Chapter 10

Operator Overloading; String and Array Objects

OBJECTIVES

- What operator overloading is and how it makes programs more readable and programming more convenient.
- To redefine (overload) operators to work with objects of user-defined classes.
- The differences between overloading unary and binary operators.
- To convert objects from one class to another class.

Topics

- **10.1 Introduction**
- 10.2 Fundamentals & Restrictions
- 10.3 Operator Functions as Class Members vs. Global Functions
- 10.4 Overloading Stream Insertion and Stream Extraction Operators
- 10.5 Overloading Unary Operators
- 10.6 Overloading Binary Operators
- 10.7 Case Study: String Class

10.1 Introduction

- cout << int_variable; // 整型变量
  cout << ptrInt; // 整型指针
  cout << ptrChar; // 字符指针
- int num = 10; num = num + 1;
- int *pNum = new int[10];
  pNum = pNum + 1;
- String s1(“happy ”), s2(“birthday”);
  s1+=s1; s1=s1+s2;
  cout<<s1;

10.1 Introduction

- Date date1(1, 30, 2011);

  date1 = date1 + 1; // date1++;

  cout << date1; // 如何实现?
- HugeInt HugeintA, HugeintB;

  HugeintA + HugeintB; // 如何实现?
- operator overloading 运算符重载

10.1 Introduction

- C++语言为了支持基本数据类型数据运算, 内置了多种运算符, 并且其中部分已针对操作数类型的不同进行了重载;
- 当需要将这些运算符用于用户自定义类型时, 用户可以(大部分情况下必须)进行运算符重载.

Topics

- 10.1 Introduction
- **10.2 Fundamentals & Restrictions**
- 10.3 Operator Functions as Class Members vs. Global Functions
- 10.4 Overloading Stream Insertion and Stream Extraction Operators
- 10.5 Overloading Unary Operators
- 10.6 Overloading Binary Operators
- 10.7 Case Study: Array Class

10.2 Fundamentals & Restrictions—需求

- To use an operator on class objects, that operator must be overloaded with three exceptions:
  - *assignment* operator (=)
  - *address* (&) and *comma* (,) operators

- 目的: 提高类代码的可用、可读性
- HugeintA.add(HugeintB) vs HugeintA + HugeintB

10.2 Fundamentals & Restrictions—语法

- 运算符重载只是一种“语法上的方便”, 也就是说它只是另一种函数调用的方式. 区别:
  - 定义方式
  - 调用方式
- 定义重载的运算符(可视为特殊函数)就像定义函数(全局/成员), 区别是该函数的名称是

operator@

其中**operator**是关键词, **@**是被重载的运算符, 如:

```
HugeInt operator+(const HugeInt& a);
```

10.2 Fundamentals & Restrictions—语法

- 运算符重载只是一种“语法上的方便”，也就是说它只是另一种函数调用的方式. 区别:
  - 定义方式
  - 调用方式
- 普通函数
  - 全局函数: 函数名(参数列表)
  - 类成员函数: 对象.函数名(参数列表)等
- 重载的运算符: 使用时可以以表达式形式出现
  - **HugeIntA.operator+(HugeIntB)**
  - **HugeIntA + HugeIntB**

10.2 Fundamentals & Restrictions—限制

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

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

10.2 Fundamentals & Restrictions—限制

- 不能更改**Precedence**(优先级), **Associativity**(结合律) 以及**Number of Operands**(操作数数目)
- 仅能重载现有运算符, 不能创造新运算符
- 仅能重载应用于用户定义数据类型操作数的运算符
- • **int + int** X
- • **Hugeint + Hugeint** √
- • **Hugeint + int** √

Topics

- 10.1 Introduction
- 10.2 Fundamentals & Restrictions
- **10.3 Operator Functions as Class Members vs. Global Functions**
- 10.4 Overloading Stream Insertion and Stream Extraction Operators
- 10.5 Overloading Unary Operators
- 10.6 Overloading Binary Operators
- 10.7 Case Study: Array Class

10.3 Operator Functions as Class Members vs. Global Functions

- 选择一: 非静态的类成员函数

```
class HugeInt {
public:
    HugeInt operator+(const HugeInt& a);
}
```

- 选择二: 全局函数
  - Friend (访问私有数据)
  - Non-friend (不访问私有数据)

```
HugeInt operator+(const HugeInt& a, const HugeInt& b);
class HugeInt {
friend  HugeInt operator+(const HugeInt& a, const HugeInt& b);
}
```

