

## Operator Overloading

What is Operator Overloading?

- Using traditional operators such as +, =, \*, etc. with user-defined types
- Allows user defined types to behave similar to built-in types
- Can make code more readable and writable
- Not done automatically (except for the assignment operator)  
They must be explicitly defined

BEGINNING C++ PROGRAMMING  
Operator Overloading

LearnProgramming  
YouTube

## Operator Overloading

What is Operator Overloading?

Suppose we have a Number class that models any number

- Using functions:

```
Number result = multiply(add(a,b),divide(c,d));
```

- Using member methods:

```
Number result = (a.add(b)).multiply(c.divide(d));
```

## Operator Overloading

What operators can be overloaded?

- The majority of C++'s operators can be overloaded
- The following operators cannot be overload

operator
::
:?
.*
.
sizeof

BEGINNING C++ PROGRAMMING  
Operator Overloading

LearnProgramming  
YouTube

## Operator Overloading

Some basic rules

- Precedence and Associativity cannot be changed
- 'arity' cannot be changed (i.e. can't make the division operator unary)
- Can't overload operators for primitive type (e.g. int, double, etc.)
- Can't create new operators
- [](), ->, and the assignment operator (=) **must** be declared as member methods
- Other operators can be declared as member methods or global functions

## Operator Overloading

MyString class declaration

```
class MyString {  
  
private:  
    char *str; // C-style string  
  
public:  
    MyString();  
    MyString(const char *s);  
    MyString(const MyString &source);  
    ~MyString();  
    void display() const;  
    int get_length() const;  
    const char *get_str() const;  
};
```

BEGINNING C++ PROGRAMMING  
Operator Overloading

LearnProgramming  
YouTube

## Operator Overloading

Copy assignment operator (=)

- C++ provides a default assignment operator used for assigning one object to another

```
MyString s1 ("Frank");  
MyString s2 = s1; // NOT assignment  
// same as MyString s2(s1);  
  
s2 = s1; // assignment
```

- Default is memberwise assignment (shallow copy)  
• If we have raw pointer data member we must deep copy

## Operator Overloading

Overloading the copy assignment operator (deep copy)

```
MyString &MyString::operator=(const MyString &rhs) {  
    if (this == &rhs)  
        return *this;  
  
    delete [] str;  
    str = new char[std::strlen(rhs.str) + 1];  
    std::strcpy(str, rhs.str);  
  
    return *this;  
}
```

BEGINNING C++ PROGRAMMING  
Copy Assignment Operator

LearnProgramming  
YouTube

## Operator Overloading

Overloading the copy assignment operator – steps for deep copy

- Check for self assignment

```
if (this == &rhs) // p1 = p1;  
    return *this; // return current object
```

- Deallocate storage for this->str since we are overwriting it

```
delete [] str;
```

BEGINNING C++ PROGRAMMING  
Copy Assignment Operator

LearnProgramming  
YouTube

## Operator Overloading

What is Operator Overloading?

Suppose we have a Number class that models any number

- Using overloaded operators

```
Number result = (a+b)*(c/d);
```

BEGINNING C++ PROGRAMMING  
Operator Overloading

LearnProgramming  
YouTube

## Operator Overloading

Some examples

- int  
 a = b + c  
 a < b  
 std::cout << a
- std::string  
 s1 = s2 + s3  
 s1 < s2  
 std::cout << s1
- MyString  
 s1 = s2 + s3  
 s1 < s2  
 s1 == s2  
 std::cout << s1
- long  
 a = b + c  
 a < b  
 std::cout << a
- Player  
 p1 < p2  
 p1 == p2  
 std::cout << p1

BEGINNING C++ PROGRAMMING  
Operator Overloading

LearnProgramming  
YouTube

## Operator Overloading

Overloading the copy assignment operator (deep copy)

```
Type &Type::operator=(const Type &rhs);
```

```
MyString &MyString::operator=(const MyString &rhs);  
  
s2 = s1; // We write this  
s2.operator=(s1); // operator= method is called
```

