Overloading and Templates II

DM2233 ADVANCED DATA STRUCTURES & ALGORITHMS

Module Schedule

Week	Lecture	Remarks
1	Overloading and Templates I	
2	Overloading and Templates II	Labour Day (Fri) – Lab 2 Make up on 27-Apr
3	Overloading and Templates III	
4	Overloading and Templates IV	
5	Exception Handling I	
6	Exception Handling II	
7	Preprocessing / Assignment 1	Vesak Day (Mon)
Week 8 and 9: Mid-Sem Break		
10	Sorting and Searching I	
11	Sorting and Searching II	
12	Sorting and Searching III	
13	Binary Tree I	Hari Raya Puasa (Fri)
14	Lab Test	
15	Binary Tree II	
16	Binary Tree III	SG50 Day (Fri)
17	Standard Template Library / Assignment 2	National Day (Mon)

Objective

- Overloading Using Friend
- Member vs. Friend
- Overloading ()
- Overloading << and >>
- Overloading =

Overloading Using Friend

- Recall what is friend

```
class CPerson {
  private:
    int age;

  public:
    friend void findAge(CPerson);
};

void main (void)
{
   CPerson Jane;
   ...
   ...
   ...
   findAge(Jane);
}
```

Overloading Using Friend

- Recall what is overloading

Function Overloading
(Eg, Overloaded Constructor)

```
class CPerson {
  private:
    int age;

  public:
    CPerson();
    CPerson(int age);
}
```

Operator Overloading

```
class APoint {
  private:
    int m_X;
    int m_Y;

  public:
    APoint operator+(APoint &);
}
```

Overloading Using Friend

- We have seen this example

```
class rectType {
  private:
    double width;
    double height;

  public:
    rectType (double w, double h) {
      width = w;
      height = h;
    }

    rectType operator+ (rectType & input);
}
```

```
rectType rectType::operator+ (rectType & r2) {
  rectType rtemp (0.0, 0.0);

  rtemp.width = width + r2.width;
  rtemp.height = height + r2.height;

  return rtemp;
}
```

Overloading Using Friend

- We have overloaded binary operators using member functions
  - Non-member friend functions can also be overloaded
- BUT, overloading the (), [], ->, = operators must be done using member functions
  - Overloading the <<, >> operators must be done using friend functions
- When do we need to overload with friend function?
  - When we want something like rect2 = 2 + rect1;

Overloading Using Friend

- Using Friend

```
class rectType {
  private:
    double width;
    double height;

  public:
    rectType (double w, double h) {
      width = w;
      height = h;
    }

    friend rectType operator+ (rectType &, rectType &);
}
```

```
rectType operator+ (rectType & r1, rectType & r2) {
  rectType rtemp (0.0, 0.0);

  rtemp.width = r1.width + r2.width;
  rtemp.height = r1.height + r2.height;

  return rtemp;
}
```

Member vs Friend

- Comparison

Member function

```
class rectType {
  ...
  rectType operator+ (rectType &);
}
```

Friend function

```
class rectType {
  ...
  friend rectType operator+ (rectType &, rectType &);
}
```

Member vs Friend

- Comparison

Member function

```
rectType rectType::operator+ (rectType & r2) {
  rectType rtemp (0.0, 0.0);

  rtemp.width = width + r2.width;
  rtemp.height = height + r2.height;

  return rtemp;
}
```

Friend function

```
rectType operator+ (rectType & r1, rectType & r2) {
  rectType rtemp (0.0, 0.0);

  rtemp.width = r1.width + r2.width;
  rtemp.height = r1.height + r2.height;

  return rtemp;
}
```

Member vs Friend

- Back to our time example

Member function

```
class timeType {
  ...
  timeType operator+ (timeType &);
}
```

Friend function

```
class timeType {
  ...
  friend timeType operator+ (timeType &, timeType &);
}
```

Member vs Friend

- Back to our time example

```
timeType timeType::operator+
  (timeType & t2) {

  timeType ttemp (0, 0, 0);

  ttemp.sec = sec + t2.sec;
  if (ttemp.sec >= 60) {
    ttemp.min += ttemp.sec / 60;
    ttemp.sec %= 60;
  }

  ttemp.min += min + t2.min;
  if (ttemp.min >= 60) {
    ttemp.hr += ttemp.min / 60;
    ttemp.min %= 60;
  }

  ttemp.hr += hr + t2.hr;

  return ttemp;
}
```

```
timeType operator+
  (timeType & t1, timeType & t2) {

  timeType ttemp (0, 0, 0);

  ttemp.sec = t1.sec + t2.sec;
  if (ttemp.sec >= 60) {
    ttemp.min += ttemp.sec / 60;
    ttemp.sec %= 60;
  }

  ttemp.min += t1.min + t2.min;
  if (ttemp.min >= 60) {
    ttemp.hr += ttemp.min / 60;
    ttemp.min %= 60;
  }

  ttemp.hr += t1.hr + t2.hr;

  return ttemp;
}
```