10.3 Operator Functions as Class Members vs. Global Functions

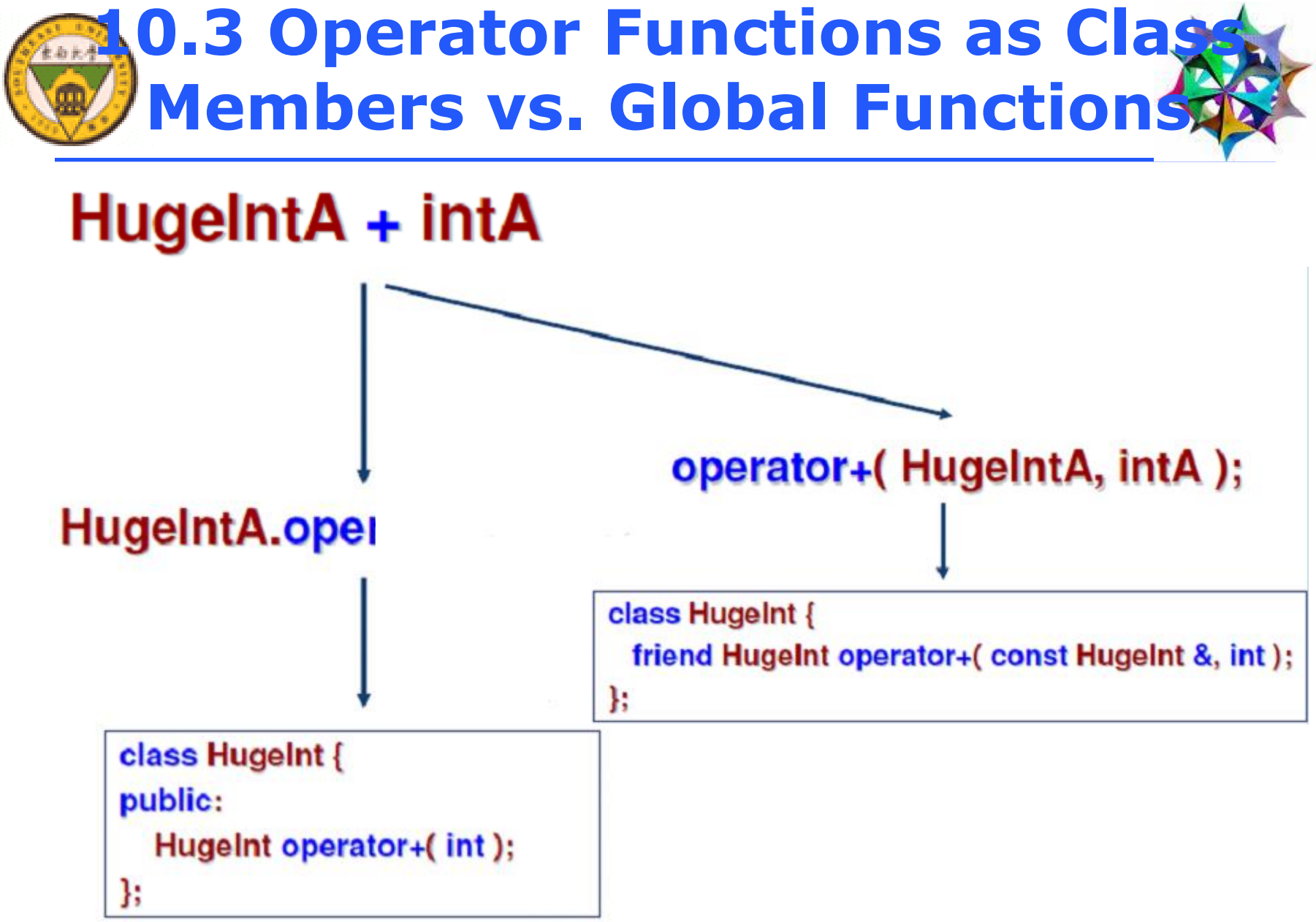
- 1. ( ), [ ], ->和赋值(=, +=, -=等)运算符必须重载为类的成员函数

