COMP 2012
Spring 2013

Midterm Solutions

1. (a)
```
Vector::Vector(int new_size, double value){
    if(new_size < 0){
        exit(-1);
    }
    else if (new_size == 0){
        size=0;
        data = NULL;
     }
     else{
        size = new_size;
        data = new double[new_size];
        for(int i = 0; i < size; i++){
            data[i] = value;
        }
     }
}
```

   (b) The constructor creates a `Vector` $v = (0, 0, 0)$.

2. (a) This is `Copy Constructor`,which constructs an object using another object of the same class (`Vector` in this case).

   (b)
```
Vector::Vector(const Vector& other_vector){
    size = other_vector.get_size();
    if(size==0) //other_vector's size may be 0
      data = NULL;
    else{
      data = new double[size];
      for(int i = 0; i < size; i++){
        data[i] = other_vector[i];
    }
}
```

3. (a) It's destructor, which shows how an object is destroyed when it goes out of scope or deleted (using `delete`).

   (b)
```
Vector::~Vector(){
    if(data != NULL){
        delete[](data);
    }
}
```

4.
```
const Vector& Vector::operator = (const Vector& other_vector){
    if(this != &other_vector) {
         if(data != NULL){
              delete [] (data);
         }
        size = other_vector.get_size();
        data = new double[size];
        for(int i = 0; i < size; i++){
            data[i] = other_vector[i];
        }
        return(*this);
    }
}
```

5. (a)
```
const double& Vector::operator [] (int i)const{

  if(i < 0)
    exit(-1);

  if(i >= size)
    exit(-1);

  // Return vector element
  return(data[i]);
}
```

(b) `double& operator [] (int);` is the overloading of operator for a non-const object, it returns a certain element of the object which can also be the lvalue. That is, it supports both, for examples, `v[3] = 3.4;` or `d=v[3];`.
`const double& operator [] (int)const;` is the overloading of operator [] for const object only. It returns some element value of this object, which cannot be lvalue and hence cannot be changed.

6.
```
Vector Vector::operator + (const Vector& other_vector) const{
    int other_size = other_vector.get_size();
    if(other_size != size){
        exit(-1);
    }

    Vector sum=Vector(size,0);
    for(int i = 0; i < size; i++){
        sum[i] = data[i] + other_vector[i];
    }
    return(sum);
}
```

7. Define a global function for overloading the operator +. First add the following in `Vector.h`:
```
Vector operator + (double scalar, const Vector& v);
```
In `Vector.cpp`, add the following codes:

```
Vector operator + (double scalar, const Vector& v){
    int size =v.get_size();
    Vector sum=Vector(size,0);
    for(int i = 0; i < size; i++){
          sum[i] =  scalar + v[i];
     }
     return(sum);
}
```

Alternatively,

```
Vector operator + (double scalar, const Vector& v){
    int size =v.get_size();
    Vector dv(size, d);

    return( dv + v );
}
```

8.
```
void Vector::readin(istream& in){
    if(data != NULL){
        delete [] (data);
    }
    int size;
    in>>size;
    set_size(size);
    data = new double[size];
    for (int i=0; i<size; i++) {
         in>>data[i];
    }
}
```

9.
```
void Vector::read_vector(const char* fileName){
    ifstream fin;
    fin.open(fileName);
    if(!fin){
        cout<<"Cannot open file.\n";
        exit(-1);
    }
    else readin(fin);
    fin.close();
}
```

10. To overloading the `>>` operator, the easiest way is to call the private utility function readin(), so first friend the function to the class `Vector` by including the following in `Vector.h` as

    ```
    friend istream& operator>>(istream &in, Vector& V);
    ```

    Then implement this function in `Vector.cpp` file as:

    ```
    istream& operator>>(istream& in, Vector& V){
        V.readin(in);
        return in;
    }
    ```