Hashing is a technique used to store and/or access elements in a list in a relatively constant amount of time $O(1)$. A hash function is a function which maps a large range of key values into a smaller range of physical addresses.

Your assignment is to build a hash table class with the following characteristics:

1) **Hash Table:**
   
   Your hash table must be dynamically allocated. The constructor for your hash table should be able to accept an integer ($n$) as a parameter and creates a hash table such that the size of the table is equal to the next prime number which is greater than ($n$).

2) **Hash Function:**
   
   A hash function is used to produce a **Semi-unique address** for a given **key** to be stored in the hash table. You must choose and combine two of the following methods (Division method and one other) for implementing your hash function:
   
   1) Division Method
   2) Folding Method
   3) Truncation Method
   4) Mid-Square Method
   5) Radix Conversion

3) **Collision Resolution:**
   
   Despite our best effort in selecting a good hash function, often multiple keys hash into the same hash table location (bucket). In such cases, we must be able to deal with the collision. For the purpose of this assignment, you have two choices to handle any possible collisions:
   
   a) Option 1: Using a Template Linked List class, develop a chaining collision resolution method to hold the overflow.
   b) Option 2: Using a Template Binary Search Tree class, develop a collision resolution method to hold the overflow.

For information about generating prime numbers see:
[http://cs.iusb.edu/~hhakimza/C441/prime.c](http://cs.iusb.edu/~hhakimza/C441/prime.c)

Also, feel free to use the Template Tree class provided in class.

**What to hand in:**

- Cover page with paper title, your name, course # and name, assignment #, date, etc.
- Source code, Data files, sample runs.

**On your own:**

- Write a **Populate** function. The **Populate (int Num_Records, ...)** function should take an integer as an argument and proceed to randomly generate $Num_Records$ records and inserts it into your hash table. Try creating 10, 100, and then 1000 records, each time try the operations (initialize, insert, delete, update and search) and time your program as the number of records increase in your file. Document your observations. Be prepared to discuss your observations in class.
If you choose to use a binary tree as your overflow, your hash class may look like the following:

```cpp
#include "BinaryTree.h"

#define DIVISION 0x01
#define FOLDING 0x02
#define TRUNCATION 0x04
#define MIDSQUARE 0x08
#define RADIXCONVERSION 0x10

class HashNode {
public:
    int data;
    BinaryTree<int> OverFlow;

    HashNode() {
        data = -1;  // Assuming -1 represents an empty data value
    }
};

class Hash {
public:
    HashNode *HashTable;
    int Size;
    unsigned char HashFunction;

    Hash(int size, unsigned char hash_function);

    int DivisionMethod(int value);
    int FoldingMethod(int value);
    int TruncationMethod(int value);
    int MidSquareMethod(int value);
    int RadixConversionMethod(int value);

    void Print();
    void Insert(int value);
    void Populate (int Num_Records, ...);
    int NextPrime(int value, char flag);
};
```
If you choose to use a linked list as your overflow, your hash class may look like the following:

```cpp
#include "LinkedList.h"
#define DIVISION 0x01
#define FOLDING 0x02
#define TRUNCATION 0x04
#define MIDSQUARE 0x08
#define RADIXCONVERSION 0x10

class HashNode {
public:
    int data;
    LinkedList OverFlow;
    HashNode() {
        data = -1; // Assuming -1 represents an empty data value
    }
};

//-------------------------------

class Hash {
public:
    HashNode *HashTable;
    int Size;
    unsigned char HashFunction;
    Hash(int size, unsigned char hash_function);
    int DivisionMethod(int value);
    int FoldingMethod(int value);
    int TruncationMethod(int value);
    int MidSquareMethod(int value);
    int RadixConversionMethod(int value);
    void Print();
    void Insert(int value);
    void Populate (int Num_Records, ...);
    int NextPrime(int value, char flag);
};
```