```
// 11027209 巫年巨
#include <stdio.h>
#include <iostream>
#include <vector>
#include <fstream>
#include <string>
#include <time.h>
#include <ctime>
#include <algorithm>
#include <cmath>
#include <stack>
#include <queue>
using namespace std;


int strtoi(string a){ // == atoi
   int i = 0, ans = 0;
   bool negative = 0, positive = 1, no_int = 0;
   for(i = 0 ; i < a.length() ; i++ ){
       if( a[0] == '-' )
         negative = 1;
       else if ( a[0] == '+')
          positive = 1;
       else if ( a[i] >= '0' && a[i] <= '9' ) {
          ans = ans * 10;
          ans = ans + a[i] -'0';
       } // else if    是 integer
       else if( a[i] >=    'a' && a[i] <= 'z' )
          return -1;
       else if( a[i] >=    'A' && a[i] <= 'Z' )
          return -1;
      else {    // other skip
          no_int = 1; // no integer
       } //
   } // for
   if ( negative == 1)
      return ans * -1; // negative integer
   else
```

```
    return ans; //    positive integer
} // strtoi


int strtoi2(string a){ // == atoi for    main input
   int i = 0, ans = 0;
   bool negative = 0, positive = 1, no_int = 0;
   for(i = 0 ; i < a.length() ; i++ ){
       if( a[0] == '-' )
         negative = 1;
       else if ( a[0] == '+')
          positive = 1;
       else if ( a[i] >= '0' && a[i] <= '9' ) {
          ans = ans * 10;
          ans = ans + a[i] -'0';
       } // else if    是 integer
      else {    // other skip
          return -1; // no integer
       } //
   } // for
   if ( negative == 1)
      return ans * -1; // negative integer
   else
      return ans; //    positive integer
} // strtoi2

string itostr(int a){ // string to int
   string str;
   int length = 1, num = 0 ;
   char c_num = 0;
   for (int i = a ; i/10 != 0    ; i = i/10)
      length++; // count the length of integer
   for( int i = length ; i > 0 ; i-- ) {
      num = a % 10;
      a = a / 10 ;
      c_num = num + '0';
      str.insert(str.begin(), c_num); // insert the integer to string
   } // for
```

```
    return str;
} // stoi

void Erase_Whitespace( string &s){
    for( int i = 0 ; i < s.length() ; i++){
        if ( s[i] == ' ' || s[i] == '\t' || s[i] == '\n' ) {
            s.erase(s.begin()+i); // erase it
            i = -1; // reloading i
        } // if
    }    // for
} // Erase_Whitespace

class CollegeType{
    private:
        int number;
        string college_code;
         string college_name;
        string department_code;
        string department_name;
        string sun_night;
        string level;
        int student_quantity;
        int teacher_quantity;
        int last_graduative_student_quantity;
        string city_name;
        string system;
    public:
        void Init(){
            number = 0;
            college_code.clear();
             college_name.clear();
            department_code.clear();
            department_name.clear();
            sun_night.clear();
            level.clear();
            student_quantity = 0;
            teacher_quantity = 0;
            last_graduative_student_quantity = 0;
```

```
    city_name.clear();
    system.clear();
} // Init

void Set_It( string test, int check ){ // set it to all
    if ( check == 0)
        college_code = test ;
     else if( check == 1)
         college_name = test ;
     else if( check == 2)
         department_code = test ;
     else if( check == 3)
         department_name = test ;
     else if( check == 4)
         sun_night = test;
     else if( check == 5)
         level = test;
     else if( check == 6)
         student_quantity = strtoi(test);
     else if( check == 7)
         teacher_quantity = strtoi(test);
     else if( check == 8)
        last_graduative_student_quantity = strtoi(test);
     else if( check == 9)
        city_name = test;
     else if( check == 10)
         system = test;
 } // Set_It

 void Set_num(int num){ // set 序號
    number = num ;
 } //
 string Get_cn(){
    return college_name;
 } //

 int Get_number(){
    return number;
```

```
        } //
        string Get_level(){
           return level;
        } //
        int Get_ls(){
           return last_graduative_student_quantity;
        } //
        string Get_sn(){
           return sun_night;
        } //

        string Get_dn(){
           return department_name;
        } //


}; // CollegeType

typedef struct slotT{ // a slot in a tree node
   vector<int> rSet;    // set of same key identifiers
   string key;
} slotType;

typedef struct nT{ // a tree node of a 23 tree
   slotType data[2]; // list of sort key
   struct nT *link[3]; // list of pointer
