CSCI 2170 **Lecture Notes (5)**
Overloaded function, Polymorphism

- **Overloading regular function**

1. overloading function – having multiple meanings assigned to the same function name -- function or operators being assigned with multiple meanings is referred to as ***polymorphism*** (many forms).
2. which function will actually be used during program execution depends on the context of the program, e.g., parameters used in function call. The actual meaning of the function is decided during Compile/link time.

For example, in the following example, the function *ComputeSum* has three meanings:

```
int ComputeSum(int, int);
float ComputeSum(float, float);
int  ComputeSum(int [], int);

const int SIZE=50;
int main()
{
    int     x1=3, x2=4;
    int     array[SIZE];
    float           v1=4.5, v2=2.3;

    for (int i=0; i<SIZE; i++)
    {
                    cout << "Enter value " << i+1 << endl;
                    cin >>  array[i];
    }

    cout << "Sum of two integers is: " << ComputeSum(x1, x2) << endl;
    cout << "Sum of two float is: " << ComputeSum(v1, v2) << endl;
    cout << "Sum of the " << SIZE << " integers is "
         << ComputeSum(array, SIZE) << endl;
}

int ComputeSum(int  a, int b)
{     return a+b;
}

float ComputeSum(float a, float b)
{     return a+b;
}

int ComputeSum(int A[], int size)
{
          int sum=0;
          for (int i=0; i<size; i++)
                    sum += A[i];
          return sum;
}
```

- **Overload functions/methods of a class**
  - In a class, multiple methods may be of the same name. Each has a different parameter set, for example, (1) constructor: default constructor, copy constructor, other constructors; (2) method: printList: print list of all club members, print all members of a particular age range, print all members in "Nashville"
  - overloaded functions in TimeClass
    - print – print in military form
    - print – print in regular form
    - SetTime – by read new time from user
    - SetTime – by setting time to be the same as another time
  - Example: ADT list with multiple constructors and overloaded methods

```
/*****************************************
// Header file List.h for the ADT list
// Array-based implementation
//
// This example illustrates the overloaded constructor
and
// methods of a class
//*****************************************
const int MAX_LIST =200;
struct ClubMemberType
{
        string name;
        int    age;
        string  city;
        string  phone;
};
typedef ClubMemberType ListItemType;

class List
{
public:
  List(); // default constructor
      // destructor is supplied by compiler

  List(ifstream & infile);

  void ReadList(ifstream &infile);

  // copy constructor
  List(const List & aList);

  // list operations:
  // print all members
  void printList();

  // print list members of a particular city
  void printList(string city);

  // print list members of a particular age
  void  printList(int age);

  bool isEmpty() const;
  // Determines whether a list is empty.
  // Precondition: None.
  // Postcondition: Returns true if the list is empty;
  // otherwise returns false.

  int getLength() const;
  // Determines the length of a list.
  // Precondition: None.
  // Postcondition: Returns the number of items
  // that are currently in the list.

  void insert(int index, ListItemType newItem,
        bool& success);
  // Inserts an item into the list at position index.
  // Precondition: index indicates the position at which
  // the item should be inserted in the list.
  // Postcondition: If insertion is successful, newItem
is
  // at position index in the list, and other items are
  // renumbered accordingly, and success is true;
  // otherwise success is false.
  // Note: Insertion will not be successful if
  // index < 1 or index > getLength()+1.

  void remove(int index, bool& success);
  // Deletes an item from the list at a given position.
  // Precondition: index indicates where the deletion
  // should occur.
  // Postcondition: If 1 <= index <= getLength(),
  // the item at position index in the list is
  // deleted, other items are renumbered accordingly,
  // and success is true; otherwise success is false.

  void retrieve(int index, ListItemType& dataItem,
        bool& success) const;
  // Retrieves a list item by position.
  // Precondition: index is the number of the item to
  // be retrieved.
  // Postcondition: If 1 <= index <= getLength(),
  // dataItem is the value of the desired item and
```

```
  // success is true; otherwise success is false.


private:
  ListItemType items[MAX_LIST];
                          // array of list items
  int         size;       // number of items in list

  int translate(int index) const;
  // Converts the position of an item in a list to the
  // correct index within its array representation.
}; // end List class
// End of header file.


//*****************************************
// Implementation file List.cpp for the ADT list
// Array-based implementation
//*****************************************
#include "List.h" //header file

List::List() : size(0)
{
} // end default constructor

List::List(ifstream & infile)
{
    ReadList(infile);
}

List::List(const List& aList)
{
     size = aList.size;
    for (int i=0; i<size; i++)
    {
          items[i] = rhs.items[i];
    }
}

void List::ReadList(ifstream & infile)
{
   size = 0;

   ClubMemberType tmp;
    while (infile.peek() != EOF)
    {
          getline(infile, tmp.name);
          infile >> tmp.age;
          getline(infile, tmp.city);
          infile.ignore(100, '\n');
          getline(infile, tmp.phone);

         items[size] = tmp;
         size++;
      }
}

// print all members
void List::printList()
{
         for (int i=0; i<size; i++)
             cout << items[i].name << endl;
}

// print list members of a particular city
void List::printList(string city)
{
         for (int i=0; i<size; i++)
            if (items[i].city == city)
                  cout << items[i].name << endl;
}

// print list members of a particular age
void  List::printList(int age1, int age2)
{
        for (int i=0; i<size; i++)
             if (items[i].age
<=age2)&&(items[i].age>=age1)
          cout << items[i].name << endl;
}

}
bool List::isEmpty() const
{
  return bool(size == 0);
} // end isEmpty


int List::getLength() const
{
  return size;
} // end getLength


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

void List::remove(int index, bool& success)
{
  success = bool( (index >= 1) && (index <= size) );

  if (success)
  {  // delete item by shifting all items at positions >
    // index toward the beginning of the list
    // (no shift if index == size)
    for (int fromPosition = index+1;
          fromPosition <= size; ++fromPosition)
      items[translate(fromPosition-1)] =
```

```
                        items[translate(fromPosition)];
    --size; // decrease the size of the list by one
  } // end if
} // end remove

void List::retrieve(int index, ListItemType& dataItem,
                    bool& success) const
{
  success = bool( (index >= 1) &&
                  (index <= size) );

  if (success)
    dataItem = items[translate(index)];
} // end retrieve

int List::translate(int index) const
{
  return index-1;
} // end translate
// End of implementation file.


//*****************************************
****
//   Client Program using ADT list
//*****************************************
****
#include “List.h”
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
   // declare aList of “List” type
    List                aList;
    ListItemType    item;
    bool                success;
    ifstream            myIn;

     myIn.open(“datafile”);
     List       bList(myIn);  // constructor with file
stream

     List       cList;  // default constructor
     cList.ReadList(myIn);

    for (int i=1; i<=MAX_LIST; i++)
    {
          cout << “Enter list item “ << i << endl;
          cin >> item;
          aList.insert(i,  item, success);
     }

     List   dList(aList);   // use copy constructor

    aList.printList();
    bList.printList(“Nashville”);
    cList.printList(18, 22);

    return 0;
}
```