Mid#2 Handout A (by Steven Li)

list.h

```
#ifndef LISTH
#define LISTH

#include "state.h"

class LinkedList;

class ListNode{
  StateCodeLong data;
  ListNode *prev, *next;
  ListNode(const StateCodeLong &d, ListNode *p, ListNode *n):data(d), prev(p),
next(n){}
  friend class LinkedList;
};

class LinkedList{
  ListNode *head;
  int size;
public:
  LinkedList():head(NULL), size(0){}
  int getSize() const {return size;}
  void read(const char* filename);
  const StateCodeLong& operator[] (int index) const;
};

#endif
```

list.cpp

```
#include "list.h"
#include "state.h"
#include <iostream>
#include <fstream>

void LinkedList::read(const char* filename){
  ifstream inf(filename); //, ios::in|ios::binary);
  StateCode *sc = new StateCode();
  StateCodeLong *scl;

  while (sc->read(inf))
  {
    ListNode *ptr, *prev = NULL;

    for (ptr = head; ptr && ptr->data < *sc; ptr = ptr->next)
      prev = ptr;

    if (ptr && ptr->data == *sc)
      ptr->data += *sc;
    else
      if (prev && prev->data == *sc)
        prev->data += *sc;
      else
        if (prev)
        {
          size++;
          scl = new StateCodeLong(*sc);
          prev->next = new ListNode(*scl, prev, ptr);
          if (ptr)
            ptr->prev = prev->next;
        }
        else
        {
          size++;
          scl = new StateCodeLong(*sc);
          head = new ListNode(*scl, NULL, ptr);

          if (ptr)
            ptr->prev = head;
        }
    sc = new StateCode();
  }
}
```

```
const StateCodeLong& LinkedList::operator[] (int index) const
{
  ListNode *ptr = head;

  for (int i=0; i < index; i++)
    ptr = ptr->next;

  return ptr->data;
}
```

state.h

```
#ifndef __STATE_H__
#define __STATE_H__

#include <iostream>
#include <fstream>
#include <cstring>

using namespace std;

class StateCode {
  char state[3];
protected:
  short areaCode;
public:
  StateCode(){}

  StateCode(const StateCode &sc)
  {
    strcpy(state, sc.state);
    areaCode = sc.areaCode;
  }

  const char* getState() const {return state;};

  istream& read(istream &is)
  {
    is >> state >> areaCode;
    return is;
  }

  short getAreaCode() const {return areaCode;};
  bool operator< (const StateCode &sc) const {return strcmp(state, sc.state) <
0;};
  bool operator== (const StateCode &sc) const {return strcmp(state, sc.state) ==
0;};

};

class StateCodeLong: public StateCode {
  short count;
  short codes[40];
```

```
public:
  StateCodeLong(const StateCode &sc):StateCode(sc){
    count = 0;
    codes[0] = sc.getAreaCode();
  }

  const StateCodeLong& operator+=(const StateCode &sc)
  {
    count++;
    codes[count] = sc.getAreaCode();
    return *this;
  }

  friend ofstream& operator<<(ofstream& os, const StateCodeLong &scl)
  {
    os << scl.getState();
    for (int i=0; i <= scl.count; i++)
      os << " " << scl.codes[i];
    os << endl;
    return os;
  }
};

#endif
```

main.cpp

```
#include "state.h"
#include "list.h"

#include <iostream>
#include <fstream>

int main(int argc, char* argv[])
{
    LinkedList stateList;
    stateList.read("AreaCodes.txt");
    ofstream outf("StateCodes5.txt");
    for(int i = 0; i < stateList.getSize(); i++)
        outf << stateList[i];
    return 0;
}
```