BEGINNING C++ PROGRAMMING  
Copy Assignment Operator

LearnProgramming  
YouTube

## Operator Overloading

Overloading the copy assignment operator – steps for deep copy

- Allocate storage for the deep copy

```
str = new char[std::strlen(rhs.str) + 1];
```

BEGINNING C++ PROGRAMMING  
Copy Assignment Operator

LearnProgramming  
YouTube

## Operator Overloading

Overloading the copy assignment operator – steps for deep copy

- Perform the copy

```
std::strcpy(str, rhs.str);
```

- Return the current by reference to allow chain assignment

```
return *this;
```

```
// s1 = s2 = s3;
```

BEGINNING C++ PROGRAMMING  
Copy Assignment Operator



by Udemy

## Operator Overloading

Overloading the Move assignment operator – steps

- Check for self assignment

```
if (this == &rhs)
    return *this; // return current object
```

- Deallocate storage for this->str since we are overwriting it

```
delete [] str;
```

BEGINNING C++ PROGRAMMING  
Move Assignment Operator



by Udemy

## Operator Overloading

Mystring operator- make lowercase

```
Mystring larry1 ("LARRY");
Mystring larry2;

larry1.display(); // LARRY
larry2 = -larry1; // larry1.operator-()

larry1.display(); // LARRY
larry2.display(); // larry
```

BEGINNING C++ PROGRAMMING  
Operator as Member Function



by Udemy

## Operator Overloading

Mystring operator==

```
bool Mystring::operator==(const Mystring &rhs) const {
    if (std::strcmp(str, rhs.str) == 0)
        return true;
    else
        return false;
}

// if (s1 == s2) // s1 and s2 are Mystring objects
```

BEGINNING C++ PROGRAMMING  
Operator as Member Function



by Udemy

## Operator Overloading

Move assignment operator (=)

- You can choose to overload the move assignment operator
- C++ will use the copy assignment operator if necessary

```
Mystring s1;
s1 = Mystring {"Frank"}; // move assignment
```

- If we have raw pointer we should overload the move assignment operator for efficiency

BEGINNING C++ PROGRAMMING  
Move Assignment Operator



by Udemy

## Operator Overloading

Overloading the Move assignment operator – steps for deep copy

- Steal the pointer from the rhs object and assign it to this->str

```
str = rhs.str;
```

- Null out the rhs pointer

```
rhs.str = nullptr;
```

- Return the current object by reference to allow chain assignment

```
return *this;
```

BEGINNING C++ PROGRAMMING  
Move Assignment Operator



by Udemy

## Operator Overloading

Mystring operator- make lowercase

```
Mystring Mystring::operator-() const {
    char *buff = new char[strlen(str) + 1];
    std::strcpy(buff, str);
    for (size_t i=0; i<strlen(buff); i++)
        buff[i] = std::tolower(buff[i]);
    Mystring temp (buff);
    delete [] buff;
    return temp;
}
```

BEGINNING C++ PROGRAMMING  
Operator as Member Function



by Udemy

## Operator Overloading

Mystring operator+ (concatenation)

```
Mystring larry ("Larry");
Mystring moe("Moe");
Mystring stooges (" is one of the three stooges");

Mystring result = larry + stooges;
// larry.operator+(stooges);

result = moe + " is also a stooge";
// moe.operator+"is also a stooge";

result = "Moe" + stooges; // "Moe".operator+(stooges) ERROR
```

BEGINNING C++ PROGRAMMING  
Operator as Member Function



by Udemy

## Operator Overloading

Overloading the Move assignment operator

```
Type &Type::operator=(Type &&rhs);
```

```
Mystring &Mystring::operator=(Mystring &&rhs);
```

```
sl = Mystring("Joe"); // move operator= called
```

```
sl = "Frank"; // move operator= called
```

