Lecture 18

Operator Overloading

Operator Overloading

Prof. Hyeong-Seok Ko
Seoul National University
Graphics & Media Lab

Contents

How many copies are made?

```
int f(int a) { a = a + 3; return a; }

int g(int& a) { a = a + 3; return a; }

int& h(int& a) { a = a + 3; return a; }

int& k(int& a) { int b; a = a + 3; b = a; return b; }

void main() {
    int u, v, w, p, x, y, z, q;
    x = y = z = q = 1;
    u = f(x); cout << x << "  " << u << endl;
    v = g(y); cout << y << "  " << v << endl;
    w = h(z); cout << z << "  " << w << endl;
    p = ++h(q); cout << q << "  " << p << endl;
}
```

Pass-through Object

Operator Overloading

- You can overload the following C++ operators with the following limitation: the original arity (unary, binary, etc.) must be observed. For example the operator– can be defined either unary or binary but not as ternary.

Operators which can be overloaded							
+	-	*	/	%	^	&	|
~	!	=	<	>	+=	-=	*=
/=	%=	^=	&=	|=	<<	>>	>>=
<<=	==	!=	<=	>=	&&	||	++
--	->*	,	->	[]	()	new	delete
new[]	delete[]						

Operators that cannot be overloaded				
.	.*	::	?:	sizeof

Basic Operations on Vec2 Class

- The following cause a number of compilation errors !

```
class Vec2 {
public :
   Vec2() : x(0), y(0) {}
   Vec2(float a, float b) : x(a), y(b) {}

   float x,y;
};

void main() {
   double d0 = 1, d1 = 2, d;
   Vec2 v0(0,0), v1(0,1), v;

   d = d0 + d1;                // OK
   v = v0 + v1;                // Compilation error !

   d += d0;                    // OK
   v += v0;                    // Compilation error !

   std::cout << d;             // OK
   std::cout << v;             // Compilation error !
}
```

How can we make this code work ?

```
Vec2 v0(0,0), v1(0,1), v;

v = v0 + v1;
v += v0;
std::cout << v;
```

Operator Overloading !

Non-member Operator Overloading

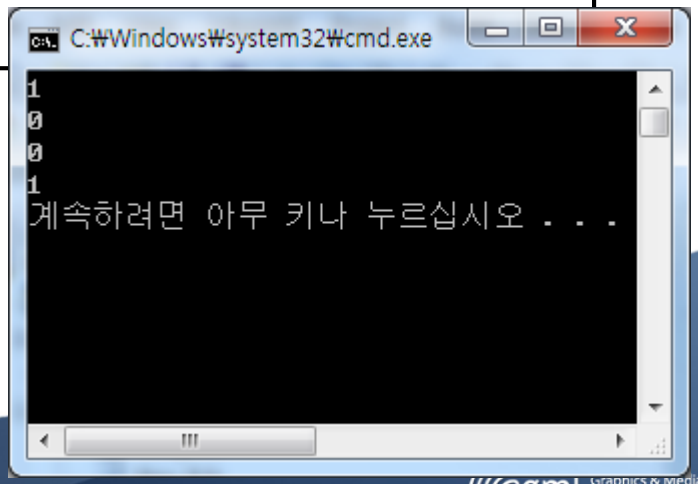
```
class vec2 {
public :
   vec2() : x(0), y(0) {}
   vec2(float a, float b) : x(a), y(b) {}

   float x,y;
};

inline vec2 operator+(const vec2& a, const vec2& b)
{ return vec2(a.x+b.x, a.y+b.y); }

void main() {
   vec2 a(1.1f,0), b(1.3f,2.5f);
   vec2 x = a + b;
}
```

Member Operator Overloading

```
class vec2 {
public :
   vec2() : x(0), y(0) {}
   vec2(float a, float b) : x(a), y(b) {}

   vec2 operator+(const vec2& a)
   { return vec2(x+a.x, y+a.y); }

   float x,y;
};

void main() {
   vec2 a(1.1f,0), b(1.3f,2.5f);
   vec2 x = a + b;
}
```

Relational Operator Overloading

```
inline bool operator==(const Vec2& a, const Vec2& b) {
        return a.x == b.x && a.y == b.y;
}
inline bool operator!=(const Vec2& a, const Vec2& b) {
        return !(a==b);
}
void main() {
    Vec2 v0(1.1f,0), v1(1.3f,2.5f);
    std::cout << (v0 == v0) << std::endl;
    std::cout << (v0 == v1) << std::endl;
    std::cout << (v0 != v0) << std::endl;
    std::cout << (v0 != v1) << std::endl;
}
```



Assignment Operator

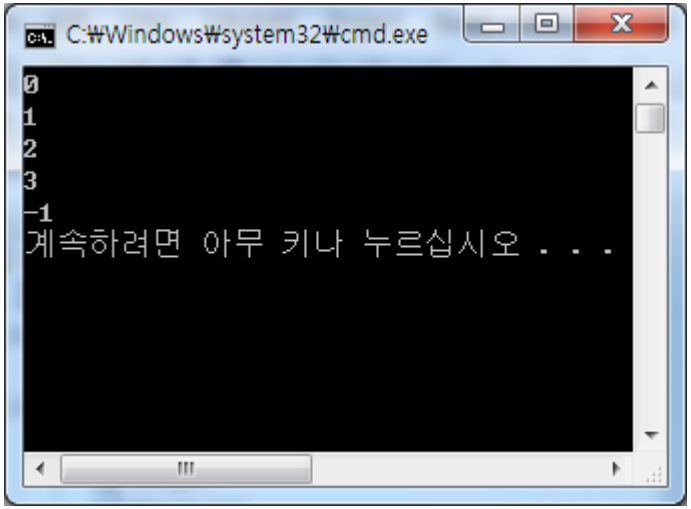
- See Lecture 12.

