CSCI 2170 **Lecture Notes (6)**
Overloading operator

- **Operator overloading**
  - Can not define new operators by overloading symbols that are not already operators in C++
  - One of the operator has to be a new type created, e.g. can not change the meaning of operator with both operands C++ defined types
  - Can not change the standard precedence of a C++ operator
  - Can overload any C++ operator except for ., *, ::, ?:, sizeof
  - A typical class should overload at least the following operators: =, ==, !=, <, <=, > >=

- **Example 1: Operator not part of a class**

```
struct  cardStruct
{
    suitType   suit;
    int value;
    int points;
};

bool operator < (const cardStruct&  c1, const cardStruct & c2);
int main()
{
      cardStruct     card1, card2;

      card1.suit = DIAMOND;
      card1.suit = DIAMOND;

      card1.value = 12;
      card2.value = 5;

       if (card1 < card2)
            cout << "card2 is current leader" ;
       else
            cout << "card 1 is still the leader";
       return 0;
}

// define a new meaning of "<" for card struct type data
bool   operator < (const cardStruct & c1, const cardStruct & c2)
{
      if   ((c1.suit < c2.suit) || (c1.suit == c2.suit)
          && (c1.value < c2.value))
            return true;
      else
            return false;
}
```

- **Exampe 2: Add overloaded operators to ADT Time**
  - Operator ==, !=, +, <

- **Example 3: Add overloaded operators to ADT listADT List with overloaded constructors and overloaded operators**

Header file

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

typedef int  ListItemType;

#ifndef ListClass_H
#define ListClass_H

const int MAX_LIST = 200;
class ListClass
{
  public:
     ListClass();

     // copy constructor
     ListClass(const ListClass& anotherList);

     void ListInsert(int index, ListItemType
          value, bool& success);

     // these three are overloaded operators
     bool operator==(const ListClass & rhs);
     ListClass operator+(const ListClass &rhs);
     ListClass& operator=(const ListClass &rhs);
     // overloading assignment operator

     void PrintList();
     int  ListLength();

  private:
     ListItemType Items[MAX_LIST];
     int size;

     int translate(int index);
};

#endif
```

Implementation file

```
#include "listclass.h"

void ListClass::PrintList()
{
  for (int i=0; i<size; i++)
    cout << Items[i] << "  ";
  cout << endl;
}

bool ListClass::operator==(const ListClass &
rhs)
{
  bool success;
  if (size != rhs.size)
   return false;
  else
  {
     for (int i=0; i<size; i++)
     {
        if (Items[i] != rhs.Items[i])
           return false;
     }
   }

  return true;
}

ListClass& ListClass::operator=(const
ListClass&rhs)
{
  size = rhs.size;
  for (int i=0; i<rhs.size; i++)
    Items[i] = rhs.Items[i];

  return *this;
}

/* this definition of operator + adds the
corresponding values in the Items array
in the two listclass objects */
ListClass ListClass::operator+(const ListClass&
rhs)
{
   // use copy constructor to create tmpList
   // to have the same content as lrs object
   ListClass tmpList(*this);

   // rhs object list is longer
   if (tmpList.size < rhs.size)
   {
      int i;
      for (i=0; i<tmpList.size; i++)
      {
         // add corresponding items
         tmpList.Items[i] += rhs.Items[i];
      }
      // copy the rest of the items over to
      // tmpList object
      for (int j=i; j<rhs.size; j++)
         tmpList.Items[i] = rhs.Items[i];
   }
   else   // lhs object list is longer
   {
      int  i;
      for (i=0; i<rhs.size; i++)
```

```
        tmpList.Items[i] += rhs.Items[i];
    }

   return (tmpList);
}

/* implementation version 2: for operator +:
   no assuming the copy constructor  */
ListClass ListClass::operator+(const ListClass&
rhs)
{
  ListClass tmpList;

   /* copy everything from lhs object to
     tmpList object */
   for (int i=0; i<size; i++)
        tmpList.Items[i] = Items[i];
   tmpList.size = size;

   // the rest of the implementation is
  // the same as the first implementation
   ….
}

/* alternative version of operator +:
   concatenate two lists */
ListClass ListClass::operator + (const
ListClass& rhs)
 {
      // tmpList is a copy of lhs object
      ListClass  tmpList(*this);

      // append rhs items to the end of tmp
      for (int i=size, int j=0; ((i<MAX_LIST) &&
            (i<rhs.size )); i++, j++)
      {
             tmpList.AddItems(i+1, rhs.Items[j],
success);
      }

       tmpList.size = i;

       return tmpList;
}


void List::insert(int index,
                  ListItemType newItem,
                  bool& success)
{
  success = bool( (index >= 1) &&
              (index <= size+1) &&
              (size < MAX_LIST) );

  if (success)
  {  // make room for new item by
     // shifting all items at
     // positions >= index toward the end of the
     // list (no shift if index == size+1)
     for (int pos = size; pos >= index; --pos)
       items[translate(pos+1)] =
            items[translate(pos)];

     // insert new item
     items[translate(index)] = newItem;
     ++size; // increase the size of the list by one
  } // end if
} // end insert
```

Client program

```
#include "listclass.h"

// this is not a member function of listClass
// operator can be overloaded with new meaning
// from within a client program
bool operator<(ListClass &l1, ListClass &l2)
{
  return (l1.ListLength() < l2.ListLength());
}


int main()
{
  bool  success;
  ListClass l1, l2;

  l1.ListInsert(1, 20, success);
  l2.ListInsert(1, 20, success);

  l1.ListInsert(2, 30, success);
  l2.ListInsert(2, 30, success);

  l1.ListInsert(3, 15, success);

  if (l1<l2)
   cout << "Wrong < operator" << endl;
  else
   cout << "correct < operator " << endl;

  l1.PrintList();
  l2.PrintList();

  if (l1 == l2)
   cout << "correct output" << endl;

  ListClass l3;
  l3 = l1;
  l3.PrintList();

  if (l2==l3)
   cout << "2nd correct" << endl;
```

```
  ListClass l4;
  l4 = l1+l3;
  l4.PrintList();

  if (l3 == l4)
    cout << "Wrong answer" << endl;
  else
    cout << "3nd correct" << endl;

   return 0;
}
```

- **Overload << and >> for ostream and istream in ListClass**

Changes in the header file

```
class ListClass
{
    public:
        …
          < other member functions discussed before>
          …
        friend ostream &  operator << (ostream & os,    const ListClass & rhs);
        friend istream  &  operator >> (istream & is,  ListClass & rhs);
    private:
         ListItemType   Items[MAX_LIST];
         int      size;
};
```

Changes in the implementation file:

```
Add:
ostream &  operator << (ostream & os, const ListClass & rhs)
{
         if (rhs.size > 0)
         {
                  os << “There are “ << rhs.size << “ items in the list: “ << endl;
                  for (int i=0; i<rhs.size; i++)
                           os << rhs.Items[i] << endl;
         }
         else
                  os << “The list is empty.”<< endl;

         return os;
}

// append items to the end of list
istream & operator >> (istream & is,  ListClass & rhs)
{
         int number;

         cout << “how many items to add?” << endl;
```

```
        is >> number;


        for (int i=0; i<number; i++)
        {
                cout << "Enter item " << i+1 << " : " << endl;
                is >> rhs.Items[i+rhs.size];
        }

        rhs.size += number;
        return is;
}
```

client program:

```
#include "ListClass.h"
#include <iostream>
using namespace std;

int main( )
{
        ListClass L1;

        L1.Insert(1, 20, success);
        L1.Insert(2, 10, success);

        cin >> L1;
        cout << L1;
}
```