- 2. 其余运算符可以选择重载为成员或全局函数

10.3 Operator Functions as Class Members vs. Global Functions

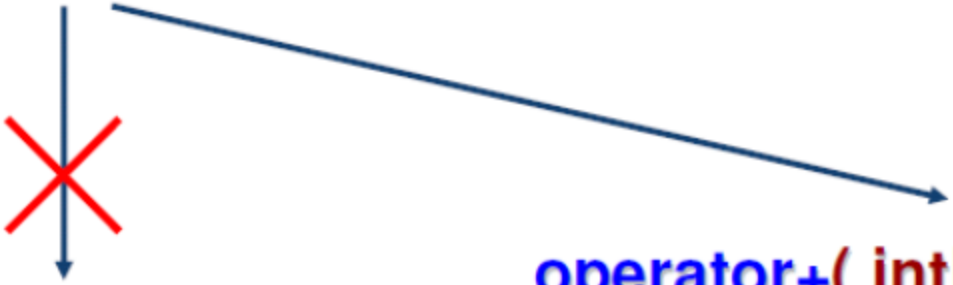
- 当重载为类的成员函数时
  - 将自动包含该类对象(或其引用)作为操作数，因此函数参数个数等于运算符目数-1
  - 并且, 左操作数(或唯一的操作数)必须为该类对象(或对象引用)
- 当重载为全局函数时
  - 函数参数个数等于运算符目数

10.3 Operator Functions as Class Members vs. Global Functions



10.3 Operator Functions as Class Members vs. Global Functions

- intB + HugeIntB



intB.operator+(HugeIntB);

operator+(intB, HugeIntB);

```
class HugeInt {
   friend HugeInt operator+( int, const HugeInt & );
};
```

Topics

- 10.1 Introduction
- 10.2 Fundamentals & Restrictions
- 10.3 Operator Functions as Class Members vs. Global Functions
- **10.4 Overloading Stream Insertion and Stream Extraction Operators**
- 10.5 Overloading Unary Operators
- 10.6 Overloading Binary Operators
- 10.7 Case Study: Array Class

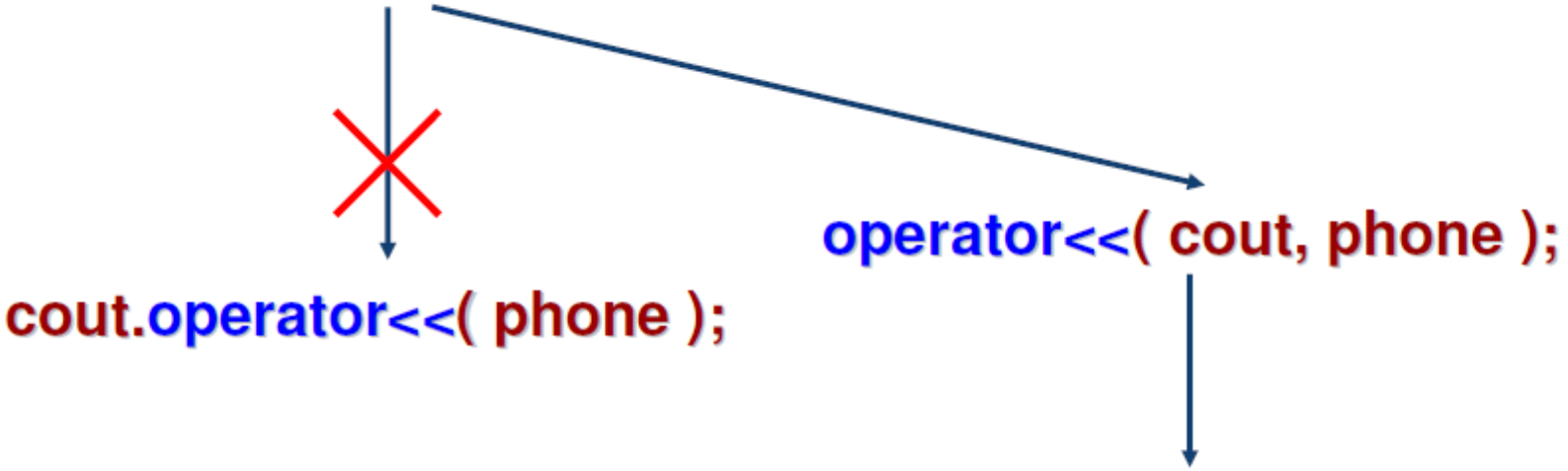
10.4 Overloading Stream Insertion and Stream Extraction Operators

- 需求:
- • cin >> phone; // (123) 456-7890
- • cout << phone; // (123) 456-7890

```
13 class PhoneNumber
14 {
17 private:
18    string areaCode; // 3-digit area code
19    string exchange; // 3-digit exchange
20    string line; // 4-digit line
21 }; // end class PhoneNumber
```

10.4 Overloading Stream Insertion and Stream Extraction Operators

- cout << phone;




```
class PhoneNumber{
  friend ostream &operator<<( ostream&, const PhoneNumber & );
};
```

