C++
Programming Bootcamp 2

COSC2802

Topic 8 **Objects and Classes II**

Feedback | Mid-Course

Keeping up with content so far?

Completed all coding exercises:

- Assessed exercises
  - DAILY pre-workshop: meant as a quick check of the days content (@10 minutes in test conditions)
  - WEEKLY exercises: more challenging/indicative of challenge questions
- Workshop Exercises are **IMPORTANT**
  - More indicative of Programming Challenge Questions
  - Formative Feedback

Not all content in zybooks

- Additional content introduced in workshops
  - see slides on `Workshop` page for each day (Canvas) and Lectorial Recordings
- Recommended Text books
  - see the 'Workshop' page for each day (Canvas)
- Selected LinkedIn Learning videos
  - see daily modules on Canvas

Workshop Overview

1. Objects Continued: Accessing Class Members with pointers
2. Operator overloading
3. Objects/Classes and Functions

Classes and Pointers

Classes Recap

```cpp
#include <iostream>

#define LENGTH 10

class Example {                              // class name
    public:
        Example(int value);                  // constructor
        void publicMethod();                 // method
        void print();

    protected:
        int protectedVariable;               //field
        int protectedMethod(double param);   // method declaration

    private:
        double privateArray[LENGTH];
        void privateMethod(int *ptr, double &ref);
};

Example::Example(int value) {                // constructor (no return type)
    protectedVariable = value;
}
int Example::protectedMethod(double param) { // method definition
        return 0;
}
void Example::print() {                       // method definition
    std::cout << "Value of ex: " << protectedVariable << "\n";
}

int main() {
    Example ex(10);          // Object definition
    ex.print();              // accessing class member
    return EXIT_SUCCESS;
}
```

Accessing Class Members

- Class members (variables and methods) are accessed using dot '.' Syntax

```
Example ex(10);
ex.publicMethod();
```

- For pointers to object, arrow syntax '->' is a shortcut for dereferencing

```
Example* ptrEx = &ex;
```

```
(*ptrEx).publicMethod();
ptrEx->publicMethod();
```

'this' pointer

- Like in Java, C++ Classes have a special keyword **this**
  - It gives a ***pointer*** to the current object of the class
  - Using the keyword, all methods and fields of the class can be accessed
  - Works in constructors and methods

```
Example::Example(int value) {
   this->protectedVariable = value;
}

int Example::protectedMethod(double param) {
   this->protectedVariable = param;
   return 0;
}
```

Note:

- you can access member functions and variables without using *this*
- Good practice (?)
  - use it to avoid ambiguities over what is local to the function vs a class member.
  - Simpler contexts: keep code concise without *this*

‘this’ implicit parameter

When an object’s member function is called that object is also **implicitly** passed to the function.

- Compiler converts the call to a function call with a pointer to the object **implicitly** passed as a parameter e.g. *void print(ExampleClass *this)*

```
class ExampleClass {
    public:
        ExampleClass();
        void print();
    private:
        int field;
};
```

```
void ExampleClass::print() {
    std::cout << this->field << std::endl;
}
```

```
ExampleClass ex;
ex.print();
```

You can imagine it looking something like this if you want to (not actually what is happening, but functionally equivalent).

```
void ExampleClass::print(ExampleClass *this) {
    std::cout << this->field << std::endl;
}
ExampleClass ex;
ex.print(&ex);
```

Operator Overloading

Is there a problem?

```
class Location
{
private:
    int x;
    int y;

public:
    Location(int x, int y) : x(x), y(y) {}

    int getX() { return x;}
    int getY() { return y;}
    void setX(int x) { this->x = x;}
    void getY(int y) { this->y = y;}
};
```

```
int main()
{
    Location loc1(4, 5);
    Location loc2(6, 2);

    if (loc1 == loc2)
        std::cout << "SAME\n";
    else
        std::cout << "DIFFERENT\n";

    return EXIT_SUCCESS;
}
```

Is there a problem?

```
class Location
{
private:
    int x;
    int y;

public:
    Location(int x, int y) : x(x), y(y) {}

    int getX() { return x;}
    int getY() { return y;}
    void setX(int x) { this->x = x;}
    void getY(int y) { this->y = y;}

   // Overload the built-in == operator
   // for comparison of user defined objects

   bool operator==(const Location& other) const {
      return (x == other.x && y == other.y);
   }
}
```

```
int main()
{
    Location loc1(4, 5);
    Location loc2(6, 2);

    if (loc1 == loc2)
        std::cout << "SAME\n";
    else
        std::cout << "DIFFERENT\n";

    return EXIT_SUCCESS;
}
```

Operator Overloading (member function)

Revisit later: non-member functions

Can overload built-operators for user-defined objects, including:

- Assignment operators e.g. =, +=, -=, …
- Binary Arithmetic operators e.g. +, -, *, …
- Binary Comparison Operators e.g. ==, !=, ..