   struct nT *parent; // a pointer to parent node
} nodeType;

typedef struct pointT{ // a point on the search path
   nodeType *pnode; // pointer to parent node
   int pidx;  // entrance index on the parent node
} pointType;


typedef struct bT{ // a data block received from a split
   slotType slot;    // a pair of (record id, key)
   nodeType *link; // a pointer to a child on the right
```

```
} blockType;

typedef struct aN{
   int key;
   vector<int> rSet;
   aN *left;
   aN *right;
   aN *parent;
}avlNode;


class twothreetree{
   nodeType *root;
   public:
   twothreetree(){
      root = NULL;
   } //


   void caltreeH_and_node(int &height, int &node_num){        // calculate tree height
      height = 0;
      node_num = 0;
      int j = 0, k = 0, l = 0 ;
       if( root == NULL )
          return;
       queue<nodeType*> q; //  用 bfs 找樹高和節點數
       q.push(root);
       while( !q.empty()){
          int size = q.size();
          for( int i = 0 ; i < size ; i++ ){
             nodeType* temp = q.front();
             node_num++;
             q.pop();
             for( j = 0 ; j < 3 ; j++ ){
                if( temp->link[j] != NULL) //  有 child 的話
                    q.push(temp->link[j]); //  把其 child 放入
             } // for
          } // for
```

```
            height++;
        } // while
    } //

    void Get_RootSet( vector<int> &r1Set, vector<int> &r2Set){
        if( root == NULL )
            return;
        r1Set = root->data[0].rSet;
        r2Set = root->data[1].rSet;
    } // rootSet

    nodeType *createNode( nodeType * left ,    nodeType * right,    nodeType * pNode,
slotType newS ){ // create a node with one record
    // input:左小孩,右小孩,父節點,new record
    // output:  新節點  or NULL
        nodeType *newNode = NULL;
        try {
            newNode = new nodeType; // create a new node
            newNode->data[0].rSet = newS.rSet; // put the record into 1st slot
            newNode->data[1].rSet.clear();
            newNode->data[0].key = newS.key;
            newNode->data[1].key = "";
            newNode->parent = pNode; // set up link to the parent
            newNode->link[0] = left; // set up leftmost link
            newNode->link[1] = right;      // set up middle link
            newNode->link[2] = NULL; // set up rightmost link
        } // try

        catch(std::bad_alloc& ba){ // unable to allocate space
            std::cerr << endl << "bad_alloc caught." << ba.what() << endl;
        } // catch
        return newNode;
    } // createnode

    nodeType *createRoot( nodeType * & left , nodeType * &right, slotType &oneSlot ){
        nodeType *newRoot = createNode(left,right,NULL,oneSlot);
        left->parent = newRoot;
        right->parent = newRoot;
```

```
        return newRoot;
    } // createroot

    void insertLeaf( pointType aLeaf, slotType newS ){ // add a record into a leaf
    // input: a new slot(rSet,key),the leaf to insert(pnode,pidx)
    // output: leaf after update
       for( int i = 1 ; i >= aLeaf.pidx ; i-- ) {    //  從右到左 scan key num i = 2 - 1 = 1
          if( i > aLeaf.pidx ){ // shift an existing record to the right
             aLeaf.pnode->data[i].rSet = aLeaf.pnode->data[i-1].rSet;
             aLeaf.pnode->data[i].key = aLeaf.pnode->data[i-1].key;
          } // if
          else if ( i == aLeaf.pidx){
             aLeaf.pnode->data[i].rSet = newS.rSet; //save the new record in a new slot
               aLeaf.pnode->data[i].key = newS.key;
           } // else if
           else
              break;
       } // for
    } // insertLeaf


    void searchPath( nodeType *cur, string name, stack<pointType> &path    ){ // find a
matched position on 23 tree
    // input root, name , output, searchPath
       pointType oneP;
       int pos;
       while ( cur != NULL ){
          oneP.pnode = cur;
           for( pos = 0 ; pos < 2 ; pos++){    // 2 = key num   '
              if( ( !cur->data[pos].rSet.size() ) || ( name.compare(cur->data[pos].key) < 0 )    )
// unused slot, name > key||    name < key
                    break;
                else if ( !name.compare(cur->data[pos].key)) {// name == key( duplicated )
                    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
              } // else if
           } // for
```

```
        oneP.pidx = pos;
        path.push(oneP);
        cur = cur->link[pos];
      } // while
  } // searchPath

  void insertNonLeaf( blockType oneB, pointType goal ){ //  把 oneB 的 slot 放到 goal  並且把 link 放到該放的位置 link

     if( goal.pnode->data[0].key.compare(oneB.slot.key) < 0 ){ // goal 比分裂的中間小於 shift an existing record to the right
