Homework 1

- Symbol Table
- Implement the functions defined in SymTab.h in a file called SymTab.c
- Do not change the contents of SymTab.h
- Implement a driver program to test your implementation
- You will have to implement a hash function in your .c file.
- You will also have to implement a resize function. The resize function should approximately double the table size when the load factor gets greater than 2.
- Due Wednesday February 8

SymTab.h

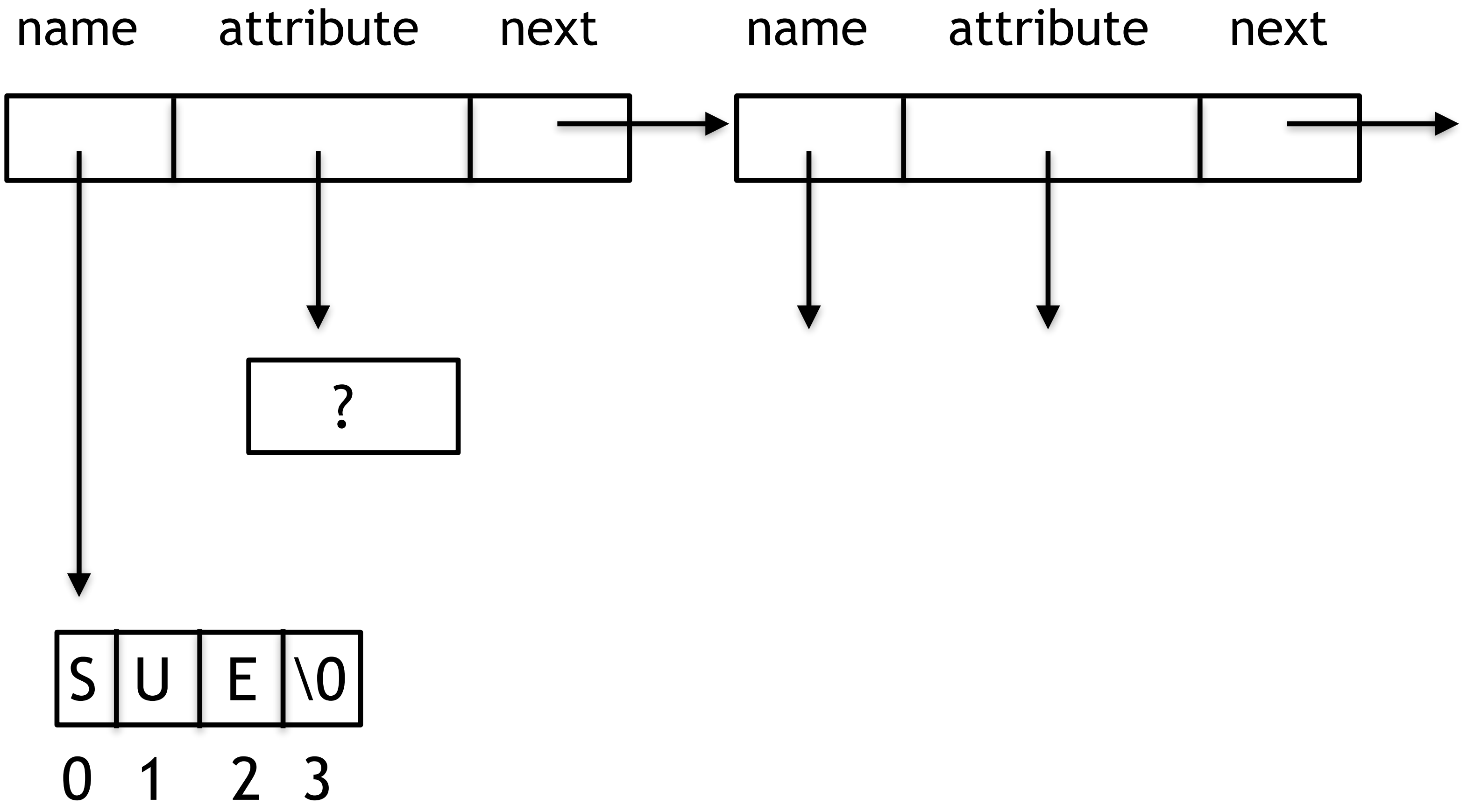
```c
/*
 API for a symbol table. The symbol table stores (name, attribute) pairs. The data
 type for the attribute is void * so programs that use the symbol table can
 associate any attribute type with a name

 The symbol table is implemented using a separate chaining hash table.

*/

//A SymEntry is the building block for linked lists of (name, attribute) pairs
typedef struct SymEntry {
    char * name;
    void * attribute;
    struct SymEntry * next;
} SymEntry;
```