To overload an operator for a user defined class (ie as a member function):

- include member function named "*operator*" with symbol as suffix e.g addition "*+*"
- In operator overloading, **if an operator is overloaded as a member, then it must be a member of the object on the left side of the operator**.

```
public:
   TimeHrMn(int timeHours = 0, int timeMinutes = 0);
   void Print() const;
   TimeHrMn operator+(TimeHrMn rhs) ;
private:
   int hours;
   int minutes;
};
```

time1 + time2;

time1.operator+(time2);

Example

```
class TimeHrMn {
public:
   TimeHrMn(int timeHours = 0
            int timeMinutes = 0)
   void Print() const;
   TimeHrMn operator+(TimeHrMn rhs);
   TimeHrMn operator+(int rhsHours);
private:
   int hours;
   int minutes;
};

// Operands: TimeHrMn, TimeHrMn. Call this "A"
TimeHrMn TimeHrMn::operator+(TimeHrMn rhs) {
   TimeHrMn timeTotal;

   timeTotal.hours   = hours   + rhs.hours;
   timeTotal.minutes = minutes + rhs.minutes;

   return timeTotal;
}

// Operands: TimeHrMn, int. Call this "B"
TimeHrMn TimeHrMn::operator+(int rhsHours) {
   TimeHrMn timeTotal;

   timeTotal.hours = hours + rhsHours;
   timeTotal.minutes = minutes; // Stays same

   return timeTotal;
```

```
int main() {
   TimeHrMn time1(3, 22);
   TimeHrMn time2(2, 50);
   TimeHrMn sumTime;
   int num;

   num = 91;

   sumTime = time1 + time2; // Invokes "A"
   sumTime.Print();

   sumTime = time1 + 10;    // Invokes "B"
   sumTime.Print();

   cout << num + 8 << endl; // Invokes built-in add

   // sumTime = 10 + time1; // ERROR: No (int, TimeHrMn)
```

Example shows:

- Overloading of + operator
- Multiple times

Note that:
sumTime = 10 + time1; // doesn't work!

10 is an int, and int doesn't have an operator+(TimeHrMn) defined.

10.operator+(time1); // Invalid!

Example

```
class TimeHrMn {
public:
   TimeHrMn(int timeHours = 0
            int timeMinutes = 0)
   void Print() const;
   TimeHrMn operator+(TimeHrMn rhs);
   TimeHrMn operator+(int rhsHours);
private:
   int hours;
   int minutes;
};

// Operands: TimeHrMn, TimeHrMn. Call this "A"
TimeHrMn TimeHrMn::operator+(TimeHrMn rhs) {
   TimeHrMn timeTotal;

   timeTotal.hours   = hours   + rhs.hours;
   timeTotal.minutes = minutes + rhs.minutes;

   return timeTotal;
}

// Operands: TimeHrMn, int. Call this "B"
TimeHrMn TimeHrMn::operator+(int rhsHours) {
   TimeHrMn timeTotal;

   timeTotal.hours = hours + rhsHours;
   timeTotal.minutes = minutes; // Stays same

   return timeTotal;
```

```
int main() {
   TimeHrMn time1(3, 22);
   TimeHrMn time2(2, 50);
   TimeHrMn sumTime;
   int num;

   num = 91;

   sumTime = time1 + time2; // Invokes "A"
   sumTime.Print();

   sumTime = time1 + 10;    // Invokes "B"
   sumTime.Print();

   cout << num + 8 << endl; // Invokes built-in add

   // sumTime = 10 + time1; // ERROR: No (int, TimeHrMn)
```

Example shows:

- Multiple times

How to make both sides work?

Note that:
sumTime = 10 + time1; // doesn't work!

10 is an int, and int doesn't have an operator+(TimeHrMn) defined.

10.operator+(time1); // Invalid!

Overloading Comparison Operators (non-member)

Member function (operator+): LHS must be of the class type e.g. TimeHrMn because compiler calls member function on the LHS object.