BEGINNING C++ PROGRAMMING  
Move Assignment Operator



by Udemy

## Operator Overloading

Unary operators as member methods (+, -, --, ++)

```
ReturnType Type::operatorOp();
```

```
Number Number::operator-() const;
Number Number::operator++();
Number Number::operator++(int); // pre-increment
Number Number::operator++(int); // post-increment
bool Number::operator!() const;
```

```
Number n1 (100);
Number n2 = -n1; // n1.operator-()
n2 = ++n1; // n1.operator()
n2 = n1++; // n1.operator++(int)
```

BEGINNING C++ PROGRAMMING  
Operator as Member Function



by Udemy

## Operator Overloading

Binary operators as member methods (+,-,==,!,<,>, etc.)

```
ReturnType Type::operatorOp(const &Type rhs);
```

```
Number Number::operator+(const &Number rhs) const;
Number Number::operator-(const &Number rhs) const;
bool Number::operator==(const &Number rhs) const;
bool Number::operator<(const &Number rhs) const;
```

```
Number n1 (100), n2 (200);
Number n3 = n1 + n2; // n1.operator+(n2)
n3 = n1 - n2; // n1.operator-(n2)
if (n1 == n2) . . . // n1.operator==(n2)
```

BEGINNING C++ PROGRAMMING  
Operator as Member Function



by Udemy

## Operator Overloading

Mystring operator+ (concatenation)

```
Mystring Mystring::operator+(const Mystring &rhs) const {
    size_t buff_size = std::strlen(str) +
                      std::strlen(rhs.str) + 1;
    char *buff = new char[buff_size];
    std::strcpy(buff, str);
    std::strcat(buff, rhs.str);
    Mystring temp (buff);
    delete [] buff;
    return temp;
}
```

BEGINNING C++ PROGRAMMING  
Operator as Member Function



by Udemy

## Operator Overloading

Unary operators as global functions (+, -, -, !)

```
ReturnType operatorOp(Type &obj);  
  
Number operator-(const Number &obj);  
Number operator++(Number &obj); // pre-increment  
Number operator++(Number &obj, int); // post-increment  
bool operator!(const Number &obj);  
  
Number n1 (100);  
Number n2 = -n1; // operator-(n1)  
n2 = ++n1; // operator++(n1)  
n2 = n1++; // operator++(n1,int)
```

BEGINNING C++ PROGRAMMING  
Operator as Global Function

LearnProgramming

## Operator Overloading

Mystring operator- make lowercase

```
Mystring larry1 ("LARRY");  
Mystring larry2;  
  
larry1.display(); // LARRY  
  
larry2 = -larry1; // operator-(larry1)  
  
larry1.display(); // LARRY  
larry2.display(); // larry
```

BEGINNING C++ PROGRAMMING  
Operator as Global Function

LearnProgramming

## Operator Overloading

Mystring operator-

- Often declared as **friend** functions in the class declaration

```
Mystring operator-(const Mystring &obj) {  
    char *buff = new char[std::strlen(obj.str) + 1];  
    std::strcpy(buff, obj.str);  
    for (size_t i=0; i<std::strlen(buff); i++)  
        buff[i] = std::tolower(buff[i]);  
    Mystring temp (buff);  
    delete [] buff;  
    return temp;  
}
```

BEGINNING C++ PROGRAMMING  
Operator as Global Function

LearnProgramming

## Operator Overloading

Binary operators as global functions (+, -, ==, !=, <, >, etc.)

```
ReturnType operatorOp(const &Type lhs, const &Type rhs);  
  
Number operator+(const &Number lhs, const &Number rhs);  
Number operator-(const &Number lhs, const &Number rhs);  
bool operator==(const &Number lhs, const &Number rhs);  
bool operator<(const &Number lhs, const &Number rhs);  
  
Number n1 (100), n2 (200);  
Number n3 = n1 + n2; // operator+(n1,n2)  
n3 = n1 - n2; // operator-(n1,n2)  
if (n1 == n2) . . . // operator==(n1,n2)
```

