CSCI 1730 - Programming Assignment 4 - 50 pts.
Due Date: Thursday, April 9, 2015

What you need to turn in:

- **Code listing:** A printed copy of your C++ code solution for each problem. (From CodeBlocks, save your code as a PDF and then print the PDF.) Remember to include your name on every page you turn in. Be sure to follow the "Code Style Guidelines" specified in the "Assignment Information and Guidelines" handout that was distributed on the first day of class (these guidelines are also available at the class web page).
- **Code files:** E-mail me a copy of your C++ code. Copy/paste all code into one e-mail message – please, do not send attachments. **Enter your name in the subject line of your email.**
- **Working in Pairs:** If you want to work with one other student in our class on this assignment, this is acceptable provided that both members of the pair make a contribution to the solution. If you decide to work in a pair, turn in only one copy of the solution – clearly identify the name of each pair member on everything that you turn in.
- **Late Assignments:** Assignments are due **by the end of class** on the specified due date (both the paper copies and the e-mail copies). Assignments turned in late will be assessed a 20% penalty per day late.

1. (20 pts.) Write a **class** `IntSet` for modeling a set that contains integers in the range 0 through 9, inclusive. Here are features that should be included in this class:

- Member data should include a private `bool` array of size 10; this array will represent the set – the i^{th} array element will be `true` whenever integer `i` is in the set and will be `false` whenever integer `i` is not in the set.
- Include a no-argument constructor that initializes a set to the "empty set," i.e., a set whose array representation contains all false values.
- Include the following public overloaded operators:
  - `+` to perform the union of two set (the union of sets A and B is the set that contains all elements of set A or set B, or both).
  - `*` to perform the intersection of two sets (the intersection of sets A and B is the set that contains all elements in *both* set A and set B.)
  - `-` to form the difference of two sets (the difference of sets A and B is the set containing those elements that are in A but not in B)
  - `+=` to add an integer into a set.
  - `-=` to delete an integer from a set.
  - `==` to determine if two sets are equal.
  - `!` to form the complement of a set (the complement of set *A*, denoted $\overline{A}$, is the set containing all the elements in the universal set that are not in A - the universal set for this problem is the set containing all integers between 0 and 9)
  - `<=` to determine if one set is a subset of another set (set *A* is a subset of set *B* means that for any element *x* in *A*, *x* is also an element of set *B*).
  - `<<` to display a set in roster notation (for example, {2, 3, 5, 9})

Requirements for the overloaded operators:

- The overloaded `+`, `*`, and `-` operators take one `IntSet` argument and should return the resulting `IntSet`, not modify the `IntSet` object that invokes them.
- The overloaded `+=` and `-=` operators take one `int` argument and return nothing – i.e., they should modify the `IntSet` object that invokes them. In addition, they should check for valid

integer input (in the range 0-9), or if an add-item is already in the set, or if a delete-item is not in the set. An error message for invalid input should be generated.

- The overloaded `!` operator takes no arguments and should return the resulting `IntSet` object, not modify the `IntSet` object that invokes it.
- The overloaded == and <= operators take one `IntSet` argument and should return a `bool` result (not display the result).

Then, write a **C++ program** that uses the new `IntSet` class. The program should allow the user to repeatedly select from these options:

- add numbers to a set
- remove numbers from a set
- form the union of two sets
- form the intersection of two sets
- form the difference of two sets
- determine if two sets are equal
- form the complement of an set
- determine if one set is a subset of another set
- display a set

The program should allow for up to six sets to be created during a given program run. Use any stand-alone functions you feel necessary.

Program hints:

- Use an array of six `IntSet` objects to represent the required six sets.
- Here is a useful function for allowing a user to enter a set given by a letter name (A-F), verify the input, and then return an integer (0-5) which can be used to access the array of six `IntSet` objects. Note – remember that `toupper` requires inclusion of `cctype`.

```
int selset()
{
   int iset;
   char set;
   do{
     cout << "set (A,B,C,D,E,F)? :";
     cin >> set;
     set = toupper(set);
     iset = set-'A';
     if (iset<0 || iset>5) cout << "Invalid - reenter\n";
   }while (iset<0 || iset>5);
   return iset;
}
```

To see how this program will work, download and run the executable version of my solution (a4-1.exe).