Member vs Friend

- We could add our time

```
timeType t1 (3, 50, 45);
timeType t2 (2, 15, 30);
timeType t3 (0, 0, 0);

t3 = t1 + t2;
```

- Now what if we want to add a specific second to a timeType object?

```
timeType t1 (3, 50, 45);
timeType t2 (2, 15, 30);
timeType t3 (0, 0, 0);

t3 = t1 + t2;
t3 = 125 + t3;    // adding 125 seconds to t3
```

Member vs Friend

```
class timeType {
  private:
    int hr, min, sec;

  public:
    timeType (int hr, int min, int sec); // constructor

    void print (void) {
      cout << hr << "hr " << min << "min "
           << sec << "sec" << endl;
    }

    timeType operator+ (timeType &);
    friend timeType operator+ (int, timeType &);
}
```

```
t3 = 125 + t3;
```

Member vs Friend

```
timetype operator+ (int sec, timeType & input) {
  input.sec += sec;
  if (input.sec >= 60) {
    input.min += input.sec / 60;
    input.sec %= 60;
  }

  if (input.min >= 60) {
    input.hr += input.min / 60;
    input.hr %= 60;
  }

  return input;
}
```

- A more elegant solution

```
timetype operator+ (int sec, timeType & input) {
  timeType ttemp (0, 0, sec);

  input = input + ttemp;  // can we use input += ttemp?
  return input;
}
```

Overloading ()

```
class timeType {
    ...
    timeType operator+ (timeType &);
    void operator() (void);
}

void timeType::operator() (void) {
  cout << "Time value is...";
  print ();
}
```

```
timeType t1 (3, 50, 45);
timeType t2 (2, 15, 30);
timeType t3 (0, 0, 0);

t3 = t1 + t2;
t3 = 125 + t3;    // adding 125 seconds to t3

t3.print ();      // 6hr 8min 20sec
t3 ();            // Time value is... 6hr 8min 20sec
```

Overloading ()

- Overloading () with parameters

```
class timeType {
    ...
    timeType operator+ (timeType &);
    void operator() (int s);   // initialise to s sec
}

void timeType::operator() (int s) {
  timeType t1 (0, 0, 0);
  timeType t2 (0, 0, s);
  t1 = t1 + t2;

  hr = t1.hr;
  min = t1.min;
  sec = t1.sec;
}
```

```
void timeType::operator() (int s)
{
  hr = 0;
  min = 0;
  sec = s;
}
```

```
...
t3.print ();        // 6hr 8min 20sec
t3 (125);
t3.print ();        // 0hr 2min 5sec
```

Overloading ()

- Overloading () with parameters

```
class timeType {
    ...
    timeType operator+ (timeType &);
    void operator() (int m, int s);   // m min, s sec
}

void timeType::operator() (int m, int s) {
  timeType t1 (0, 0, 0);
  timeType t2 (0, m, s);
  t1 = t1 + t2;

  hr = t1.hr;
  min = t1.min;
  sec = t1.sec;
}
```

```
...
t3.print ();        // 6hr 8min 20sec
t3 (60, 125);
t3.print ();        // 1hr 2min 5sec
```

Overloading << and >>

```
class timeType {
    ...
    timeType operator+ (timeType &);
    friend ostream & operator<< (ostream &, timeType &);
}

ostream & operator<< (ostream & os, timeType & tt) {
  os << tt.hr << "hr " << tt.min << "min " << tt.sec << "sec";
  return os;
}
```

```
...
t2 (0, 10);                        // 10 sec
t3 (60, 125);                      // 60min 125sec
cout << t2 << " " << t3 << endl;   // 0hr 0min 10sec 1hr 2min 5sec
```

