An-Najah National University
Department of Computer Engineering
Compiler Construction-66416
Fall 2016

Programming Assignment #3

Dr. Raed Alqadi

Write, test, and debug a symbol table class that implements the operations below.
A symbol table maps a string (an identifier's name) to a record containing information about the identifier. Symbol tables are of type SymbolTable. They hold records of type SymbolTableEntry that contain fields one of which is the string naming the identifier.

The symbol table class must provide the following operations:

Constructor: SymbolTable (int fold_case_flag)
If fold_case-.flag is true, the symbol table should fold uppercase letters to lowercase (e.g., "DoG" would be the same as "dog"), otherwise the table should preserve case. There should be no limit (except memory size) on the number of tables created by a program.

SymbolTableEntry *GetSymbol (char *str)
Look up the string *str* in the symbol table. If an entry for *str* is already in the table, return the entry. If there is no such entry, do not modify the table and return NULL.

SymbolTableEntry .*PutSymbol (char *str)
Look up the string *str* in the symbol table. If an entry for *str* is already in the table, return the entry. Otherwise, make a new entry, insert it into the symbol table, and return a pointer to it.

void ClearSymbolTable ( )
Remove all entries from the symbol table and reclaim their memory space.

void PrintSymbolStats ( )
Print some statistics on the utilization of symbol. Choose appropriate metrics such as the number of entries, the number of free slots, the length of the search chains, etc:

You may implement your symbol table with any hashing algorithm. For more details on hashing, see Aho, Sethi, and Ullman, pp. 433-438; and your favorite data structure textbook.

Following is the header file ***Symbol.h***, you may modify or improve the header file as you wish.

```
/* symbol.h  */
 //A symbol table entry holds a string and any other values that a programmer wants to
add to it. */

typedef enum{
        type_integer,
        type_string,
        type_boolean,
        type_float,
        type_none
        } j_type;

/* printable versions of their names. */
static char *type_names[] = {"integer", "string", "boolean", "float", "none"};

typedef enum {
        ste_var;                // a variable
        ste_const;              //A constant
        ste_routine;            //A routine
        ste_undefined;          // ste_entry
} ste_entry_type;

// You may change the following definition  to a class
struct  SymbolTableEntry{
        char                    *name;
        SymbolTableEntry        *next;
        ste_entry_type          entry_type;

        // User-modifiable fields go here:

        union{
                /l .for a variable record its type
                struct{
                        j_type      type;
                } var;
                // for a constant record its value
                struct{
                        int     value;
                } constant;
                /* for a routine, record formal parameters and result type */
                struct{
                        // SteListCelll   *formals;// will be defined later
                        j_type          result_type;
                } routine;
        } f;
}; //end of SymbolTableEntry definition
```

/* A symbol table is a hash table that maps from strings to symbol_table_entries: */

```
class  SymbolTable{
        /* Hash table */
        private:
        SymbolTableEntry      **slots;          //Pointer to hash table array of entries
        int       fold_case;                     // Non-zero => fold upper to lower case

        // Statistics on hash table effectiveness
        int       number_entries;        // Number of entries in table
        int       number_probes;         // Number of probes into table
        int       number_hits;           // Number of probes that immediately found entry
        int        max_search_dist;      // Maximum entries searched

        SymbolTable    *next;            // To be used to create a stack of symbol table

        // add your defined functions, e.g hash .

        // Externally-visible functions
        public:
        SymbolTable();                          // fold_case will be set to zero
        SymbolTable(int  flod_case_flag);
        void     clear_symbol_table ( );
        SymbolTableEntry      *Get_symbol ( char *);
        SymbolTableEntry      *Put symbol ( char *);
        void     print_symbol_stats ( );

        // will add more functions here in the future

};
```