SymEntry

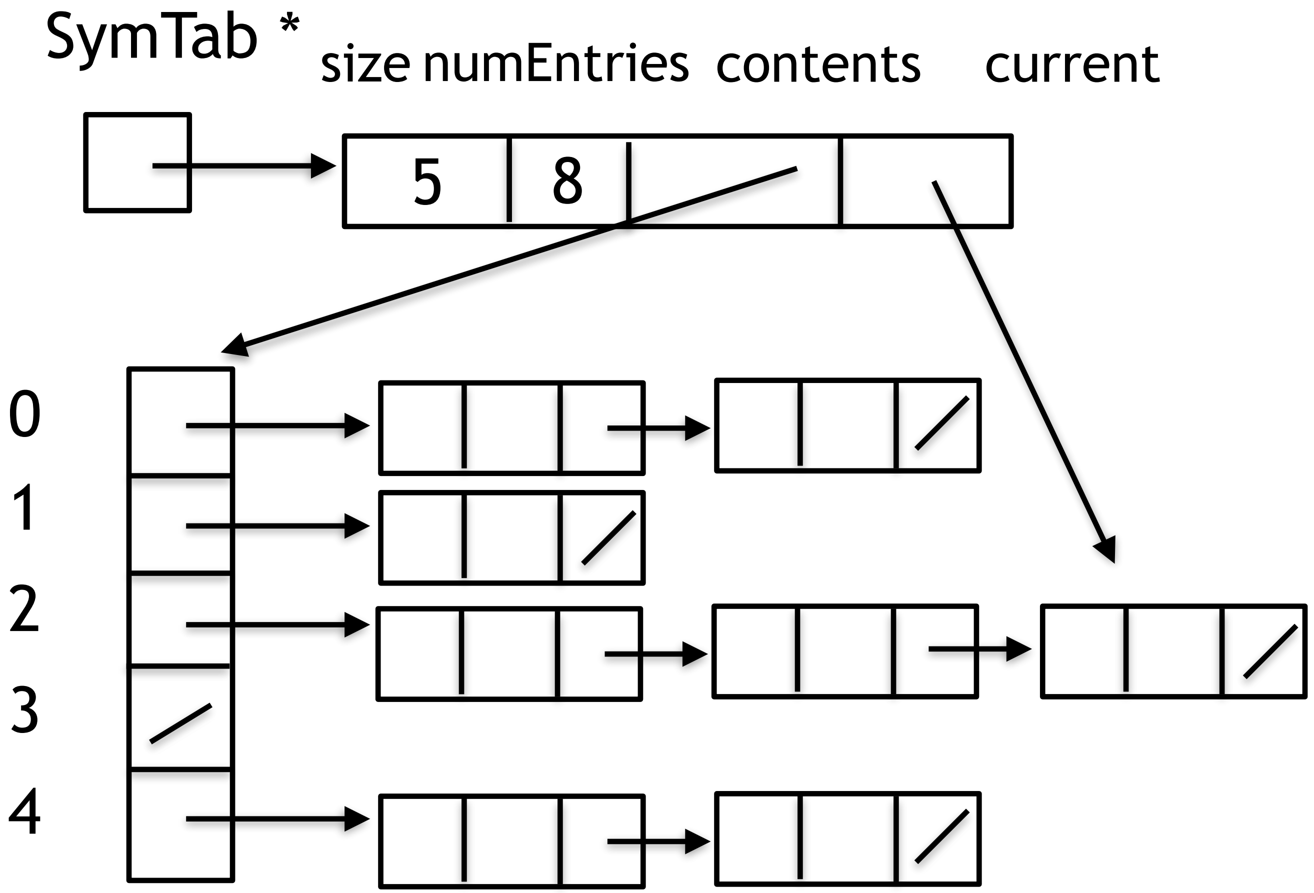


SymTab.h

```c
/*
 Each symbol table is represented by a SymTab
 size is the current number of lists in the separate chaining hash table
 numEntries is the current number of names stored in the symbol table
 contents is an array of lists (i.e. points to the zeroth element in the array)
 if current is not NULL it points to the current (name,attribute)
    pair in the symbol table
*/
typedef struct {
    int size;
    int numEntries;
    SymEntry **contents;
    SymEntry *current;
} SymTab;

SymTab * createSymTab(int size);
/* PRE: size >= 0
   size is an estimate of the number of items that will be stored in the symbol
        table
   The initial size should be a prime number greater than or equal to size.
   Return a pointer to a new symbol table
*/
```

SymTab



SymTab.h

```
//In the following functions assume a pre condition that table references a
//previously created symbol table

void destroySymTab(SymTab *table);
//recover space created by the symbol table functions
//no functions should use the symbol table after it is destroyed

int enterName(SymTab * table, char *name);
/*if name is not in the symbol table, a copy of name is added to the symbol table
   with a NULL attribute, set current to reference the new (name, attribute) pair
   and return 1

   If a new entry is added and the load factor becomes greater than 2 the table should
   resized.

  if name is in the symbol table, set current to reference the (name, attribute)
   pair and return 0


*/

int findName(SymTab *table, char *name);
/*if name is in the symbol table, set current to reference the (name, attribute)
   pair and return 1
  otherwise do not change current and return 0
*/
```

SymTab.h

```c
int hasCurrent(SymTab *table);
//if current references a (name, attribute) pair return 1
//otherwise return 0;

void setCurrentAttr(SymTab *table, void * attr);
//PRE: hashCurrent() == 1
//change the attribute value of the current (name, attribute) pair to attr

void * getCurrentAttr(SymTab *table);
//PRE: hasCurrent() == 1
//return the attribute in the current (name, attribute) pair

char * getCurrentName(SymTab *table);
//PRE: hasCurrent() == 1
//return the name in the current (name, attribute) pair
```

SymTab.h

```c
//Assume no changes are made to the symbol table while iterating through the symbol table

int startIterator(SymTab *table);
//if the symbol table is empty, return 0
//otherwise set current to the "first" (name, attribute) pair in the symbol table and return 1

int nextEntry(SymTab *table);
/*if all (name, attribute) pairs have been visited since the last call to
  startIterator, return 0
  otherwise set current to the "next" (name, attribute) pair and return 1
*/
```

Homework 1 Submission

- Late next week I will post a test driver and data file for the homework.
- Upload one zip file. The file should contain SymTab.c and a text file containing the output of my test driver.