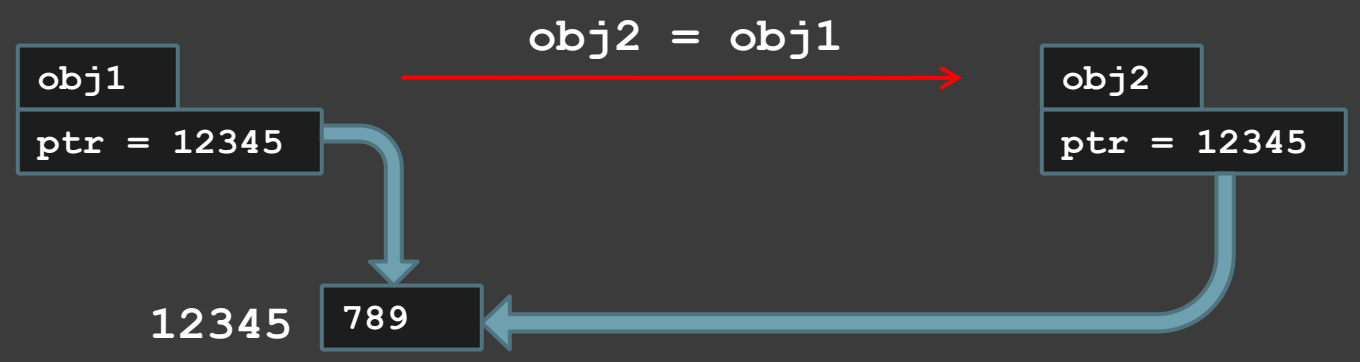
Overloading << and >>

```
class timeType {
    ...
    timeType operator+ (timeType &);
    friend ostream & operator<< (ostream &, timeType &);
    friend istream & operator>> (istream &, timeType &);
}

ostream & operator<< (ostream & os, timeType & tt) {
  os << tt.hr << "hr " << tt.min << "min " << tt.sec << "sec";
  return os;
}

istream & operator>> (istream & is, timeType & tt) {
  is >> tt.hr >> tt.min >> tt.sec;
  return is;
}
```

```
...
timeType t1 (0, 0, 0);
cout << "Enter hour, min and sec" << endl;
cin >> t1;
cout << t1 << endl;
```

Overloading =

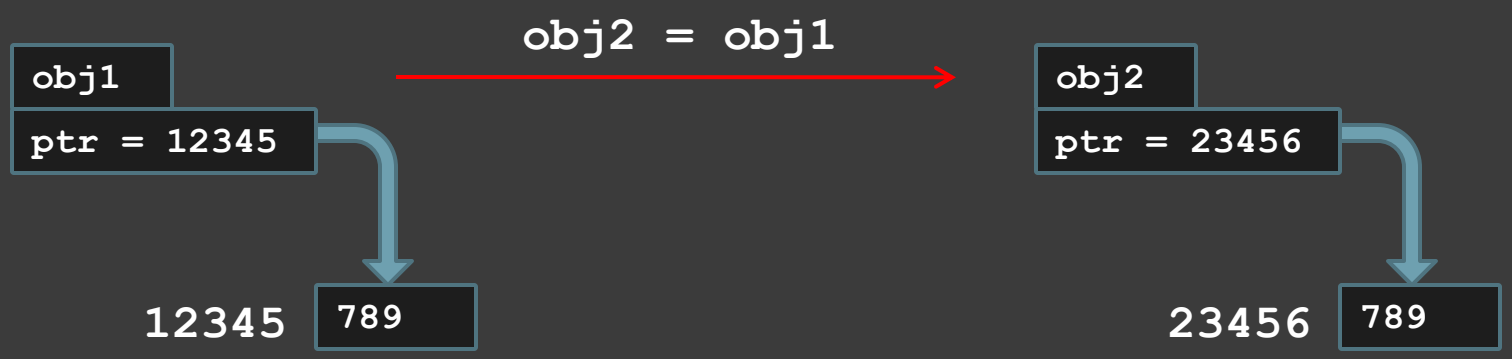
- By default, the = operator causes a member-wise copy, but this only works for classes that do not have pointers



- When *obj2* modifies *ptr, the value for *obj1* is changed too
  - This is known as shallow copy