BEGINNING C++ PROGRAMMING  
Operator as Global Function

LearnProgramming

## Operator Overloading

Mystring operator==

```
bool operator==(const Mystring &lhs, const Mystring &rhs){  
    if (std::strcmp(lhs.str, rhs.str) == 0)  
        return true;  
    else  
        return false;  
}
```

- If declared as a friend of Mystring can access private str attribute
- Otherwise must use getter methods

BEGINNING C++ PROGRAMMING  
Operator as Global Function

LearnProgramming

## Operator Overloading

Mystring operator+ (concatenation)

```
Mystring larry ("Larry");  
Mystring moe ("Moe");  
Mystring stooges (" is one of the three stooges");  
  
Mystring result = larry + stooges;  
// operator+(larry, stooges);  
  
result = moe + " is also a stooge";  
// operator+(moe, " is also a stooge");  
  
result = "Moe" + stooges; // OK with non-member functions
```

BEGINNING C++ PROGRAMMING  
Operator as Global Function

LearnProgramming

## Operator Overloading

Mystring operator+ (concatenation)

```
Mystring operator+(const Mystring &lhs, const myString &rhs) {  
    size_t buff_size = std::strlen(lhs.str) +  
        std::strlen(rhs.str) + 1;  
    char *buff = new char[buff_size];  
    std::strcpy(buff, lhs.str);  
    std::strcat(buff, rhs.str);  
    Mystring temp (buff);  
    delete [] buff;  
    return temp;  
}
```

BEGINNING C++ PROGRAMMING  
Operator as Global Function

LearnProgramming

## Operator Overloading

stream insertion and extraction operators (<<, >>)

```
Mystring larry ("Larry");  
  
cout << larry << endl; // Larry  
  
Player hero ("Hero", 100, 33);  
  
cout << hero << endl; // (name: Hero, health: 100, xp:33)
```

BEGINNING C++ PROGRAMMING  
Overloading Insertion and Extraction

LearnProgramming

## Operator Overloading

stream insertion and extraction operators (<<, >>)

```
Mystring larry;  
  
cin >> larry;  
  
Player hero;  
  
cin >> hero;
```

BEGINNING C++ PROGRAMMING  
Overloading Insertion and Extraction

LearnProgramming

## Operator Overloading

stream insertion and extraction operators (<<, >>)

- Doesn't make sense to implement as member methods
- Left operand must be a user-defined class
- Not the way we normally use these operators

```
Mystring larry;  
larry << cout; // huh?  
  
Player hero;  
hero >> cin; // huh?
```

BEGINNING C++ PROGRAMMING  
Overloading Insertion and Extraction

LearnProgramming

## Operator Overloading

stream insertion operator (<<)

```
std::ostream &operator<<(std::ostream &os, const Mystring &obj) {  
    os << obj.str; // if friend function  
    // os << obj.get_str(); // if not friend function  
    return os;  
}
```

- Return a reference to the ostream so we can keep inserting
- Don't return ostream by value!

BEGINNING C++ PROGRAMMING  
Overloading Insertion and Extraction

LearnProgramming

## Operator Overloading

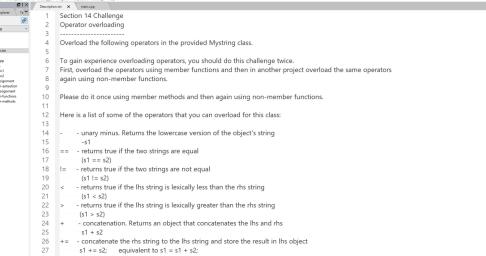
stream extraction operator (>>)

```
std::istream &operator>>(std::istream &is, Mystring &obj) {  
    char *buff = new char[1000];  
    is >> buff;  
    obj = Mystring(buff); // If you have copy or move assignment  
    delete [] buff;  
    return is;  
}
```

- Return a reference to the istream so we can keep inserting
- Update the object passed in

BEGINNING C++ PROGRAMMING  
Overloading Insertion and Extraction

LearnProgramming



The screenshot shows the Java code for the 'Section 14 Challenge' in the Eclipse IDE. The code implements the `MyString` class with various methods for string manipulation and operator overloading. The code includes comments explaining the purpose of each method and how it relates to the challenge.

