Overloading and Templates III

DM2233 ADVANCED DATA STRUCTURES & ALGORITHMS

Module Schedule

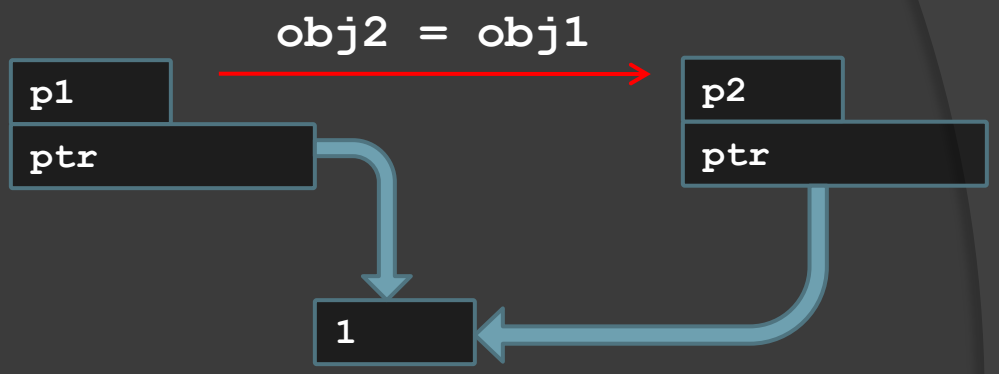
Week	Lecture	Remarks
1	Overloading and Templates I	
2	Overloading and Templates II	Labour Day (Fri) – Lab 2 Make up on 27-Apr
3	Overloading and Templates III	
4	Overloading and Templates IV	
5	Exception Handling I	
6	Exception Handling II	
7	Preprocessing / Assignment 1	Vesak Day (Mon)
Week 8 and 9: 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

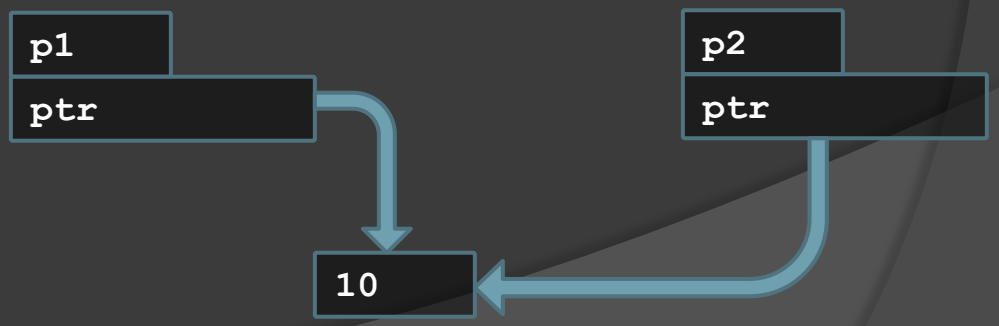
- Overloading Copy Constructor
- Overloading Unary Operator

Recalling from last week

```
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;
```



Spot the difference

```
pType p1 (1);

pType p2;
p2 = p1;

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

Using overloaded ‘=‘ operator

```
pType p1 (1);

pType p2 = p1;