10.4 Overloading Stream Insertion and Stream Extraction Operators

- cout << phone; // (123) 456-7890

```
12.  ostream &operator<<( ostream &output,
13.                       const PhoneNumber &number )
14.  {
15.     output << "(" << number.areaCode << ") "
16.            << number.exchange << "-" << number.line;
17.     return output; // enables cout << a << b << c;
18.  } // end function operator<<
```

10.4 Overloading Stream Insertion and Stream Extraction Operators

- cin >> phone; // (123) 456-7890

```
18  // overloaded stream extraction operator; cannot be
19  // a member function if we would like to invoke it with
20  // cin >> somePhoneNumber;
21  istream &operator>>( istream &input, PhoneNumber &number )
22  {
23     input.ignore(); // skip (
24     input >> setw( 3 ) >> number.areaCode; // input area code
25     input.ignore( 2 ); // skip ) and space
26     input >> setw( 3 ) >> number.exchange; // input exchange
27     input.ignore(); // skip dash (-)
28     input >> setw( 4 ) >> number.line; // input line
29     return input; // enables cin >> a >> b >> c;
30  } // end function operator>>
```

```
string s;
cin >> setw(n) >> s;   //指定读入n个字符赋值给s
```

Q & A

- 针对Date类, 分析如何实现>>， <<重载(成员vs全局, 函数原型).

- Date date1(1, 31, 2011);

  cout << date1; // January 31, 2011

  cin >> date1; // 2/4/2011

Topics

- 10.1 Introduction
- 10.2 Fundamentals & Restrictions
- 10.3 Operator Functions as Class Members vs. Global Functions
- 10.4 Overloading Stream Insertion and Stream Extraction Operators
- **10.5 Overloading Unary Operators**
- 10.6 Overloading Binary Operators
- 10.7 Case Study: Array Class

10.5 Overloading Unary Operators

- 一元运算符重载
- As a non-static member function with no arguments (无参数的非静态成员函数)
- As a global function with one argument (一个参数的全局函数)
  - Argument must be either an object of the class or a reference to an object of the class (参数必须是对象或者对象的引用)

10.5 Overloading Unary Operators

- IntegerSet s; // 设计!s 判断是否为空集合

Topics

- 10.1 Introduction
- 10.2 Fundamentals & Restrictions
- 10.3 Operator Functions as Class Members vs. Global Functions
- 10.4 Overloading Stream Insertion and Stream Extraction Operators
- 10.5 Overloading Unary Operators
- **10.6 Overloading Binary Operators**
- 10.7 Case Study: Array Class

10.6 Overloading Binary Operators

- 二元运算符重载
- as a non-static member function with one argument (一个参数的非静态成员函数)
- as a global function with two arguments (两个参数的全局函数)
  - one of those arguments must be either a class object or a reference to a class object (至少一个参数必须是对象或者对象的引用)

10.6 Overloading Binary Operators

- IntegerSet s1, s2;
- s1 == s2 判断集合s1是否等于s2

Q & A

- 针对Date类, 分析如何实现+=, ++重载(成员vs全局, 函数原型).

- Date date1(1, 31, 2011);

  date1 += 3;

  cout << date1++; // February 3, 2011

  cout << ++date1; // February 5, 2011

```
 9  class Date
10  {
11     friend std::ostream &operator<<( std::ostream &, const Date & );
12  public:
13     Date( int m = 1, int d = 1, int y = 1900 ); // default constructor
14     void setDate( int, int, int ); // set month, day, year
15     Date &operator++(); // prefix increment operator
16     Date operator++( int ); // postfix increment operator
17     Date &operator+=( unsigned int ); // add days, modify object
18     static bool leapYear( int ); // is date in a leap year?
19     bool endOfMonth( int ) const; // is date at the end of month?
20  private:
21     unsigned int month;
22     unsigned int day;
23     unsigned int year;
24
25     static const std::array< unsigned int, 13 > days; // days per month
26     void helpIncrement(); // utility function for incrementing date
27  }; // end class Date
```

```
            // overloaded output operator
            ostream &operator<<( ostream &output, const Date &d )
 9  const arr {
10     { 0, 3    static string monthName[ 13 ] = { "", "January", "February",
                     "March", "April", "May", "June", "July", "August",
                     "September", "October", "November", "December" };
                  output << monthName[ d.month ] << ' ' << d.day << ", " << d.year;
                  return output; // enables cascading
               } // end function operator<<
```

```
// overloaded prefix in
Date &Date::operator++(
{
   helpIncrement(); //
   return *this; // ref
} // end function opera
```

```
bool Date::endOfMonth( int testDay ) const
{
   if ( month == 2 && leapYear( year ) )
      return testDay == 29; // last day of Feb. in leap year
   else
      return testDay == days[ month ];
} // end function endOfMonth
```

```
Date Date::operator++(
{
   Date temp = *this;
   helpIncrement();

   // return unincreme
   return temp; // val
} // end function oper
```

```
void Date::helpIncrement()
{
   // day is not end of month
   if ( !endOfMonth( day ) )
      ++day; // increment day
   else
      if ( month < 12 ) // day is end of month and month < 12
      {
         ++month; // increment month
         day = 1; // first day of new month
      } // end if
      else // last day of year
      {
         ++year; // increment year
         month = 1; // first month of new year
         day = 1; // first day of new month
      } // end else
} // end function helpIncrement
```

```
// add specified numbe
Date &Date::operator+=
{
   for ( int i = 0; i
      helpIncrement();

   return *this; // er
} // end function oper
```

```
 9  class Date
10  {
11     friend std::ostream &operator<<( std::ostream &, const Date & );
12  public:
13     Date( int m = 1, int d = 1, int y = 1900 ); // default constructor
14     void setDate( int, int, int ); // set month, day, year
15     Date &operator++(); // prefix increment operator
16     Date operator++( int ); // postfix increment operator
17     Date &operator+=( unsigned int ); // add days, modify object
```

