```
//***********************************************************************************************/
//*** Header file for Search 23 Tree in Exercise 2 by Wu, Y.H.@CYCU-ICE
//***********************************************************************************************/
#include <stack>                              // push, pop, top
#define     PTR_NUM 3
#define     KEY_NUM PTR_NUM - 1
//***********************************************************************************************/
typedef struct slotT                          // a slot in a tree node
{     vector<int> rSet;                       // a set of record identifiers with the same key
      string      key;                        // a key for comparisons
}     slotType;
typedef struct nT                             // a tree node of a 23 tree
{     slotType   data[KEY_NUM];               // a list of records sorted by keys
      struct nT  *link[PTR_NUM];              // a list of pointers
      struct nT  *parent;                     // a pointer to the parent node
}     nodeType;
typedef struct pointT                         // a point on the search path
{     nodeType   *pnode;                      // pointer to a parent node
      int        pidx;                        // entrance index on the parent node
}     pointType;
//***********************************************************************************************/
void search23tree(    );     // add one record into 23 tree
void searchPath(      );     // find a matched slot or the position to insert
//***********************************************************************************************/
vector<int> search23tree (  )                 // search 23 tree to find matches
{   stack<pointType>    aPath;                // stack to keep the search path
    vector<int>         ridS;

    searchPath(sKey, aPath);                  // find a matched entry on 23 tree
    if  (!aPath.empty())
    {   pointType   curP = aPath.top();       // reference to the last-visited node

        if (!sKey.compare(curP.pnode->data[curP.pidx].key))      // match or not?
            ridS = curP.pnode->data[curP.pidx].rSet;
                                  // get record identifiers on the matched entry
    }   // end outer if
    return ridS;
}   // end search23tree
```

```
void searchPath(        )     // find a matched position on 23 tree
{    // input: the root, a name; output: the search path
        pointType     oneP;
        int           pos;

        while    (cur != NULL)
     {     oneP.pnode = cur;
              for ( pos = 0; pos < KEY_NUM; pos++)
                if ((!cur->data[pos].rSet.size()) ||              // unused slot, name > key
              ((name.compare(cur->data[pos].key)) < 0))          // or name < key
                     break;                                      // search the next level
                else if (!name.compare(cur->data[pos].key))      // name == key (a duplicate!)
                     {        oneP.pidx = pos;                    // keep track of the pointer
                     path.push(oneP);                   // visited node: (parent node, entrance index)
                     return;                                     // the last-visited node is at the top of stack
              }        // end else
        oneP.pidx = pos;                                         // keep track of the pointer
        path.push(oneP);                                // visited node: (parent node, entrance index)
              cur = cur->link[pos];                              // recursive search at the next level
        }     // end while
}       // end searchPath
//***************************************************************************************************/
// Keep the above codes unchanged unless its correctness can be guaranteed.
//***************************************************************************************************/
```

```
//*****************************************************************************************/
//*** Header file for Search AVL Tree in Exercise 2 by Wu, Y.H.@CYCU-ICE
//*****************************************************************************************/
    vector<int> searchAVLtree(     );               // search AVL tree to find matches
//*****************************************************************************************/
vector<int> searchAVLtree( )                        // search AVL tree to find matches
{     vector<int> ridS;
      nodeType      *cur = root;                    // pointer to a node

      while (cur != NULL)                           // search a matched node until the bottom
      {     if (!sKey.compare(cur->key))            // found
            {     ridS = cur->rSet;
                  break;
            }
            else if (sKey.compare(cur->key) < 0)    // enter the left subtree
                  cur = cur->leftChild;
            else      cur = cur->rightChild;
      }     // end while
      return ridS;
}     // end searchAVLtree
//*****************************************************************************************/
// Keep the above codes unchanged unless its correctness can be guaranteed.
//*****************************************************************************************/
```