Programming Assignment

This assignment gives you more experience working with designing classes, namespaces, operator overloading, etc. The goal of the project is to implement a class SparseVector, which is exactly the linked list assignment rewritten in C++. You should be able to reuse a lot of the code, just make sure to use **delete and new** instead of **free and malloc.** There is a couple of new features - vector by integer and integer by vector multiplication and a couple more operators that need to be implemented.

Most (if not all) of the information required to implement this assignment can be gleaned from the driver that is provided. This technique of development is called Test Driven Development. The idea behind TDD is that you are given test cases and need to implement code that will cause the tests to succeed. Your job is considered complete when all of the tests have succeeded. You have been given a framework, a driver, and the expected output so your task is not as hard as it might sound. You need to "fill in the blanks", so to speak. Additional information will be discussed in class during the lectures.

All of the classes will reside in the *CS225* namespace.

The most interesting operator in this assignment is **operator[ ]**. Since SparseVector class doesn't store 0's - we **can not** return a reference to the actual data (this is how **operator[]** is implemented for most data types), it just may not exist yet (for example vector V has a single element at position 3, but nevertheless V[5] should be a legal expression). It's possible though to implement a function like

```
int operator[](long pos);
```

which returns the data by value (0 if the corresponding ElelmentNode does not exist), but then the following doesn't compile

```
SparseVector v;
v[12]=10;
```

To achieve the above functionality we need an additional **proxy class**:

- the actual array modifications are done via 3 functions, Insert, Delete, and Get.
- SparseVector::operator[] returns an instance of a proxy class.
  `ElementProxy operator[](long pos);`
  This class is defined as follows:
  - proxy constructor simply stores parameters of operator[] and a reference to the SparseVector object element of which it represents.
  - proxy's conversion operator allows automatic conversion of the proxy type to the type contained in the SparseVector (integer). Conversion operator simply calls Get, and returns the result by value. It will be called implicitly in most r-value contexts. For example “int i = v[3];” - v[3] is a proxy, so compiler uses conversion operator to perform assignment.
  - proxy's operator= is overloaded to take a value of the type stored in the vector. It calls Insert with this value.
- another question is what if the SparseVector object is const-qualified. The above operator[] is a non-const method, so cv[i] will not compile if cv is const. At this point we should decide whether this behavior (not compiling) is a bug or a feature. The truth is - it depends: “cv[i]=10;” should not compile, while “int i=cv[10];” should be OK. The last command only accesses – does not modify SparseVector. To solve this problem we need a second operator[] which is a const-method. One may try a signature like this
  ```
  const ElementProxy operator[](long pos) const;
  ```
  but then the following is possible in the client's code
  ```
  const SparseVector cv = v;
  ElementProxy ep = cv[10]; //non-const from a const
  ep=20;
  ```
  and cv is modified. One may try to continue “fixing” by inventing more hacks and quite possible to get it to work (for example by overloading ElementProxy assignments for const and non-const parameters).
  But - there is a simple and elegant solution add this to SparseVector:
  ```
  int operator[](long pos) const;
  ```
  Now the returned value may be used in RHS, but not in the LHS, since it returns by value. Spend 5 minutes looking at both operators[]'s and make sure you see how they work in “combination”.

Here is a skeleton of the header file. Do not modify provided declaration of these functions/classes, but you may and have to extend them. Proxy related stuff is commented out. My suggestion is get the basic functionality first and then move to proxy.

```
struct ElementNode {
int data;
int pos;
struct ElementNode *next;
};

//forward declaration
//class ElementProxy;

class SparseVector {
public:
int Get(long pos) const;
void Insert(int val, long pos);
void Delete(long pos);
void PrintRaw () const { //used for grading
ElementNode* curr=pHead;
std::cout << "Raw vector: ";
while (curr) {
std::cout << "(" << curr->data << ", " << curr->pos << ")";
curr=curr->next;
}
std::cout << std::endl;
}
friend std::ostream& operator<<(std::ostream &out, const SparseVector &v);//implemented
//......
private:
ElementNode* pHead;
long dimension;
};
/* uncomment when done with basic class functionality and ready to implement Proxy
class ElementProxy {
public:
ElementProxy(SparseVector& v, long pos);
operator int() const;
// .......
private:
SparseVector &v;
long pos;
};
*/
```

Here is partial implementation that shows how to maintain and use dimension (size) of a vector:
first we have to increase it automatically in the Insert method (or any other method that inserts new elements):

```
void SparseVector::Insert(int val, long pos) {
if (val==0 ) { Delete(pos); return; } //you may change this line
if (pos>=dimension) dimension=pos+1; // automatically set dimension (it effects
operator<< only)
// ..................................
}
```

and this is the only place where we use it:

```
std::ostream& operator<<(std::ostream &out, const SparseVector &v) {
int i,last_pos=-1;
ElementNode* p_e=v.pHead;
while (p_e) {
for (i=last_pos+1;i<p_e->pos;++i) out << " " << "0";
out << " " << p_e->data;
last_pos=p_e->pos;
p_e = p_e->next;
}
for (i=last_pos+1;i<v.dimension;++i) out << " " << "0";

return out;
}
```

Note that dimension is NEVER decreased, that is – DO NOT MODIFY dimension in Delete method.

Possible next step (**DO NOT DO IT FOR THIS ASSIGNMENT**): when you are done you may hide Insert, Delete, and Get from the client (move to a private area and declare Proxy as a friend), since they are useful while testing, but client should only use the provided operator[].

To submit:

- vector.cpp, vector.h