date in a leap year?
date at the end of month?

```
int main()
{
   Date d1( 12, 27, 2010 ); // December 27, 2010
   Date d2; // defaults to January 1, 1900

   cout << "d1 is " << d1 << "\nd2 is " << d2;
   cout << "\n\nd1 += 7 is " << ( d1 += 7 );

   d2.setDate( 2, 28, 2008 );
   cout << "\n\n  d2 is " << d2;
   cout << "\n++d2 is " << ++d2 << " (leap year allc

   Date d3( 7, 13, 2010 );

   cout << "\n\nTesting the prefix increment operato
      << "  d3 is " << d3 << endl;
   cout << "++d3 is " << ++d3 << endl;
   cout << "  d3 is " << d3;

   cout << "\n\nTesting the postfix increment operator:\n"
      << "  d3 is " << d3 << endl;
   cout << "d3++ is " << d3++ << endl;
   cout << "  d3 is " << d3 << endl;
} // end main
```

```
d1 is December 27, 2010
d2 is January 1, 1900

d1 += 7 is January 3, 2011

  d2 is February 28, 2008
++d2 is February 29, 2008 (leap year allows 29th)

Testing the prefix increment operator:
  d3 is July 13, 2010
++d3 is July 14, 2010
  d3 is July 14, 2010

Testing the postfix increment operator:
  d3 is July 14, 2010
d3++ is July 14, 2010
  d3 is July 15, 2010
```

Dynamic Memory Management with Operators new and delete

- **Motivation**
  - **char sentence[1000];**
  - **Fixed-size array.**
  - **1000?**
- **Dynamic memory management**
  - **根据需求分配(allocate)/释放(deallocate)内存**
  - **new / delete operator**

Dynamic Memory Management with Operators new and delete

- use the new operator to dynamically allocate the exact amount of memory required at execution time in the free store (sometimes called the heap堆)
- return the memory to the free store by using the delete operator to deallocate (i.e., release) the memory, which can then be reused by future new operations
- memory leak (内存泄露)

Dynamic Memory Management with Operators new and delete

- (1) 基本数据类型

```
1.  double *ptr = new double(3.14159);
2.  cout << ptr << endl;
3.  cout << *ptr << endl;
4.  delete ptr;
5.  cout << ptr << endl;
6.  ptr = 0;
```

Dynamic Memory Management with Operators new and delete

- (2) 类对象

```
1.  class Time{
2.  public:
3.      Time(){ cout << "Time constructor called.\n"; }
4.      ~Time(){ cout << "Time destructor called.\n"; }
5.  };
6.  int main()
7.  {
8.      Time *timePtr = new Time;
9.      delete timePtr;
10.     return 0;
11. }
```

```
Time constructor called.
Time destructor called.
```

Dynamic Memory Management with Operators new and delete

- **new 对象:**
  - **allocates storage of the proper size for an object**
  - **calls the default constructor to initialize the object**
  - **returns a pointer of the type specified to the right of the new operator**
- **delete 对象:**
  - **calls the destructor for the object to which pointer points**
  - **deallocates the memory associated with the object**

Dynamic Memory Management with Operators new and delete

- (2) 类对象

```
1.  class Time2{
2.  public:
3.      Time2( int, int, int);
4.      ~Time2();
5.  };
6.  int main()
7.  {
8.      Time2 *timePtr = new Time2( 12, 45, 0 );
9.      delete timePtr;
10.     return 0;
11. }
```