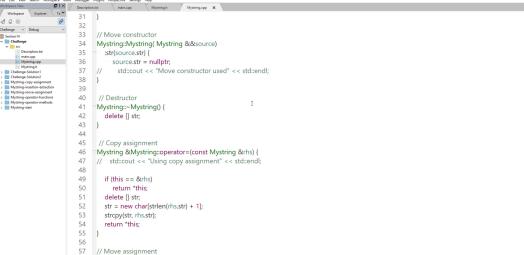
```
1 Section 14 Challenge
2 Operator overloading
3
4 Overload the following operators in the provided MyString class.
5
6 To gain experience overloading operators, you should do this challenge twice.
7 First, overload the operators using member functions and then in another project overload the same operators
8 again using non-member functions.
9
10 Please do it once using member methods and then again using non-member functions.
11
12 Here is a list of some of the operators that you can overload for this class:
13
14 - <unary minus> Returns the lowercase version of the object's string
15 - ==
16 - !=
17 - =
18 - !=
19 - ==
20 - <unary plus>
21 - <unary less than> If the lhs string is lexically less than the rhs string
22 - >
23 - >=
24 - +> concatenation: Returns an object that concatenates the lhs and rhs
25 - +
26 - +=> Append the rhs string to the lhs string and store the result in its object
27 - <unary plus plus> equivalent to  $s1 = s1 + s2$ 
28 - repeat - results in a string that is copied n times
29 - <unary plus plus plus>
30 - <unary plus plus plus plus>
31     s1 + "abc" * 3
        s1 will result as "abcabcabc"
```



```
1 //ifndef _MYSTRING_H_
2 #define _MYSTRING_H_
3
4 class MyString
5 {
6 public:
7     friend std::ostream &operator<<(std::ostream &os, const MyString &rhs);
8     friend std::istream &operator>>(std::istream &in, MyString &rhs);
9
10    private:
11        char *str; // pointer to a char[] that holds a C-style string
12
13    MyString(); // No-param constructor
14    MyString(const char *s); // Overloaded constructor
15    MyString(const MyString &source); // Copy constructor
16    MyString( MyString &source); // Move constructor
17    ~MyString(); // Destructor
18
19    MyString &operator=(const MyString &rhs); // Copy assignment
20    MyString &operator=(MyString &rhs); // Move assignment
21
22    void display() const;
23
24    int getLength() const; // getters
25    const char *getStr() const;
26
27 #endif // _MYSTRING_H_
```



```
1 //include <iostream>
2 //include <string>
3 //include "MyString.h"
4
5 //No-args constructor
6 MyString::MyString()
7 {
8     str[0] = '\0';
9     str[1] = '\0';
10 }
11
12 //Overloaded constructor
13 MyString::MyString(const char *s)
14 {
15     str[0] = '\0';
16     if(s != NULL) {
17         str = new char[strlen(s)];
18         *str = '\0';
19         for(int i = 0; i < strlen(s); i++) {
20             str[i] = s[i];
21         }
22     }
23 }
24
25 //Copy constructor
26 MyString::MyString(const MyString &source)
27 {
28     str = new char[strlen(source.str)+1];
29     strcpy(str, source.str);
30     std::cout << "Copy constructor used" << endl;
31 }
```



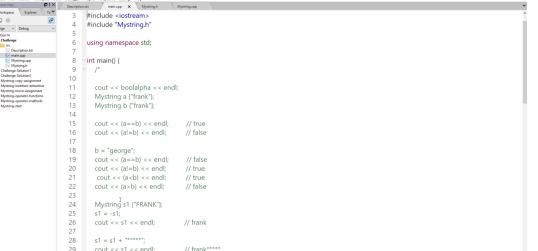
The screenshot shows the Code::Blocks IDE with the following details:

