CISC 2000 — Computer Science II
Fall, 2014

Note 11/13/2014

1 Review of operator overloading

(a) Lab class: take-away

```
############################
# Pass-by-value parameters #
############################

 Pass-by-value parameters
  are local variables. PERIOD.

   They are like those i's, j's, and temp's
       that appear and disappear when
         the function starts and finishes...

   (That is when constructor/destructor are secretly
        invoked to make and then kill them off ...)

   The one and only catch is:
    They are initialized ...
     with V. A. L. U. E. of arguments.

   (That's why it is the COPY constructor that is called.
       I hope there is no more secret!)

   and, these are ALL there is to know about
     Pass-by-value parameters
```

(b) Behavior of default assignment operator: Each memer variable is assigned in the manner appropriate to its type:

- if the member variable is of class type, the copy assignment operator for the class is used (as if by explicit qualification.
- if the member variable is an array, each element is assigned, in the manner appropriate to the element type;
- if the member variable is of built-in type, the built-in assignment operator is used.

(c) Big three: For class that uses dynamic variables, we need to overload assignment operator to perform **deep copy**, overload copy constructor, and implement destructor.

(d) Three different ways to pass parameters and return something from function.

- pass-by-value, return a value
- pass-by-pointer, return a pointer
- pass-by-reference, return a reference

```
class StaticIntArray
{
   public:
```

```
        ...
        // return reference to i-th element
        int & Element(int i)
        {
           if (i<0 || i>=length)
               cout <<"Out of bounds\n";
           else
                return array[i];
        }

        //overload element of operator ([])
        int & operator[] (int i)
        {
           if (i<0 || i>=length)
               cout <<"Out of bounds\n";
           else
                return array[i];
        }
     private:
        static const int CAPACITY=20;
        int array[CAPACITY];
        int length; // the actual length of filled part
};

// in main ...
StaticIntArray a;
a.Append(1);
a.Append(2);

a.Element(0) = 3;  //set array[0] to 3 ...
a.Element(4) = 10; // this will lead to error message: Out of bounds
     // A safer array to use !
a[1] = 4; // set array[1] of object a to 4...
// call the operator overloading [] with
//  a as the invoking object, 1 is the argument
// which returns a.array[1] by reference, allowing you to access that variable ...
```

2 string class

Please read Chapter 8.1, 8.2 on C-string, and string class.

Executive Summary.

(a) C-string is just an array of char, using terminating char to indicate the end. C standard library provides functions that work on them (strlen, strcpy, strcat, strcmp, ...). The **StringVar** example uses some of these functions.

Want to find out more? Run the following command in terminal: *man strcat*, which means show online manual for function strcat.

```
char greeting[20]={'H','E','L','L','O','\0','A','B'};
char anotherString[100];

cout <<greeting << endl; //only display HELLO
```

```
strcpy (greeting, anotherString); //might overflow the array greetings,
// depends on how long is the string stored in anotherString
// This is source for a lot of bugs!
```

(b) C++ standard string class is a much more sophisticated StringVar class (that you've studied), and standardized (i.e., you can expect same interface in all different implementation of the library). It hides the detail from you, for example, how large is the array used for the object, growing the array when needed...

Want to find out more: google C++ string !

```
string greetings ("Hello"); //constructor!
string name = "C++ Coder"; //constructor called, and assignment operator
// called
string banner = greetings+","+name; //we don't need to worry about overflowing
            // array used by banner object
// programmers who overload operator+ took care of that !
banner += "!!!";
banner = "!!!" + banner;

banner[0]='*'; // modify the first char in the string banner to *
               // i.e., string class has overload "element of" operator, []
cout <<banner<<endl; // this will display:
                     // *!!Hello,C++ Coder!!!
```

3 Stream

See slides.

4 stringstream class

(a) Use stringstream to output data to string We have seen earlier how we can use output operator to write something (anything) to a stringstream:

```
#include <iostream>
#include <sstream>
#include <fstream>
using namespace std;

void OutputTable (ostream & outs);

int main()
{
   cout <<"Write multiplication table to output stream\n";
   cout <<"f for disk file, t for terminal, and s for string stream\n";
   cout <<"Where do you want the output go (f/t/s)?";
   char output;
   cin >> output;

   ofstream file;
   stringstream sstream;
```

```
   switch (output) //we can only use switch on int type variable: char, int, long. Not string,
    //double, ...
   {
     case 't':
       OutputTable (cout);
       break;
     case 'f':
       file.open("output.txt");
       OutputTable (file);
       file.close();
       break;
     case 's':
       OutputTable(sstream);

       // we add divier lines to it...
       string Table="*******************************\n";
       Table = Table + sstream.str(); //str is a member function that returns the string
       Table = Table+"*******************************\n";

       cout <<"Here is the table\n";
       cout <<Table;
       break;
   }
}

void OutputTable (ostream & outs)
{
   for (int i=1;i<10;i++)
   {
       for (int j=1;j<10;j++)
           outs << i*j <<" ";
       outs << endl;
   }
}
```

(b) Use stringstream to extract from a string

```
/* Set the invoking object's value from user input */
istream operator>> (istream & ins, rational & result)
{
   string str;

   char c;
   int top, bottom;
   bool slash=false;
   bool decimal=false;
   bool valid = true;
   int digit=0;

   int num,denom;

   //read from the ins, everything up to a space, or newline is read
```

```
   // into str object
   ins >> str;

   //check for validity, 12.34, or 2/45, or 24,
   // we assume the denominator cannot be negative
   for (int i=0;i<str.length() && valid;i++)
   {
         if (str[i]=='-' && i!=0)
         { //only first char can be '-'
             valid = false;
         }
         else if (str[i]=='/' || str[i]=='.')
         {
             // if we've seen them before, it's invalid input
             if (slash || decimal)
                valid = false;

             //remember we just see them
              if (str[i]=='/')
                 slash = true;
              else
                 decimal = true;
        } else if (isdigit(str[i]))
 {
             if (decimal) // keep a counter about number of digits after .
                  digit++;
         }
        else //any other character => invalid input
             valid = false;

    }

    if (!valid)
    {
ins.setstate (ios::failbit); //set failbit on ins
     return ins;
    }

   // Now we are ready to extract information from str
   // To do that, we create a stringstream object, which
   // allow us to call input operator on it (similar to
   // reading from cin, or an ifstream (a disk file)

   stringstream ss(str,stringstream::in);

   if (!slash && !decimal)
   {
        // this is the case when the input is an int
        ss>> result.numerator;
        result.denominator=1;
   } else {
```

```
      // extract int before and after the '/' or '.'

      ss >> top >> c >> bottom;

      if (c=='/')
      {
        // set the invoking object's numerator, denominator
        result.numerator = top;
        result.denominator = bottom;
      }
      else if (c=='.')
      {
        num = top;
        denom = 1;

        // shift the decimal point to the end of the
        // number ... digit tells us how many times we need to shift
        for (int i=0;i<digit;i++)
        {
             num = num * 10;
             denom = 10*denom;
        }
        num+=bottom;

        // finally set rational_number
        result.numerator = num;
        result.denominator = denom;
      }
   }

   //"normalize" invoking object so that denominator is positive,
   // and numerator and denominator is co-prime
   result.simplify();

   return ins;
}
```