构造函数参数列表

Dynamic Memory Management with Operators new and delete

- (3) 数组– 基本数据类型

```
1. int size = 10;
2. int *gradesArray = new int[ size ];
3. delete [ ] gradesArray;
```

- 注意与Fixed size数组的区别:
- *Constant integral expression* vs *Any integral expression*

Topics

- 10.1 Introduction
- 10.2 Fundamentals & Restrictions
- 10.3 Operator Functions as Class Members vs. Global Functions
- 10.4 Overloading Stream Insertion and Stream Extraction Operators
- 10.5 Overloading Unary Operators
- 10.6 Overloading Binary Operators
- **10.7 Case Study: Array Class**

10.7 Case

Array Class 需求

- Array a1(3), a2;
- cout<<a1.getSize();
- cin>>a1>>a2;
- cout<<a1<<a2;
- if(a1!=a2) //if(a1==a2)
- a1[2]=12;
- cout<<a1[2];
- Array a3(a1); //Array a3=a1;
- a2=a1;

```
class Array
{
   friend ostream &operator<<( ostream &, const Array & );
   friend istream &operator>>( istream &, Array & );

public:
   explicit Array( int = 10 ); // default constructor
   ~Array(); // destructor
   size_t getSize() const; // return size

   bool operator==( const Array & ) const; // equality operator
   bool operator!=( const Array & ) const;

   int &operator[]( int );
   int operator[]( int ) const;

   Array( const Array & ); // copy constructor
   const Array &operator=( const Array & ); // assignment operator
private:
   size_t size; // pointer-based array size
   int *ptr; // pointer to first element of pointer-based array
}; // end class Array
```

10.7 Case

Array Class 需求

- Array a1(3), a2;
- cout<<a1.getSize();
- cin>>a1>>a2;
- cout<<a1<<a2;
- if(a1!=a2) //if(a1==a2)
- a1[2]=12;
- cout<<a1[2];
- Array a3(a1); //Array a3=a1;
- a2=a1;

```
class Array
{
   friend ostream &operator<<( ostream &, const Array & );
   friend istream &operator>>( istream &, Array & );

public:
   explicit Array( int = 10 );
   ~Array(); // destructor
   size_t getSize() const; //

   bool operator==( const Arra
   bool operator!=( const Arra

   int &operator[]( int );
   int operator[]( int ) const

   Array( const Array & ); //
   const Array &operator=( cor
private:
   size_t size; // pointer-bas
   int *ptr; // pointer to first element of pointer-based array
}; // end class Array
```

```
Array::Array(int arraySize)
{
    if(arraySize>0)
        size=arraySize;
    else
        size=10;
    ptr=new int[size];
    for(int i=0;i<size;i++)
        ptr[i]=0;
}
Array::~Array()
{
    delete []ptr;
}
```

10.7 Case

Array Class 需求

- Array a1(3), a2;
- cout<<a1.getSize();
- cin>>a1>>a2;
- cout<<a1<<a2;
- if(a1!=a2) //if(a1==a2)
- a1[2]=12;
- cout<<a1[2];
- Array a3(a1); //Array
- a2=a1;

```
class Array
{
   friend ostream &operator<<( ostream &, const Array & );
   friend istream &operator>>( istream &, Array & );

istream &operator>>( istream &input, Array &a )
{
    for ( size_t i = 0; i < a.size; ++i )
       input >> a.ptr[ i ];
    return input; // enables cin >> x >> y;
} // end function

ostream &operator<<( ostream &output, const Array &a )
{
   // output private ptr-based array
   for ( size_t i = 0; i < a.size; ++i )
   {
      output << setw( 12 ) << a.ptr[ i ];
      if ( ( i + 1 ) % 4 == 0 ) // 4 numbers per row of outp
         output << endl;
   } // end for
   if ( a.size % 4 != 0 ) // end last line of output
      output << endl;
   return output; // enables cout << x << y;
} // end function operator<<
```

10.7 Case

Array Class 需求

- Array a1(3), a2;
- cout<<a1.getSize();
- cin>>a1>>a2;
- cout<<a1<<a2;
- if(a1!=a2) //if(a1==a2)
- a1[2]=12;
- cout<<a1[2];
- Array a3(a1); //Array a
- a2=a1;

```
class Array
{
   friend ostream &operator<<( ostream &, const Array & );
   friend istream &operator>>( istream &, Array & );

public:
   explicit Array( int = 10 ); // default constructor
   ~Array(); // destructor
   size_t getSize() const; // return size

   bool operator==( const Array & ) const; // equality operator
   bool operator!=( const Array & ) const;
```

```
bool Array::operator==(const Array &right) const
{
    if(size!=right.size)
        return false;
    for(int i=0;i<size;i++)
        if(ptr[i]!=right.ptr[i])
            return false;
    return true;
}}

bool Array::operator!=( const Array &right ) const
{
   return !( *this == right);
} // end function operator!=
```