- Title Bar:** MyString.h (MyString.cpp) - Code::Blocks - Microsoft Windows
- File Menu:** File, New, Open, Save, Save As, Build Manager, Projects, Preferences, Setup, Help
- Editor Area:** Contains the C++ code for the MyString class.
- Toolbars:** Standard, Edit, View, Search, History, Tools, Manager, Projects, Preferences, Setup, Help
- Bottom Status Bar:** Line 61, Col 1, 1000, 1000, 1000, 1000, 1000

```
31 // Move constructor
32 MyString( MyString& source )
33 : str( source.str ) {
34     str[ source.length ] = '\0';
35 }
36
37 // std::cout << "Move constructor used" << std::endl;
38
39 // Destructor
40 MyString::~MyString() {
41     delete [] str;
42 }
43
44 // Copy assignment
45 MyString& operator=(const MyString& rhs) {
46     // std::cout << "Using copy assignment" << std::endl;
47
48     if (this == &rhs)
49         return *this;
50     delete [] str;
51     str = new char[ strlen(rhs.str) + 1 ];
52     strcpy(str, rhs.str);
53
54     return *this;
55 }
56
57 // Move assignment
58 MyString& operator=( MyString&rhs ) {
59     // std::cout << "Using move assignment" << std::endl;
60     if (this == &rhs)
61         return *this;
62 }
```

Learn Programming

```
40 // Destructor
41 MyString::~MyString() {
42     delete [] m;
43 }
44
45 // Copy assignment
46 MyString &MyString::operator=(const MyString &rhs) {
47     std::cout << "Using copy assignment" << std::endl;
48     if (this == &rhs)
49         return *this;
50     delete [] m;
51     m = new char[strlen(rhs.m) + 1];
52     strcpy(m, rhs.m);
53     return *this;
54 }
55
56
57 // Move assignment
58 MyString &MyString::operator=(MyString &&rhs) {
59     std::cout << "Using move assignment" << std::endl;
60     if (this == &rhs)
61         return *this;
62     delete [] m;
63     m = rhs.m;
64     rhs.m = nullptr;
65     return *this;
66 }
67
68
69 // Display method
70 void MyString::display() const {
```



The screenshot shows a Microsoft Visual Studio Code window with the following code:

```
1 #include <iostream>
2 #include "MyString.h"
3
4 using namespace std;
5
6 int main() {
7     cout << endl;
8     cout << "Mysting a ('Frank')";
9     cout << endl;
10    cout << "Mysting b ('Frank')";
11    cout << endl;
12    cout << "(a+b) << endl";
13    cout << "(a+b) << endl"; // true
14    cout << "(a+b) << endl"; // false
15
16    b = "george";
17    cout << "(a+b) << endl"; // false
18    cout << "(a+b) << endl"; // true
19    cout << "(a+b) << endl"; // true
20    cout << "(a+b) << endl"; // false
21
22    cout << "(a+b) << endl";
23
24    Mysting st1("FRANK");
25    st1 += 't';
26    cout << st1 << endl; // frank
27
28    st1 += '*****';
29    cout << st1 << endl; // frant*****
30
31    st1 += '*****';
32    cout << st1 << endl; // frant*****
```



```
18 b = "george";
19 cost <= (a+b) << endl; // false
20 cost <= (a+b) << endl; // false
21 cost <= (a+b) << endl; // false
22 cost <= (a+b) << endl; // false
23
24 MyString s1("FRANK");
25 s1 += "t"; // frank
26 cost <= s1 << endl; // frank
27
28 s1 += s1 + "*****"; // frank*****
29 cost <= s1 << endl;
30
31 s1 += "*****"; // frank*****.....
32 cost <= s1 << endl;
33
34 MyString s2("12345");
35 s2 += "1"; // 123451
36 cost <= s1 << endl; // 123451234512345
37
38 MyString s3("abcdef");
39 s3 += "a"; // abcdefabcde
40 cost <= s3 << endl; // abcdefabcdefabcdefabcdef
41
42 MyString s4("FRANK");
43 s4 += " "; // FRANK
44 cost <= s4 << endl;
45
46 s4 += " "; // frank
47 cost <= s4 << endl; // frank
```