Prototypes for Operators

```
class X {
// members (with implicit 'this' pointer)
  X operator& (X);// binary  &  (and)
  X* operator& ();// prefix unary &
  X operator++ (int);        // operator is defined only between the same
                             // type. Therefore int here is an artificial
                             // argument, and indicates postfix version.
  X operator& (X,X);         // error: ternary
  x operator/ ();   // error: prefix unary
};

// global functions (often friends)
X operator- (X);   // prefix unary minus
X operator- (X,X);// binary minus
X operator-- (X&,int);       // postfix decrement
X operator- ();              // error: no operand
X operator- (X,X,X);         // error: ternary
X operator% (X); // error: unary %
```

Subscript Operator Overloading

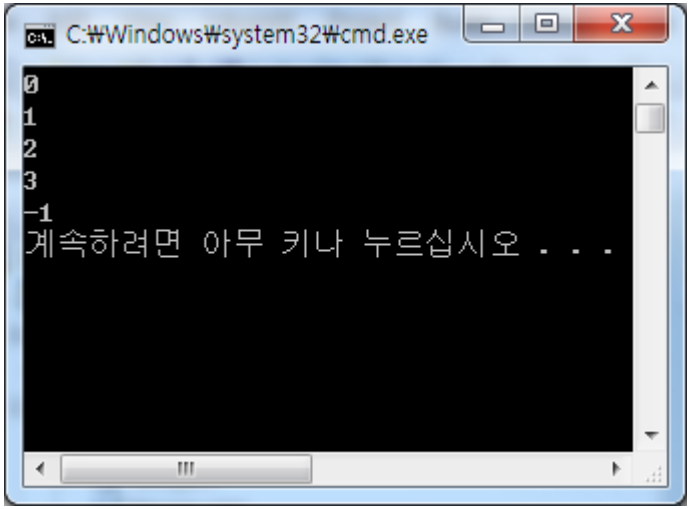
```
class Array {
public :
   Array(std::size_t num) : _size(num) { ptr = new int[num]; }
   Array(const Array& arr) : _size(arr._size) {
      ptr = new int[_size];
      for(std::size_t i=0;i<_size;++i)
         ptr[i] = arr.ptr[i];
   }
   ~Array() { if(ptr != NULL) delete [] ptr; }
   Array& operator=(const Array& arr) {
      if(ptr != NULL) delete [] ptr;
      _size = arr._size;
      ptr = new int[arr._size];
      for(std::size_t i=0;i<_size;++i)
         ptr[i] = arr.ptr[i];
      return (*this);
   }
   const std::size_t size() const { return _size; }
```




```
public :
   int *          ptr;
   std::size_t    _size;
};

void main() {
   Array a(5);
   *(a.ptr) = 0; *(a.ptr+1) = 1; a.ptr[2] = 2; *(a.ptr+3) = 3; a.ptr[4] = -1;
   for(std::size_t i=0;i<a.size();++i)
      std::cout << a.ptr[i] << std::endl;
}
```

Subscript Operator Overloading

```
class Array {
public :
   Array(std::size_t num) : _size(num) { ptr = new int[num]; }
   Array(const Array& arr) : _size(arr._size) {
      ptr = new int[_size];
      for(std::size_t i=0;i<_size;++i)
         ptr[i] = arr.ptr[i];
   }
   ~Array() { if(ptr != NULL) delete [] ptr; }
   Array& operator=(const Array& arr) {
      if(ptr != NULL) delete [] ptr;
      _size = arr._size;
      ptr = new int[arr._size];
      for(std::size_t i=0;i<_size;++i)
         ptr[i] = arr.ptr[i];
      return (*this);
   }
   const std::size_t size() const { return _size; }
```




```
public :
   int *          ptr;
   std::size_t    _size;
};

void main() {
   Array a(5);
   a[0] = 0; a[1] = 1; a[2] = 2; a[3] = 3; a[4] = -1;
   for(std::size_t i=0;i<a.size();++i)
      std::cout << a[i] << std::endl;
}
```

Subscript Operator Overloading

```
class Array {
public :
   Array(std::size_t num) : _size(num) { ptr = new int[num]; }
   Array(const Array& arr) : _size(arr._size) {
      ptr = new int[_size];
      for(std::size_t i=0;i<_size;++i)
         ptr[i] = arr.ptr[i];
   }
   ~Array() { if(ptr != NULL) delete [] ptr; }
   Array& operator=(const Array& arr) {
      if(ptr != NULL) delete [] ptr;
      _size = arr._size;
      ptr = new int[arr._size];
      for(std::size_t i=0;i<_size;++i)
         ptr[i] = arr.ptr[i];
      return (*this);
   }
   const std::size_t size() const { return _size; }

   int& operator[](const std::size_t i) { return ptr[i]; }
   // int operator[](const std::size_t i) const { return ptr[i]; }

public :
   int *          ptr;
   std::size_t    _size;
};

void main() {
   Array a(5);
   a[0] = 0; a[1] = 1; a[2] = 2; a[3] = 3; a[4] = -1;
   for(std::size_t i=0;i<a.size();++i)
      std::cout << a[i] << std::endl;
}
```