10.7 Case

Array Class 需求

- Array a1(3), a2;
- cout<<a1.getSize();
- cin>>a1>>a2;
- cout<<a1<<a2;
- if(a1!=a2) //if(a1==a2)
- a1[2]=12;
- cout<<a1[2];
- Array a3(a1); //Array
- a2=a1;

```
class Array
{
   friend ostream &operator<<( ostream &, const Array & );
   friend istream &operator>>( istream &, Array & );

public:
   explicit Array( int = 10 ); // default constructor
   ~Array(); // destructor
   size_t getSize() const; // return size

   bool operator==( const Array & ) const; // equality operator
   bool operator!=( const Array & ) const;

   int &operator[]( int );
   int operator[]( int ) const;

   Array( const Array & ); // copy constructor
   const Array &operator=( const Array & ); // assignment operator
private:
   size_t size; // pointer-based array size
   int *ptr; // pointer to first element of pointer-based array
}; // end class Array
```

```
int Array::operator[](int subscript) const
{
    return ptr[subscript];
}

int &Array::operator[](int subscript)
{
    return ptr[subscript];
}
```

10.7 Case Study: Array Class

- 拷贝构造函数Copy Constructor

```
Num b;
Num a = b; // 拷贝构造
a = b; // 赋值
```

10.7 Case Study: Array Class

```
class Num{
public:
  Num(){
    nums = new int[10];
    for( int i=0; i<10; i++) nums[i] = i;
  }
  void setvalue( int i, int v ){ nums[i] = v; }
  void print(){
    cout << nums << ":";
    for( int i=0; i<10; i++ )
      cout << nums[i] << " ";
    cout << endl;
  }
  ~Num(){ delete [] nums; }
private:
  int *nums;
};
```

```
int main()
{
  Num a;
  a.setvalue( 0, 100 );
  a.print();

  Num b = a;
  b.print();
  return 0;
};
```

```
00031090: 100 1 2 3 4 5 6 7 8 9
00031090: 100 1 2 3 4 5 6 7 8 9
```

10.7 Case Study: Array Class

```
class Num{
public:
    .............
    Num (const Num & n){
        nums = new int[10];
        for( int i=0; i<10; i++ )
            nums[i] = n.nums[i];
        cout << "Copy constructor called." << endl;
    }
    .............
};
```

1. 拷贝构造函数: 参数为同类对象引用的构造函数!

10.7 Case Study: Array Class

```
class Num{
public:
  Num(){
    nums = new int[10];
    for( int i=0; i<10; i++) nums[i] = i;
  }
  Num (const Num & n){
    nums = new int[10];
    for( int i=0; i<10; i++ )
      nums[i] = n.nums[i];
    cout << "Copy constructor called." << endl;
  }
  void setvalue( int i, int v ){ nums[i] = v; }
  void print(){
    cout << nums << ": ";
    for( int i=0; i<10; i++ )
      cout << nums[i] << " ";
    cout << endl;
  }
  ~Num(){ delete [] nums; }
private:
  int *nums;
};

int main()
{
  Num a;
  a.setvalue( 0, 100 );
  a.print();

  Num b = a;
  b.print();
  return 0;
};
```

```
00031090: 100 1 2 3 4 5 6 7 8 9
Copy constructor called.
00031168: 100 1 2 3 4 5 6 7 8 9
```

10.7 Case Study: Array Class

- 拷贝构造函数Copy Constructor, 何时被调用:
  - 传值方式传递对象参数
  - 函数返回对象
  - 使用同类对象来初始化对象

- 总结: 当类中含有需要动态分配内存的指针数据成员, 应提供拷贝构造函数并重载赋值运算符, 以避免缺省拷贝和赋值.

