The following algorithm accepts a vector of coefficients and an x value and returns the corresponding polynomial evaluated at x.

```
double PolynomialEval(const vector<double>& coeff, double x)
{
  assert(coeff.size() > 0);
  double y = 1.0;
  double retVal = coeff[0];
  for(int i = 1; i < coeff.size(); ++i) {
    y *= x;
    retVal += y * coeff[i];
  }
  return retVal;
}
```

Identify the component which represents the input size and give the Big-oh notation for the time complexity of this algorithm. In order to receive partial credit, be sure to explain your reasoning.

Show that the following equality is incorrect:

$$10n^2 + 9 = O(n)$$

The following algorithm is an alternative version of the Chain<T>::insert() function. Determine whether it is a valid substitute for the one given in class. If not, indicate the input conditions for which it would fail.

```
template <class T>
void Chain<T>::insert(unsigned int k, const T& val)
{
  if( k<=size() ) {
    Link<T> *newLnk = new Link<T>(val);
    if ( 0==k ) {
      newLnk->next = first;
      first = newLnk;
    } else {
      Link<T> *itr = first;
      while( --k>0 ) {
        itr = itr->next;
      }
      newLnk->next = itr->next;
      itr->next = newLnk;
    }
  }
}
```

Recall the Chain<T> class discussed in lecture:

```
template<class T>
class Chain {
public:
  Chain();
  ~Chain();
  bool empty() const;
  unsigned int size() const;
  int find(const T& val) const;
  bool erase(unsigned int k);
  void insert(unsigned int k, const T& val);
  ostream& display(ostream& out) const;
private:
  Link<T> *first;
};
```

Write the class generated by the precompiler the first time the precompiler encounters the following:

```
  Chain<bool> aChainOfBools;
```

Briefly describe the role of a *stub* in program testing and give an example.

Briefly describe the role of an *invariant* in establishing confidence in program correctness.

What characteristics make an *adaptor* class different from a standard class?

Recall from lecture the generalized generic algorithm accumulate():

```
template <class InputIterator, class T,
          class BinaryOperation>
T accumulate(InputIterator start, InputIterator finish,
             T count, BinaryOperation binary_op)
{
  while (start != finish)
    count = binary_op(count, *start++);
  return count;
}
```

What type of iterator is required by this generic algorithm.

Describe the concept of **folding** as it pertains to hash functions. Give an example.

Suppose you are asked to place all of the buildings in Milwaukee into a hash table of size 1010.

Which hash function would be best:

1. (Number of bricks)% 1003
2. (Number of windows)% 2000
3. (Numerical part of street address)% 1010
4. (Sum of all the ASCII values of the street address)% 1003
5. (Number of broken ceiling tiles)% 1010

For full credit, justify your answer.

Recall that the BinaryNode class has three protected data members (BinaryNode<T>* left, BinaryNode<T>* right, and T data) and the BinaryTree class has one protected data member (BinaryNode<T>* root). Suppose you are given the following find() member function:

```
  template <class T>
2 bool BinaryTree<T>::find(const T& x) const
  {
4   BinaryNode<T>* found = find(root, x);
    if(found==0)
6     return false;
    return true;
8 }
```

Finish writing the recursive find() member function called in line four of the above function. You may assume that the binary tree is sorted, i.e., it is a binary search tree.

```
template <class T>
BinaryNode<T>* BinaryTree<T>::find(BinaryNode<T>* subRoot, const T& x) const
{
```

Write a templated C++ class called BinaryNode which represents a node in a binary. Your class should contain all of the necessary data members for a node in a binary tree. (*Hint: Consider the* link *class from the* Chain *implementation*)

Recall that in class we used a map to represent a sparse vector. Suppose that we wished to use the map class to represent a sparse matrix. How could this be done? Give an example by creating an object called sMatrix that could represent a sparse matrix with only one non-zero value:
sMatrix[52333][3] = 5.821.