### Inheritance

Related classes:

- Player, Enemy, Level Boss, Hero, Super Player, etc.
  - Account, Savings Account, Checking Account, Trust Account, etc.
  - Shape, Line, Oval, Circle, Square, etc.
  - Person, Employee, Student, Faculty, Staff, Administrator, etc.

---

BEGINNING C++ PROGRAMMING



Inheritance

## Accounts

- Account
    - `balance`, `deposit`, `withdraw`, ...
  - Savings Account
    - `balance`, `deposit`, `withdraw`, interest rate, ...
  - Checking Account
    - `balance`, `deposit`, `withdraw`, minimum balance, per check fee, ...
  - Trust Account
    - `balance`, `deposit`, `withdraw`, interest rate

---

BEGINNING C++ PROGRAMMING



Inheritance

## Accounts – without inheritance – code duplication

```
class Account {
    // balance, deposit, withdraw, ...
}

class Savings_Account {
    // balance, deposit, withdraw, interest rate,
}

class Checking_Account {
    // balance, deposit, withdraw, minimum balance
}

class Trust_Account {
    // balance, deposit, withdraw, interest rate,
}
```

---

BEGINNING C++ PROGRAMMING



## Inheritance

Accounts - with inheritance - code reuse

```
class Account {  
    // balance, deposit, withdraw, . . .  
};  
  
class Savings_Account : public Account {  
    // interest rate, specialized withdraw, . . .  
};  
  
class Checking_Account : public Account {  
    // minimum balance, per check fee, specialized withdraw, . . .  
};  
  
class Trust_Account : public Account {  
    // interest rate, specialized withdraw, . . .  
};
```

BEGINNING C++ PROGRAMMING

What is Inheritance?

LearnProgramming

11 min

## Inheritance

Terminology

- "Is-A" relationship
  - Public inheritance
  - Derived classes are sub-types of their Base classes
  - Can use a derived class object wherever we use a base class object
- Generalization
  - Combining similar classes into a single, more general class based on common attributes
- Specialization
  - Creating new classes from existing classes proving more specialized attributes or operations
- Inheritance or Class Hierarchies
  - Organization of our inheritance relationships

BEGINNING C++ PROGRAMMING

Terminology and Notation

LearnProgramming

11 min

## Inheritance

Terminology

- Inheritance
  - Process of creating new classes from existing classes
  - Reuse mechanism
- Single Inheritance
  - A new class is created from another 'single' class
- Multiple Inheritance
  - A new class is created from two (or more) other classes

BEGINNING C++ PROGRAMMING

Terminology and Notation

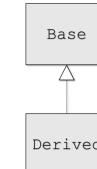
LearnProgramming

11 min

## Inheritance

Terminology

- Base class (parent class, super class)
  - The class being extended or inherited from
- Derived class (child class, sub class)
  - The class being created from the Base class
  - Will inherit attributes and operations from Base class



BEGINNING C++ PROGRAMMING

Terminology and Notation

LearnProgramming

11 min

## Inheritance

Terminology

- "Is-A" relationship
  - Public inheritance
  - Derived classes are sub-types of their Base classes
  - Can use a derived class object wherever we use a base class object
- Generalization
  - Combining similar classes into a single, more general class based on common attributes
- Specialization
  - Creating new classes from existing classes proving more specialized attributes or operations
- Inheritance or Class Hierarchies
  - Organization of our inheritance relationships