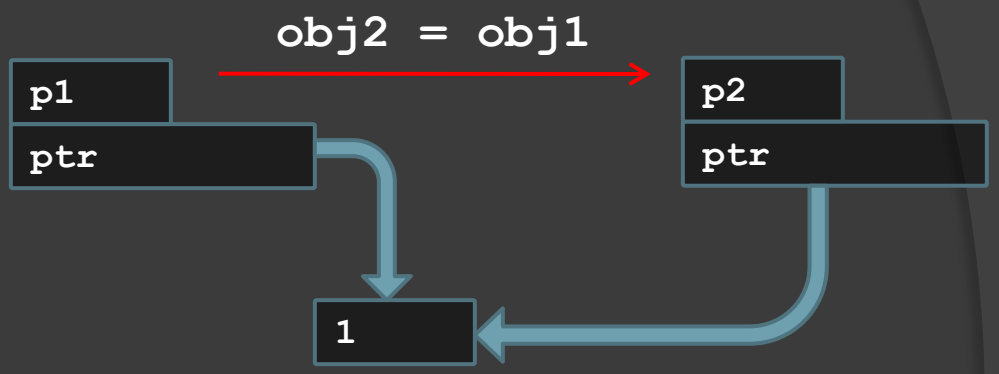
Overloading =

- What should happen is this

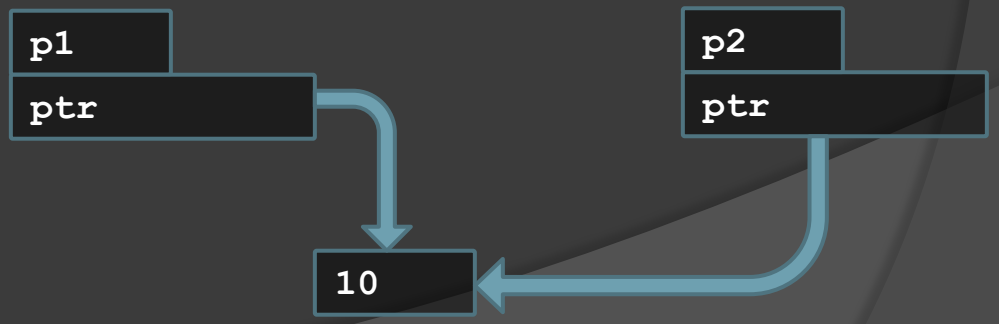


Overloading =

```
class pType {
  public:
    int * ptr;

    pType (int i = 0) {
      ptr = new int;
      *ptr = i;
    }

    ~pType (void) {
      delete ptr;
    }
}
```

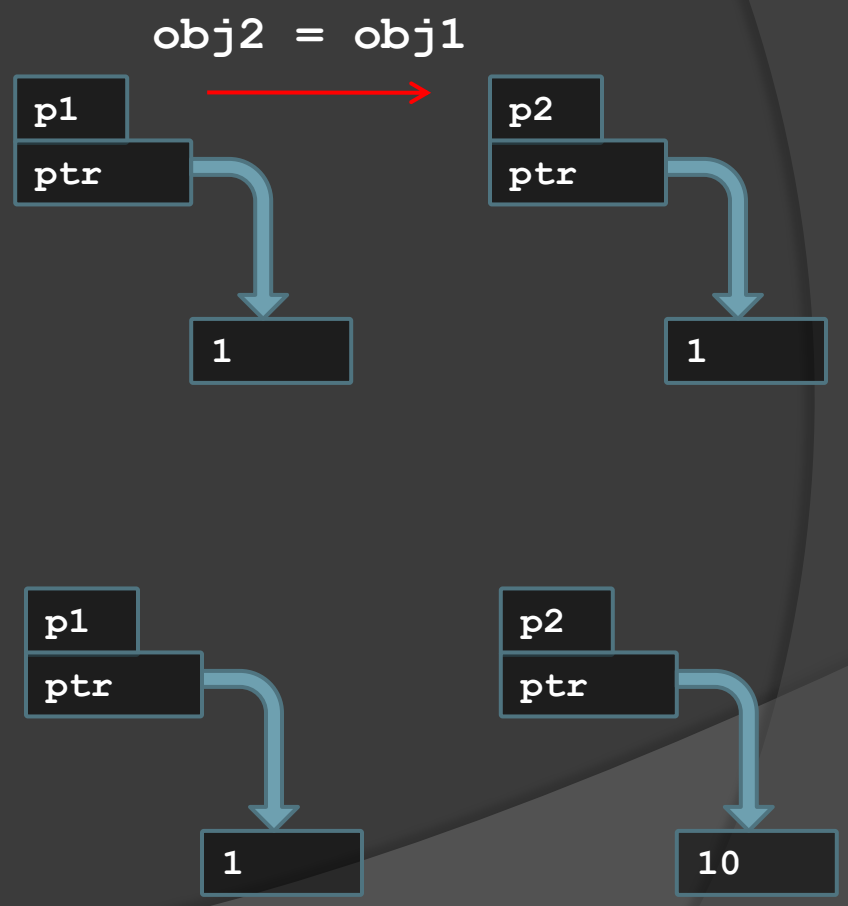



```
pType p1 (1);
pType p2;

p2 = p1;
cout << *(p1.ptr) << endl;

*(p2.ptr) = 10;
cout << *(p1.ptr) << endl;
```



Overloading =

```
class pType {
  public:
    int * ptr;

    pType (int i = 0) {
      ptr = new int;
      *ptr = i;
    }

    ~pType (void) {
      delete ptr;
    }

    pType & operator= (pType & tp) {
      if (this != &tp) {
        ptr = new int;
        *ptr = *(tp.ptr);
      }

      return *this;
    }
}
```



Summary

- We had just discussed about,
  - Overloading Using Friend
  - Member vs. Friend
  - Overloading ()
  - Overloading << and >>
  - Overloading =