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

Using copy constructor

Overloading Copy Constructor

- The following are some scenarios that will invoke the copy constructor

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

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

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

- This is different from the = operator

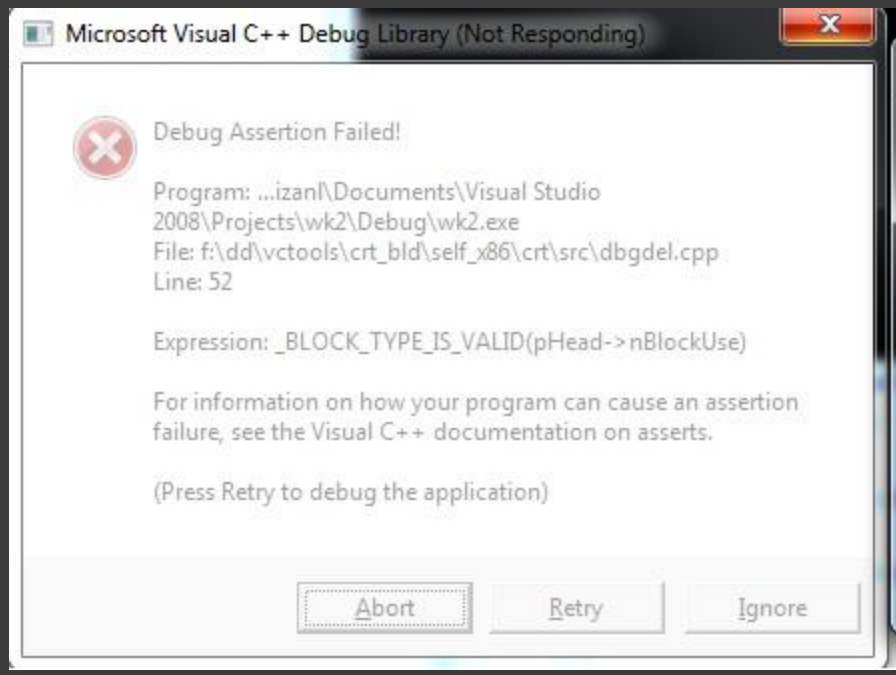
```
pType p1 (1);
pType p2;

p2 = p1;
```

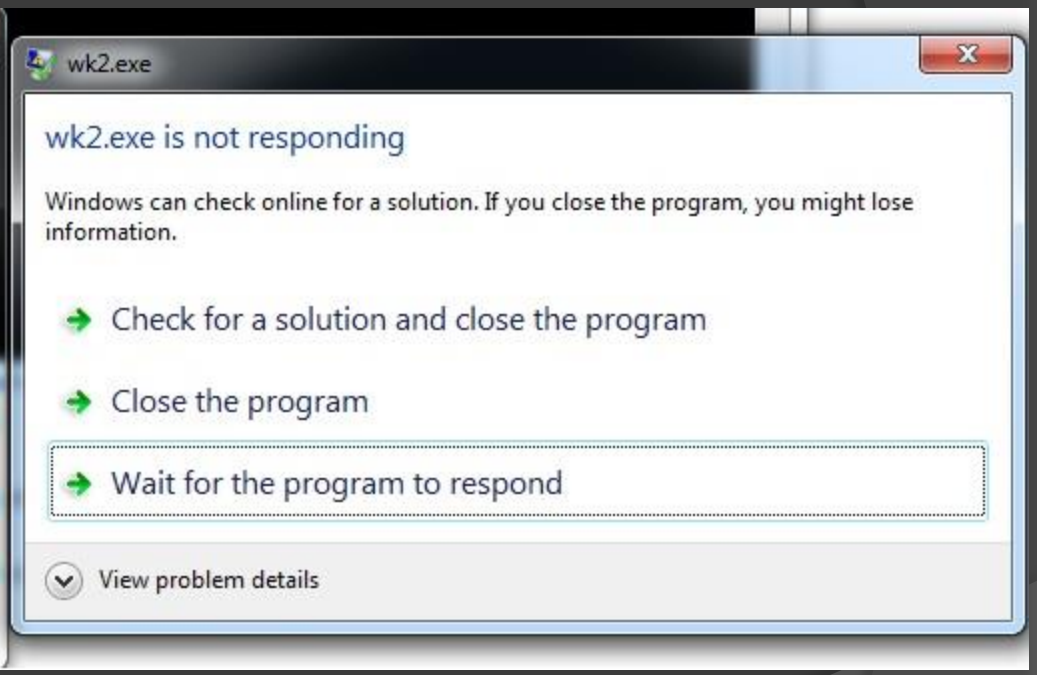
- The copy constructor also does shallow copy by default

Overloading Copy Constructor

```
class pType {
  public:
    ...

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

      return *this;
    }

    pType (pType & tp) {
      ptr = new int;
      *ptr = *(tp.ptr);
    }
}
```

Recall the issue last week





Consider these code...

```
class APoint {
  public:
    ...
    int * ptr;
    APoint& operator=(APoint &p);

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

```
void main(void)
{
    APoint p1(1,2,3);
    APoint p2(4,5,6);
    APoint p3;

    p3 = p2 + p1;

    cout << p3;
}
```

```
APoint& APoint::operator=(APoint &p)
{
    if (this != &p)
    {
        x = p.x;
        y = p.y;
        *ptr = *(p.ptr);
    }
    return *this;
}
```

```
APoint operator+(APoint &pt1, APoint &pt2)
{
    APoint temp;
    temp.x = pt1.x + pt2.x;
    temp.y = pt1.y + pt2.y;
    *(temp.ptr) = *(pt1.ptr) + *(pt2.ptr);

    return temp;
}

APoint::~APoint(void)
{
    delete ptr;
}
```

Solution?

