CSCI 340 **Computer Assignment 9** **Fall 2017**
(10 points)

Hashing with Separate Chaining and Indirect Sorting

For this computer assignment, you are to write a C++ program to create, search, print, and sort an item inventory. The item inventory information will be kept in a sequentially allocated table with a given size of entries with default size TBLSZ = 31. Each table entry is described by the following structure, as specified in the header file Entry.h.

struct Entry { string key, desc; unsigned num; };

The key field is the item identifier, which is two uppercase letters followed by a decimal digit (e.g., AD5 or XR8). The desc field contains the item description, and the num field contains the number of copies of an item in the inventory. The item table can be accessed directly using the *separate chaining* technique to resolve the collisions. The *private* hash function expects an item key as the input argument, and it returns an integer in the range: 0 ... (TBLSZ−1), which has the following prototype: int HT :: hash (const string& key). The object file hash.o, which is in the same directory with the header file Entry.h, contains the implementation of this function.

The item inventory system is defined as a class, named as HT, and its definition is given in the header file hTable.h, which is in the same directory with Entry.h. The hash table hTable (container of type vector < list < Entry > >) is used to store the entries of the item inventory, and the pointer table pTable (container of type vector < Entry* >) is used to store the addresses of the entries in hTable. The member functions of HT are described below.

- HT :: HT (const unsigned& hs) : The hash table, which is a vector of list < Entry > containers, can be created dynamically for a given fixed size hs by its constructor. Initially, the pointer table, which is a vector of type Entry*, is empty but its size will increase dynamically after inserting a new Entry item in this table, which is the address of the inserted item in the corresponding list container.

- HT :: ~HT ( ) : Since the hash table is implemented as a vector of list containers, the destructor of the hash table first frees the memory for all list containers and then frees the memory for the vector containers for the hash table and the pointer table.

- void HT :: insert (const Entry& e) : This *public* function inserts the record of the item e : (key, desc, num) in the hash table. If the key already exists in the table, then the function prints an error message; otherwise, it prints the index value of the inserted record in the hash table and it also inserts the address of the record (in the hash table) into the pointer table. Since each element of the hash table is implemented as a list container with the data type Entry, in the case of a collision, simply insert the new record at the beginning of the corresponding list container. To check if the record e is already in the hash table, you can use the function find_if ( ) from the STL. To compare the key of the record e with the keys of the elements in the list container, list < Entry >& l = hTable [i], in hash table position i,

you can use either a *predicate* or a *lambda* for the compare component of the find_if ( ) function. If the item is a new item, then the find_if ( ) function returns l.cend ( ).

- void HT :: search (const string& key) : This *public* function searches the hash table for a record with a given key. As in the insert ( ) function, you can use the find_if ( ) function from the STL to search for a record in the hash table. If the search is successful, search ( ) function prints the information for the record; otherwise, it prints an error message.

- void HT :: hTable_print ( ) : This *public* function prints the subscript and the contents of all (and only) the active records in the hash table.

- void HT :: pTable_print ( ) : This *public* function prints the contents of all (and only) the active records in the hash table. Since the records need to be printed in alphabetical order with their key values, this function first sort the elements of the pointer table using the sort ( ) function from the STL. The cmp component for sorting, bool cmp (Entry* p, Entry* q), returns true if the key of the item in location p comes before the item in location q in alphabetical order.

The input data consists of three types of records, as described below:

- Insertion records have the format
  A:item-key:item-number:item-description:
- Search records have the format
  S: item-key:
- Table print records have the format
  P:

Program Note: The records stored in the inventory system have unique keys. Duplicate keys are not permitted.

The main ( ) routine in the driver program prog9.cc (provided), implements the following tasks:

1. Prints a header message.
2. Creates an empty hash table.
3. Reads the entire input file, and process each record as follows: if the record transaction code is:
   - A – Searches the hash table for a record having the same key. If such a record is found, prints an appropriate "duplicate key" message, and ignores the information from the input record. Otherwise, inserts the record in the hash table, and places the address of the record in the pointer table.
   - S – Searches the hash table for a record with a given key, and prints the key and other information for the record. If the record is not found in the table, it prints an appropriate "key not found" message.
   - P – Prints the subscript and contents of each active record in the hash table.

4. After the data in the input file has been read, and the tables have been constructed, sorts the records in the pointer table, and then prints the records in the hash table in ascending order with the key field.
5. Prints a program termination message.

To compile and link the source file prog9.cc (contains the main ( ) routine and some auxiliary functions for reading and printing data) of the driver program and your source file hTable.cc (contains your implementations of the member functions of HT) with the system library routines, execute: Make N=9. To test your program, execute: Make execute N=9. This will execute the program with data files prog9.d1 and prog9.d2, and generate the output on both your terminal screen and in output files prog9.out1 and prog9.out2. After you are done with the program, you don't need its object and executable files any more. To delete them, execute: Make clean.

At the top of your source file hTable.cc, insert the following statement:
#include "/home/cs340/progs/17f/p9/hTable.h".

The following is a dependency chart for the files for this computer assignment. All these files, including the input and correct output files are in the program directory: ~cs340/progs/17f/p9, but the only file that you need to submit is: hTable.cc.