Mid#2 Handout B (by Steven Li)

date.h

```
//  date.h

#ifndef DATE_H
#define DATE_H

#include <iostream>
using namespace std;

class Date
{
  char month[10];
protected:
  int day;
public:
  Date(const char *m, int d);
  const char* getMonth() const;
  bool operator< (const Date& d) const;
  virtual void print(ostream& os) const;
}; // class Date


#endif // DATE_H
```

date.cpp

```
//  date.cpp

#include <iostream>
#include <iomanip>
#include <cstring>
#include "date.h"

using namespace std;

Date::Date(const char* m, int d):day(d)
{
  strcpy(month, m);
}; // Date constructor

const char* Date::getMonth() const
{
  return month;
} // getMonth()

int convertMonthtoInt(const char* month)
{
  if (!strcmp(month, "January")) return 1;
  else if (!strcmp(month, "February")) return 2;
  else if (!strcmp(month, "March")) return 3;
  else if (!strcmp(month, "April")) return 4;
  else if (!strcmp(month, "May")) return 5;
  else if (!strcmp(month, "June")) return 6;
  else if (!strcmp(month, "July")) return 7;
  else if (!strcmp(month, "August")) return 8;
  else if (!strcmp(month, "Septempber")) return 9;
  else if (!strcmp(month, "October")) return 10;
  else if (!strcmp(month, "November")) return 11;
  else return 12;
} // local function : convertMonthtoInt

bool Date::operator<(const Date& d) const
{
  if((convertMonthtoInt(month) < convertMonthtoInt(d.getMonth()))
     || (convertMonthtoInt(month) == convertMonthtoInt(d.getMonth()) && day <
d.day))
    return true;
  else
```

```
    return false;
} // opeator <

void Date::print(ostream& os) const
{
  os << setw(10) << getMonth() << setw(3) << day << endl;
} // print()
```

dateTime.h

```
//  DateTime.h

#ifndef DATETIME_H
#define DATETIME_H

#include <iostream>
#include <iomanip>
#include "date.h"

using namespace std;

class DateTime : public Date
{
  int hour;
  int minute;
public:
  DateTime(const char* mon, int dat, int hou, int min) : Date(mon, dat),
hour(hou), minute(min){};
  void print(ostream& os) const
  {
    os << right << setw(10) << getMonth() << setw(3) << day << setw(3) << hour <<
":" << setfill('0') << setw(2) << minute << setfill(' ') << endl;
  }; // print()
}; // class DateTime

#endif // DATETIME_H
```

list.h

```
//  list.h

#ifndef LIST_H
#define LIST_H

#include <iostream>
#include <fstream>
#include "date.h"

using namespace std;

class List;

class ListNode
{
  Date* data;
  ListNode* previous;
  ListNode* next;
  ListNode(Date* d, ListNode* p, ListNode* n);
  friend class List;
}; // class ListNode

class List
{
  ListNode* head;
public:
  List();
  void insert(Date * d); // sorted using Date < operator
  void save(const char* filename);
}; // class List

#endif // LIST_H
```

list.cpp

```
//  list.cpp

#include "list.h"

using namespace std;

ListNode::ListNode(Date* d, ListNode* p, ListNode* n) : data(d), previous(p),
next(n)
{
}; // ListNode constructor


List::List() : head(NULL)
{
}



void List::insert(Date* d)
{
  ListNode* ptr;
  ListNode* prev = NULL;

  for(ptr = head; ptr && *(ptr->data) < *d; ptr = ptr->next)
    prev = ptr;

  if(!prev)
  {
    head = new ListNode(d, prev, ptr);

    if(!ptr)
      ptr->previous = head;
  } // does not even enter the for loop
  else
  {
    prev->next = new ListNode(d, prev, ptr);

    if(!ptr)
      ptr->previous = prev->next;
  } // if it is in the middle of the for loop or even at the end of the loop
```

```
} // insert()


void List::save(const char* filename)
{
  ofstream outf(filename);

  for (ListNode* ptr = head; ptr; ptr = head)
  {
    ptr->data->print(outf);
    head = head->next;
    delete ptr;
  } // for loop

} // save
```

main.cpp

```
//  main.cpp

#include <iostream>
#include "list.h"
#include "date.h"
#include "DateTime.h"

using namespace std;

int main(int argc, char* argv[])
{
  List list;
  char month[10];
  int day, hour, minute;

  cout << "Event: ";
  cin >> month >> day >> hour >> minute;

  while (day > 0) {
    if (hour >= 0)
      list.insert((Date*) new DateTime(month, day, hour, minute));
    else
      list.insert(new Date(month, day));

    cout << "Event: ";
    cin >> month >> day >> hour >> minute;
```

```
  }

  list.save(argv[1]);
  return 0;
} // main()
```