```
class APoint {
  public:
    ...

    APoint(APoint &);

    ...
}
```

```
APoint::APoint(APoint & copy)
{
    x = copy.x;
    y = copy.y;
    ptr = new int;
    *ptr = *(copy.ptr);
}
```

Overloading Unary Operator

- These unary operators operate on a single operand:
  - The increment (++) and decrement (--) operators.
  - The unary minus (-) operator.
  - The logical not (!) operator.

Overloading Unary Operator

- We add another functionality to the timeType class

```
timeType t1 (3, 4, 59);    // 3hr 4min 59sec
t1++;                      // 3hr 5min 0sec
```

- Pre-increment is defined as

```
timeType operator++ (void);
```

- Post-increment is defined as

```
timeType operator++ (int ignore);
```

Overloading Unary Operator

- Pre-increment

```
class timeType {
  ...

  timeType operator++ (void) {
    timeType tt (0, 0, 1);
    (*this) = (*this) + tt;

    return (*this);
  }
}
```

```
timeType t1 (3, 4, 59);
cout << ++t1 << endl;      // 3hr 5min 0sec
cout << t1 << endl;        // 3hr 5min 0sec
```

Overloading Unary Operator

- Post-increment

```
class timeType {
  ...

  timeType operator++ (int ignore) {
    timeType returnTime = (*this);

    timeType tt (0, 0, 1);
    (*this) = (*this) + tt;

    return returnTime;
  }
}
```

```
timeType t1 (3, 4, 59);
cout << t1++ << endl;      // 3hr 4min 59sec
cout << t1 << endl;        // 3hr 5min 0sec
```

Overloading Unary Operator

- Compare Pre and Post Increment

Pre-increment

```
timeType operator++ (void) {
  timeType t1 (0, 0, 1);
  (*this) = (*this) + t1;

  return (*this);
}
```

Post-increment

```
timeType operator++ (int ignore) {
  timeType returnTime = (*this);

  timeType t1 (0, 0, 1);
  (*this) = (*this) + t1;

  return returnTime;
}
```

Overloading Unary Operator

- Member function

```
class timeType {
  ...

  timeType operator++ (void);
  timeType operator++ (int);
}
```

- Friend function

```
class timeType {
  ...

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

Overloading Unary Operator

- Member function

```
timeType timeType::operator++ (void) {
  timeType t1 (0, 0, 1);
  (*this) = (*this) + t1;

  return (*this);
}
```

- Friend function

```
timeType operator++ (timeType &tt) {
  timeType t1 (0, 0, 1);
  tt = tt + t1;

  return tt;
}
```

Overloading Unary Operator

- Member function

```
timeType timeType::operator++ (int ignore) {
  timeType returnTime = (*this);

  timeType t1 (0, 0, 1);
  (*this) = (*this) + t1;

  return returnTime;
}
```

- Friend function

```
timeType operator++ (timeType & tt, int ignore) {
  timeType returnTime = tt;

  timeType t1 (0, 0, 1);
  tt = tt + t1;

  return returnTime;
}
```

Overloading Unary Operator

- Pro
  - Convenient to use
  - Changes long and complex codes into short and simple codes.

```
String szValue = "";
szValue += "hello";
```

vs

```
String szValue = "";
szValue.concatenate( "hello");
```

- Con
  - Nil

Overloading Unary Operator

- We add another functionality to the timeType class

```
timeType operator-() const {
     timeType t;
     t.hr = -hr;
     t.min = -min;
     t.sec = -sec;
     return t;
  }
```

Overloading Unary Operator

- We add another functionality to the timeType class

```
bool operator !() const
{
     if (x < 10 && y < 20 && z < 30)
         return true;
     else
         return false;
}
```

Summary

- We had just discussed about,
  - Overloading Copy Constructor
  - Overloading Unary Operator