2. (15 pts.) Write a **C++ program** that repeatedly prompts for and reads an e-mail address and then determines and displays whether the address is valid or invalid. For purposes of this program, an e-mail address is valid if it satisfies these conditions:

- The address cannot contain blanks. For example, *ed seifert@minneapolis.edu* is not a valid e-mail address.
- The @ character must occur exactly once. For example, *baz.cs.dpu.edu* and *bar@cs@dpu* are not valid e-mail addresses.
- The @ character cannot be the first character of the address. For example, *@cs.dpu.edu* is not a valid e-mail address.

- Every occurrence of the dot character (.)must have a non-@, non-dot character on either side. For example, *bar@cs.* , *.ed@comcast.net*, and *bar@.depaul*, *joe..smith@bob.com* are not valid e-mail addresses.

After reading an email address, the program should display it. If the address is valid, a message should be displayed stating that it is. For invalid addresses, the program should generate an error message for each of the above conditions that was violated.

Suggestions and hints:

- Use `string` class strings in your program. The string functions `length` and `find` are useful for this problem (see p.401 and p.404 of the text).
- Write a separate `bool`-valued function to check each of the four invalidity checks given above. Each should receive the email address via a parameter and then return `true` if the email address is invalid according to the particular invalidity conditions.
- Use a `bool` array of size four to store the results of calling each of these invalidity check functions.
- If you have buffering problems when repeatedly reading strings and chars, make use of the function call `cin.ignore(80,'\n')` to clear the buffer at appropriate times.

To see how this program will work, download and run the executable version of my solution (a4-2.exe).

3. (15 pts.) The following **C++** `main` driver, along with function `myfunc`, uses a **C++ class** `dynarray` that models a dynamic integer array – that is, the class uses dynamic memory allocation to create a contiguous block of memory for storing a specified number of integers. The indexing for a `dynarray` object is the same as for a regular array. But, a `dynarray` can be initialized to size zero.

Write the **C++** `dynarray` **class**. Here is a brief description of all of the class functions that your class should include:

- No-argument constructor – initializes a `dynarray` object to being empty.
- One-argument constructor – uses dynamic memory allocation to obtain a contiguous block of memory for storing *n* `int` values, where *n* is its argument.
- `show` – displays the n^{th} element in the `dynarray`. If the `dynarray` is empty or if *n* is an invalid index, this function should generate an error message.
- `set` – will set the n^{th} element in the `dynarray` to *x*, where *n* is the value of its first argument and *x* is value of its second argument. If the `dynarray` is empty or if *n* is an invalid index, this function should generate an error message.
- `expand` – will take an existing `dynarray` and expand its size by its argument, *s*. **Hint:** To expand a `dynarray`, allocate a new, larger block of dynamic memory, copy the values from the old `dynarray` to the new memory, and deallocate the old memory.
- A destructor to deallocate dynamic memory when a `dynarray` object passes out of scope.

Requirement: When accessing the dynamic array elements in the `set`, `show` and `expand` member functions, you must use the dereferencing operator, `*`, along with pointer arithmetic instead of the array indexing operator, `[]`.

Next, combine your `dynarray` class with the following `main` and `myfunc` code (and needed #include and using namespace statements) and run the resulting **C++ program**. The output

generated from a run of your program should be similar to that shown in the output of a sample run given after the code.

```
void myfunc();
int main()
{
      int size,more,i;
      dynarray y;
      cout << "Enter dynamic array size: ";
      cin >> size;
      dynarray x(size);
      for(i=0;i<size;i++)
            x.set(i,3*i);
      for(i=0;i<size;i++)
            x.show(i);
      cout << "How much more dynamic array space do you want? ";
      cin >> more;
      x.expand(more);
      for(i=0;i<(size+more);i++)
            x.set(i,5*i);
      for(i=0;i<(size+more);i++)
            x.show(i);
      x.show(size+more+5);    //invalid index in show
      x.set(-2,9);            //invalid index in set
      y.set(3,6);             //empty dynarray set
      y.show(3);              //empty dynarray show
      myfunc();
      return 0;
}
void myfunc()
{
     int i;
     cout << "hi from myfunc...\n";
     dynarray y(5);
     for(i=0;i<5;i++)
        y.set(i,i*i);
     for(i=0;i<5;i++)
        y.show(i);
     cout << "bye from myfunc...\n";
}
```

Output from a sample run of the program (user input is in **bold**):

```
Enter dynamic array size: 3
0
3
6
How much more dynamic array space do you want? 2
0
5
10
15
20
Invalid index in show
Invalid index in set
Cannot set - dynarray empty
Cannot show - dynarray empty
hi from myfunc...
0
1
4
9
16
bye from myfunc...
hi from the dynarray destructor...
hi from the dynarray destructor...
hi from the dynarray destructor...
```