operator, e.g. +
- Uses function whose name is the keyword **operator** followed by the operator (e.g. operator+)
- The arguments of the function are the operands of the operator
- A default assignment operator is provided if none is defined
 - Copies members from RHS to LHS

```
rectangle operator+(rectangle right) {  
    return rectangle(length + right.length, width + right.width);  
}
```

STATIC MEMBERS

- Member of the class
- Shared between all objects of the class
- Initialized to zero on first object creation if not explicitly initialized

FULL EXAMPLE

<https://github.com/ProgrammingLanguagesClass/Lecture-9>

SUMMARY

- Structures
- Classes
- Constructors & Destructor
- Access Modifiers
- Getters & Setters
- Constant Functions
- this Operator
- Operator Overloading