For instance:

- time1 + time2
- time1 + 10

Both ok because member operator+ defined as member functions

- 10 + time1
- Fails because LHS is int (not a class, doesn't have functions)

Non-member function (operator+): No such restriction exists, works when LHS is not of class type e.g., int in the example

Example: operator+ from both side (non-member)

```
class TimeHrMn {
public:
  TimeHrMn(int timeHours = 0, int timeMinutes = 0)
     : hours(timeHours), minutes(timeMinutes) {}
  int GetHours() const { return hours; }
  int GetMinutes() const { return minutes; }
  void Print() const {
    cout << "H:" << hours << ", M:" << minutes << "\n";}
private:
  int hours;
  int minutes;
};
```

```
// TimeHrMn + TimeHrMn
TimeHrMn operator+(TimeHrMn lhs, TimeHrMn rhs) {
  int totalHours = lhs.GetHours() + rhs.GetHours();
  int totalMinutes = lhs.GetMinutes() + rhs.GetMinutes();
  return TimeHrMn(totalHours, totalMinutes);
}
// TimeHrMn + int (add hours)
TimeHrMn operator+(TimeHrMn lhs, int rhsHours) {
  int totalHours = lhs.GetHours() + rhsHours;
  int totalMinutes = lhs.GetMinutes();
  return TimeHrMn(totalHours, totalMinutes);
}
//  int + TimeHrMn (also adds hours)
TimeHrMn operator+(int lhsHours, TimeHrMn rhs) {
  int totalHours = lhsHours + rhs.GetHours();
  int totalMinutes = rhs.GetMinutes();
return TimeHrMn(totalHours, totalMinutes);
}
```

Example: operator+ from both side (non-member)

```
// TimeHrMn + TimeHrMn
TimeHrMn operator+(TimeHrMn lhs, TimeHrMn rhs) {
  int totalHours = lhs.GetHours() + rhs.GetHours();
  int totalMinutes = lhs.GetMinutes() + rhs.GetMinutes();
  return TimeHrMn(totalHours, totalMinutes);
}
// TimeHrMn + int (add hours)
TimeHrMn operator+(TimeHrMn lhs, int rhsHours) {
  int totalHours = lhs.GetHours() + rhsHours;
  int totalMinutes = lhs.GetMinutes();
  return TimeHrMn(totalHours, totalMinutes);
}
//  int + TimeHrMn (also adds hours)
TimeHrMn operator+(int lhsHours, TimeHrMn rhs) {
  int totalHours = lhsHours + rhs.GetHours();
  int totalMinutes = rhs.GetMinutes();
return TimeHrMn(totalHours, totalMinutes);
}
```

```
int main() {
 TimeHrMn time1(3, 22);
 TimeHrMn time2(2, 50);
 TimeHrMn sumTime;

 sumTime = time1 + time2; // TimeHrMn + TimeHrMn
 sumTime.Print();
 sumTime = time1 + 10; // TimeHrMn + int
 sumTime.Print();
 sumTime = 10 + time1; // int + TimeHrMn
 sumTime.Print();
 return EXIT_SUCCESS;
}
```

Comparison: Overloading member vs non-member

Feature	Member Function	Non-Member Function
Definition	Defined within the class	Defined outside the class
Tied to Object	Yes, operates on an instance of the class	No, operates independently of any specific object
Access to this	Has access to this pointer	No access to this pointer
Access Scope	Can access all members (private, protected, public)	Can only access public members via methods
Number of Arguments	Takes one fewer argument i.e. implicit this pointer	Requires all operands explicitly e.g., lhs, rhs
Calling Method	Called via an object e.g `ob1 + ob2 vs ob1.operator+(ob2)`	Called independently e.g., func(lhs, rhs)
Use Case	Used for instance-specific operations	Used for operations not tied to a particular object
Comparison Operators	Can be overloaded to compare the object itself	Often better suited for symmetric comparisons

Overloading | Member or non-Member function

Define as

- Member function OR non-Member function

```
bool Location::operator==(const Location &rhs) {
    return (this->getX() == rhs.getX() && this->getY() == rhs.getY());
}
```

Considerations:

- **Symmetry** e.g. a+b and b+a
  - Better as non-member function
- Access to Private Members?
  - Better as Member function
  - or use public getters/setters?
  - or *friend* (more late in the course)

See: zybooks Operator Overload Example

Classes and Functions

Classes & Functions

Pass Objects of classes to functions either by:

- Pass by Value:
  - Is possible BUT requires special COPY constructor (which we will cover later)

Not to Return the object!

```
#include <iostream>
class myClass {
public:
  myClass() { std::cout << "Default constructor called" << std::endl; }
};

myClass CreateMyClass() {
  myClass temp;
  return temp; // Returned by value
}

int main() {
  myClass obj = CreateMyClass();   // Object created and returned by value
  return EXIT_SUCCESS;
}
```

Classes & Functions

Pass Objects of classes to functions either by:

- Pass by Value:
  - Is possible BUT requires special COPY constructor (which we will cover later)

- Pass by Reference (and const reference) e.g. operator overloading

  bool operator==(const Restaurant& lhs, const Restaurant& rhs)

  - *Function can directly access and modify the original object*
  - *No need to dereference in function – more readable? Simpler syntax*
- Pass by Pointer:
  - a legacy method used by C-style programs (as well as function pointers)
  - Function can directly access and modify the original object
  - Pointer can be null so need to check