10.7 Case

Array Class 需求

- Array a1(3), a2;
- cout<<a1.getSize();
- cin>>a1>>a2;
- cout<<a1<<a2;
- if(a1!=a2) //if(a1==a2)
- a1[2]=12;
- cout<<a1[2];
- Array a3(a1); //Array
- a2=a1;

```
class Array
{
   friend ostream &operator<<( ostream &, const Array & );
   friend istream &operator>>( istream &, Array & );

public:
   explicit Array( int = 10 ); // default constructor
   ~Array(); // destructor
   size_t getSize() const; // return size

   bool operator==( const Array & ) const; // equality operator
   bool operator!=( const Array & ) const;

   int &operator[]( int );
   int operator[]( int ) const;

   Array( const Array & ); // copy constructor
   const Array &operator=( const Array & ); // assignment operator
private:
   size_t size; // pointer-based array size
   int *ptr; // pointer to first element of pointer-based array
}; // end class Array
```

```
Array::Array( const Array &arrayToCopy )
   : size( arrayToCopy.size ),
     ptr( new int[ size ] )
{
   for ( size_t i = 0; i < size; ++i )
      ptr[ i ] = arrayToCopy.ptr[ i ]; //
} // end Array copy constructor
```

10.7 Case

Array Class 需求

- Array a1(3), a2;
- cout<<a1.getSize();
- cin>>a1>>a2;
- cout<<a1<<a2;
- if(a1!=a2) //if(a1==a2)
- a1[2]=12;
- cout<<a1[2];
- Array a3(a1); //Array a3
- a2=a1;

```
class Array
{
   friend ostream &operator<<( ostream &, const Array & );
   friend istream &operator>>( istream &, Array & );

public:
   explicit Array( int = 10 ); // default constructor
   ~Array(); // destructor
   size_t getSize() const; // return size

   bool operator==( const Array & ) const; // equality operator
   bool operator!=( const Array & ) const;

   int &operator[]( int );
   int operator[]( int ) const;

   Array( const Array & ); // copy constructor
   const Array &operator=( const Array & ); // assignment operator
pr
};
```

```
const Array &Array::operator=( const Array &right )
{
   if ( &right != this ) // avoid self-assignment
   {
      if ( size != right.size )
      {
         delete [] ptr; // release space
         size = right.size; // resize this object
         ptr = new int[ size ];
      } // end inner if
      for ( size_t i = 0; i < size; ++i )
         ptr[ i ] = right.ptr[ i ];
   } // end outer if
   return *this; // enables x = y = z, for example
} // end function operator=
```

10.7 Case Study

需求

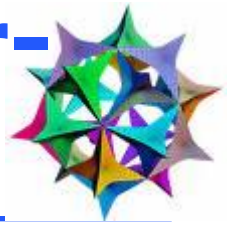
```
String s1("happy"),s2(" birthday");
cout<<"s1是: "<<s1<<",s2是: "<<s2<<endl;
cout<<boolalpha
<<"s1是否等于s2 "<<(s1==s2)<<endl
<<"s1是否不等于s2 "<<(s1!=s2)<<endl
<<"s1是否大于s2 "<<(s1>s2)<<endl
<<"s1是否小于s2 "<<(s1<s2)<<endl
<<"s1是否大于等于s2 "<<(s1>=s2)<<endl
<<"s1是否小于等于s2 "<<(s1<=s2)<<endl
<<"s1是否为空 "<<!s1<<endl;

cout<<"s1+=s2是: "<<(s1+=s2)<<endl;

s1+=" to you!";
cout<<s1<<endl;

cout<<"s1从0字符取长14的子串"<<s1(0,14)<<endl;
cout<<"s1从15字符取默认长度的子串"<<s1(15)<<endl;

String s3(s1);
cout<<"s3是: "<<s3<<endl;

s1[0]='H';
cout<<s1<<endl;
```

```
s1是: happy,s2是:  birthday
s1是否等于s2 false
s1是否不等于s2 true
s1是否大于s2 true
s1是否小于s2 false
s1是否大于等于s2 true
s1是否小于等于s2 false
s1是否为空 false
s1+=s2是: happy birthday
happy birthday to you!
s1从0字符取长14的子串happy birthday
s1从15字符取默认长度的子串to you!
s3是: happy birthday to you!
Happy birthday to you!
```

- **String Class**
  - **conversion constructor**
  - **copy constructor**

- 重载13个运算符<<, >>, =, +=, !, ==, <, !=, >, <=, >=, [ ], ()

8.7 Introduction to Pointer-Based String Processing



- char *strcpy(char *s1, const char *s2);
- char *strncpy(char *s1, const char *s2, size_t n);

- int strcmp(const char *s1, const char *s2);
- int strncmp(const char *s1, const char *s2, size_t n

- size_t strlen(const char *s);

Summary

- 哪些运算符可以重载? 何时需要重载? 有何限制? 如何重载?
- 成员函数vs 全局函数
- 拷贝构造函数和转换构造函数

Homework

- 实验必选题目：

  8、9、10、EX4

- 实验任选题目：

  11