CS 536 — Spring 1995
Announcement 2

Programming Assignment #1

Symbol Table Module

Due: Monday, February 6, 1995

Not accepted after class on February 13, 1995

Write a C or C++ module that implements an abstract data type *symbol table.* Also write a main program to thoroughly test the symbol table module. A symbol table contains a collection of *attributesRefs,* which are accessed through an integer *serial number.* Each serial number is uniquely associated with an identifier. Each attributesRef points to an *attributes* type (usually a struct) that holds the data associated with a given identifier. The C version of the symbol table module should export the types *symbolTable, attributes, attributesRef, serialNum* and *string.* The C++ version of the symbol table module should export the classes *symbolTable* and *registration* and the types *serialNum* and *string.*

C Version

The C version should implement the following operations:

(1) serialNum registerStr(string s);

RegisterStr registers string s as an identifier and returns a serial number that uniquely identifies it. If string s has previously been registered, the same serial number as before should be returned. If s has never been registered, a new serial number should be assigned to it and returned. For purposes of registration, case is insignificant, so “ABC,” “AbC,” and “abc” should all receive the same serial number when registered.

(2) string symToStr(serialNum sn);

Return the string associated with the serial number sn. If sn has never been assigned by the registerStr routine, an error message should be produced.

(3) symbolTable ∗create(void);

Create and initialize an empty symbol table.

(4) void destroy(symbolTable ∗st);

Destroy a symbol table, but *not* associated attributes values.

(5) bool insert(symbolTable *st, serialNum key, attributesRef atr);

Insert an attributesRef corresponding to the given serial number into the symbol table. If no attributesRef corresponding to the serial number was in the table, return false. If an attributesRef corresponding to the serial number was in the table, return true and reset the attributesRef to the new value of atr.

(6) attributesRef lookUp(symbolTable *st, serialNum key, bool *present);

Look up the attributesRef corresponding to the given serial number and return it; set present to true. If no entry exists, set present to false; value returned is undefined (but NULL is a good choice).

C++ Version

In C++, you'll define two classes, *registration* and *symbolTable.* All required operations will be members of one of these two classes.

(1) registration

This class is used to register strings, associating them with unique serialNums. Normally only one instance of this class is needed. Operations provided by registration (as class members):

(a) serialNum registerStr(string s)

Register string s as an identifier and returns a serial number that uniquely identifies it. If string s has previously been registered, the same serial number as before should be returned. If s has never been registered, a new serial number should be assigned to it and returned. For purposes of registration, case is insignificant, so "ABC," "AbC," and "abc" should all receive the same serial number when registered.

(b) string symToStr(serialNum sn)

Return the string associated with the serial number sn. If sn has never been assigned by the registerStr routine, an error message should be produced.

(c) registration()

The class constructor.

(2) symbolTable<attributes>

The type attributes is a parameter to symbolTable, which is a template class. A symbolTable is indexed by serialNums which are created by a call to registerStr (in class registration). A symbolTable exists to associate with each serialNum an attributes pointer. This pointer, usually to a struct, references all the data we wish to associate with a symbol (its type, its value or address, etc). Operations provided by symbolTable (as class members):

(a) symbolTable()

The class constructor; initialize an empty symbolTable.

(b) ˜symbolTable()

The class destructor; deallocate internal structure of a symbolTable. **Do not** destroy the associated attributes values.

(c) bool insert(serialNum key, attributes ∗atr)

If there is no attributes pointer currently associated with key in the symbolTable, insert atr and associate it with key; return false. If there is an attributes pointer already associated with key in the symbolTable, replace it with atr; return true.

(d) attributes ∗lookUp(serialNum key, bool& present)

If there is an attributes pointer associated with key in the symbolTable, return it and set present to true. If there is no attributes pointer associated with key in the symbolTable, set present to false; the value returned is undefined (though NULL is a good choice)

Considerations/ Requirements

(1) In implementing the registerStr routine you will need to use a hash table. You will hash the given string to see if it has previously been registered. If it has, its assigned serial number is stored in the hash table. If it hasn't, a new serial number is assigned, and this value is stored in the hash table.