BEGINNING C++ PROGRAMMING

Terminology and Notation

LearnProgramming

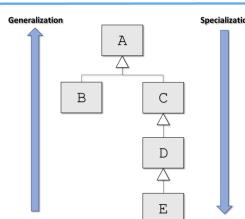
11 min

## Inheritance

Class hierarchy

Classes:

- A
  - B is derived from A
  - C is derived from A
  - D is derived from C
  - E is derived from D



BEGINNING C++ PROGRAMMING

Terminology and Notation

LearnProgramming

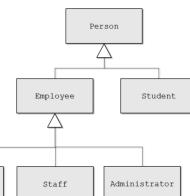
11 min

## Inheritance

Class hierarchy

Classes:

- Person
  - Employee is derived from Person
  - Student is derived from Person
  - Faculty is derived from Employee
  - Staff is derived from Employee
  - Administrator is derived from Employee



BEGINNING C++ PROGRAMMING

Terminology and Notation

LearnProgramming

11 min

## Inheritance

Public Inheritance vs. Composition

- Both allow reuse of existing classes
- Public Inheritance
  - "Is-a" relationship
    - Employee is-a Person
    - Checking Account is-a Account
    - Circle is-a Shape
- Composition
  - "has-a" relationship
    - Person has-a Account
    - Player has-a Special Attack
    - Circle has-a Location

BEGINNING C++ PROGRAMMING

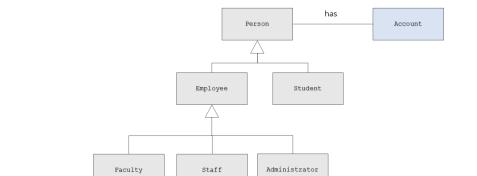
Inheritance vs Composition

LearnProgramming

11 min

## Inheritance

Public Inheritance vs. Composition



BEGINNING C++ PROGRAMMING

Inheritance vs Composition

LearnProgramming

11 min

## Inheritance

Composition

```
class Person {  
private:  
    std::string name; // has-a name  
    Account account; // has-a account  
};
```

BEGINNING C++ PROGRAMMING

Inheritance vs Composition

LearnProgramming

11 min

## Deriving classes from exiting classes

C++ derivation syntax

```
class Base {  
    // Base class members . . .  
};  
  
class Derived: access-specifier Base {  
    // Derived class members . . .  
};  
  
Access-specifier can be: public, private, or protected
```

BEGINNING C++ PROGRAMMING

Deriving Classes from Existing Classes

LearnProgramming

11 min

## Deriving classes from exiting classes

Types of inheritance in C++

- public
  - Most common
  - Establishes 'is-a' relationship between Derived and Base classes
- private and protected
  - Establishes "derived class has a base class" relationship
  - Is implemented in terms of "is-a"
  - Different from composition

BEGINNING C++ PROGRAMMING

Deriving Classes from Existing Classes

LearnProgramming

11 min

## Deriving classes from exiting classes

C++ derivation syntax

```
class Account {  
    // Account class members . . .  
};  
  
class Savings_Account: public Account {  
    // Savings_Account class members . . .  
};  
  
Savings_Account 'is-a' Account
```

BEGINNING C++ PROGRAMMING

Deriving Classes from Existing Classes

LearnProgramming

11 min

## Deriving classes from existing classes

C++ creating objects

```
Account account ();
Account *p_account = new Account ();

account.deposit(1000.0);
p_account->withdraw(200.0);

delete p_account;
```

BEGINNING C++ PROGRAMMING  
Deriving Classes from Existing Classes

{ } Learn Programming

1 lesson

## Deriving classes from existing classes

C++ creating objects

```
Savings_Account sav_account ();
Savings_Account *p_sav_account = new Savings_Account ();

sav_account.deposit(1000.0);
p_sav_account->withdraw(200.0);

delete p_sav_account;
```

BEGINNING C++ PROGRAMMING  
Deriving Classes from Existing Classes

{ } Learn Programming

1 lesson