EXPLICIT FINAL REVIEW

1. **Consider the following class declarations when answering this question:**

   (IMPLEMENT THE COPY CONSTRUCTOR TO PERFORM A DEEP COPY)

```
class address_record
{
   public:
       string name;
       string address;
       string phone;
        int miles_away;
   };
```

```
class address_book
   {
  public:
     address_book(const address_book &);
  .......
  private:
        int count;//cells in used
               int capacity;//tot cells
             address_record  *address_DB; // dyn array
        };
```

Implement the copy constructor for the class address_book.

```
address_book:: address_book (const address_book & Org)
{
	capacity = Org.capacity;
	count = Org.count;
	address_DB = new address_record[capacity];
	for(int i=0;  i<count;i++)
	{
		address_DB[i] = Org.address_DB[i];
	}
}
```

2. Consider the following class declarations when answering this question:

(FINDING THE SUM OF A FIELD IN THE RECORDS IN A DYNAMIC ARRAY)

```
class address_record
{
    public:
        string name;
        string address;
        string phone;
        int miles_away;
    };
```

```
class address_book
{
    public:
        int closest( const int threshold);
        .......
    private:
        int count;//cells in used
        int capacity;//tot cells
        address_record  *address_DB; // dyn array
};
```

Implement the function "closest". The function will return the total number of address_records in address_DB with the "miles_away" field greater than or equal to " threshold". See the prototype for "closest" inside the class declaration for the class "address_book".

```
int closest(cons tint threshold)
 {
    int sum = 0;
   for(int i=0;  i<count;i++)
   {
    If( address_DB[i].miles_away >= threshold)
     {
         sum++;
     }

    return sum;
}
```

Consider the following class declarations when answering this question:
(FRIEND OPERATOR OVERLOADING WITH CHAINING)

```
class address_record          class address_book
{                                    {
   public:                          public:
      string name;                     friend ostream & operator<<( ostream &, const address_book & org);
      string address;               .......
      string phone;               private:
       int miles_away;                int count;//cells in used
   };                                        int capacity;//tot cells
                                            address_record  *address_DB; // dyn array
                                         };
```

Implement the overloaded "operator<<" with chaining (hit: remember to return the ostream that invoked the function). This function will print all the fields of every address_record stored in address_DB to the screen.

```
cout<<B;
ostream & operator<<(ostream & out, const address_book & org)
{
     for (int i = 0; i < org.count; i++)
     {
          out << org.address_DB[i].name << ”\t”
               << org.address_DB[i].address<< ”\t”
               << org.address_DB[i].phone<< ”\t”
               << org.address_DB[i].miles_away<< endl;
     }
     return out;
}
```

3. **Consider the following class declarations when answering this question:**
   (OPERATOR OVERLOADING AS A MEMBER FUNCTION WITHOUT CHAINING – ALSO EXAMPLE OF ADDING TO THE END OF AN ARRAY)

```
class address_record                    class address_book
{                                           {
   public:                                public:
        string name;                        void operator+( const address_record &);
        string address;                   .......
        string phone;                   private:
         int miles_away;                       int count;//cells in used
   };                                                  int capacity;//tot cells
                                                     address_record  *address_DB; // dyn array
                                               };
```

Implement the overloaded "operator+" without chaining as a member function. This function will add an address_record to address_DB only if the name field does not match any of the address_records stored in address_DB. If a name matches any record in address_DB, do not add it, and print the message "duplicate record". If address_DB is full print the message "address_DB is full".

```
void  address_book::operator+(const address_record & org)
{
    //searching array
    int i;
    for(i=0; i<count; i++)
     {
          if (address_DB[i].name == org.name)
          {
                   cout<<"duplicate record\n";
                   return;
           }
     }
     if (count == capacity)
    {
          cout<<"Array Full\n";
    }
    else
    {
          address_DB[count] = org;
          count = count + 1;
     }
}
```

4. **Consider the following class declarations when answering this question:**

(DELETE A RECORD FROM A DYNAMIC ARRAY)

```
class address_record
{
  public:
       string name;
       string address;
       string phone;
        int miles_away;
 };
```

```
class address_book
  {
  public:
    void delete( const string & key);
    int search( const string & key).
  private:
    int count;//cells in used
    int capacity;//tot cells
    address_record  *address_DB; // dyn array
  };
```

Implement the function "delete" which removes the address_record with a “name” field that matches “key”. You may use the search function to help you implement this function. Assume search returns -1 if key is not in address_DB; otherwise it returns the location of the address_record with a “name” field that matches “key”.

```
void address_book::delete(const string & key)
{
    int loc = search(key);

    if ( loc != -1)
    {
      for(int i=loc; i<count-1; i++)
      {
          address_DB[i] = address_DB[i+1];
       }
       count--;
     }
}
```