(2) Information on hash tables: Fischer and LeBlanc pp. 253-261; Aho, Sethi, and Ullman pp. 433 - 438; any data structures book (on reserve for cs367).

(3) You may use any reasonable collision-resolution scheme (a scheme that allows the hash table to become full is not reasonable). Chained resolution is a good choice.

(4) Individual symbol tables *need not* be implemented as hash tables because they are indexed by serial numbers which are consecutive integers starting at 0 (or 1). A simple array of attributesRefs would do *except* for the fact that upper bound of the array should be the maximum serial number assigned, which usually isn't known in advance. Thus you should allow the symbol table to grow by allocating it in segments or resizing it. The library routine *realloc* can be used to conveniently extend arrays allocated in the heap.

(5) There are "skeleton" symbol table modules, main programs and makefiles in the public directories (˜cs536-1/public/proj1/c and ˜cs536-1/public/proj1/c++). Feel free to copy these files into your own directory and to use them as the basis for your assignment.

(6) The C++ skeleton uses a template class to implement symbolTables. This is really the right way to do it, since the type of attributes varies for different symbol tables. If you find templates too hard to use or understand, you may hardwire the type of attributes directly into symbolTable (this is what C users will have to do).

What To Hand In

Hand in listings of your symbol table module, your main program, your test data, and the outputs your program produces using your test data. Remember that the quality of both your documentation and test data is very important. Make sure that you provide both external and internal documentation and that your test data exercises *every* statement in your symbol table module (the purpose of the test data is *not* to test the main program, but rather to test the symbol table module).

For testing purposes, your main program should allow the user to create a symbol table, destroy it, insert (identifier, integer) pairs into the symbol table (using the registerStr function indirectly), lookup a given identifier and print out its associated integer (again, using the registerStr function indirectly), register identifiers, and print symbols associated with particular serial numbers. To test multiple symbol tables, implement a stack of symbol tables, with create pushing a new (empty) table, destroy popping and deleting the top symbol table on the stack, and insert, lookup, register and print implicitly accessing the top symbol table on the stack. To facilitate grading, the format of the dialogue with the user should look like the following example (text entered by the user is printed in bold face). (Note that this example is only meant to suggest a convenient testing structure; it is not by itself an exhaustive test set.)

```
Create, Destroy, Lookup, Insert, Register or Print? C
symbol table created.
Create, Destroy, Lookup, Insert, Register or Print? I
Enter identifier: wisconsin
Enter associated integer: 1848
Create, Destroy, Lookup, Insert, Register or Print? I
Enter identifier: florida
Enter associated integer: 1845
Create, Destroy, Lookup, Insert, Register or Print? L
Enter identifier to search for: wisconsin
"wisconsin" found: associated integer is 1848
Create, Destroy, Lookup, Insert, Register or Print? L
Enter identifier to search for: florida
"florida" found: associated integer is 1845
Create, Destroy, Lookup, Insert, Register or Print? L
Enter identifier to search for: hawaii
"hawaii" not found
Create, Destroy, Lookup, Insert, Register or Print? I
Enter identifier: wisconsin
"wisconsin" already in symbol table.
Enter new associated integer: 1836
Create, Destroy, Lookup, Insert, Register or Print? R
Enter identifier: wisconsin
serial number is 0
Create, Destroy, Lookup, Insert, Register or Print? R
Enter identifier: california
serial number is 3
Create, Destroy, Lookup, Insert, Register or Print? P
Enter serial number: 1
identifier is "florida"
Create, Destroy, Lookup, Insert, Register or Print? P
Enter serial number: 7
"7" is an illegal serial number
Create, Destroy, Lookup, Insert, Register or Print? D
symbol table destroyed.
```

You should also place an executable copy of your symbol table test program in the directory ˜cs536-1/public/proj1/handin. The name of the program should be your login. When your program begins execution it should print out your full name and student ID number. Your program should follow the input format illustrated above. Symbol table commands are single letters, identifiers are strings (delimited by blanks, tabs or ends-of-line) and attributes (in this case) are simple integers. If we think your test cases are inadequate, we may try your program with some of *our* test data.