        goal.pnode->data[0].rSet = goal.pnode->data[1].rSet;   // shift an existing record to the right
        goal.pnode->data[0].key = goal.pnode->data[1].key;

        goal.pnode->data[0].rSet = oneB.slot.rSet; //save the new record in a new slot
        goal.pnode->data[0].key = oneB.slot.key;
     } // if
     else {     // goal 比分裂的中間大
        goal.pnode->data[1].rSet = oneB.slot.rSet; //save the new record in a new slot
        goal.pnode->data[1].key = oneB.slot.key;
      } // else if

     if(goal.pnode->link[2] == NULL ) { //  把原本的右 link 放到該放的位置去
        goal.pnode->link[2] = oneB.link;   //  最右 pointer 有東西嗎
        swap(goal.pnode->link[1],goal.pnode->link[2]); //  原本的右變成中間，中變成右邊
      } // if
      else {
        oneB.link->parent->data[1].rSet = oneB.link->data[0].rSet;
        oneB.link->parent->data[1].key = oneB.link->data[0].key;
        oneB.link->parent = NULL ;
     } //   回到原來的位置
  } // insertNonleaf

  void insert23tree( int newRid, string newKey ){
    slotType newSlot;
    newSlot.rSet.push_back(newRid);
    newSlot.key = newKey;
```

```
      if( root == NULL )
         root = createNode(NULL,NULL,NULL,newSlot);
      else {
         stack<pointType> aPath; // stack to keep serach path
         pointType curP; // last-visited node at the top of stack
         blockType blockUp; // a data block received from a spilt
         searchPath(root,newKey,aPath); // find a matched position on 23 tree
         if( !aPath.empty() ){
            curP = aPath.top(); // reference to the last-visited node
             if( ( curP.pnode->data[curP.pidx].rSet.size() ) &&
( !newKey.compare(curP.pnode->data[curP.pidx].key     ) )   )  // 是重複的 key(大學),所以
直接 insert
                curP.pnode->data[curP.pidx].rSet.push_back(newRid);
             else if ( !curP.pnode->data[1].rSet.size() ) // 至少有一個(最右)未使用的 slot
**********
                insertLeaf(curP,newSlot ); // add a record into a leaf
             else {   // split a full leaf
              splitLeaf(newSlot,curP,blockUp); // split a leaf for an insertion
               if( curP.pnode->parent == NULL ) // if root is split , create a new root
                  root = createRoot(curP.pnode, blockUp.link,blockUp.slot);
               else { //   else
                  do {
                     aPath.pop() ;
                     curP = aPath.top(); // the next parent for an insertion
                     if( !curP.pnode->data[1].rSet.size() ) { // 至少有一個(最右)未使用的 slot
**********
                        insertNonLeaf(blockUp,curP); //          add a slot into a non-leaf
                   break; // finish insertion
                     } // if
                     else {
                        splitNonLeaf(blockUp,curP); // split a non-leaf for an insertion
                         if( curP.pnode->parent == NULL ) { // if a root is split , create a new
root
                            root = createRoot(curP.pnode, blockUp.link,blockUp.slot);
                            break; // finish insertion
                         } // if
                     } // else
                   }while(true);
```

```
            } // else
          } // else
        } // if
      } // else
  } // insert23tree


  void splitLeaf( slotType newS, pointType aLeaf , blockType & aBlock){ // split a non-leaf
for an insertion
  // input: a new slot(rSet,key),the leaf to insert(pnode,pidx)
  // output: block after split to move upwards
     slotType buf[3]; // a buffer to keep a full node plus a new record
     int idx = 0; // index of the full node
     for(int i = 0 ; i < 3 ; i++ ){
       buf[i].rSet = ( i == aLeaf.pidx) ? newS.rSet : aLeaf.pnode->data[idx].rSet;
        // 上面那段是三元運算式  if( i == aLeaf.pidx )的話  buf[i].rSet = newS.key  不是的話
buf[i].rSet = aLeaf.pnode->data[idx].rSet
        buf[i].key = ( i == aLeaf.pidx) ? newS.key : aLeaf.pnode->data[idx++].key;
      } // for
      aLeaf.pnode->data[0].rSet = buf[0].rSet; // 留下最左紀錄
     aLeaf.pnode->data[0].key = buf[0].key;
     for( int i = 1; i < 2 ; i++ ){ // 剩下的 unused slots
       aLeaf.pnode->data[i].rSet.clear();
       aLeaf.pnode->data[i].key.clear();
     } // for
     aBlock.link = createNode(NULL,NULL,aLeaf.pnode->parent,buf[2]);
      aBlock.slot.rSet = buf[1].rSet;
      aBlock.slot.key = buf[1].key;
  } // splitLeaf


  void splitNonLeaf(    blockType   &oneB,   pointType goal   ){
      slotType buf[3]; // a buffer to keep a full node plus a new record
      nodeType *ptr[4]; // a buffer to keep pointers of children
      int idx = 0; // index of the full node
      for( int i = 0 ; i < 3 ; i++ ){
        ptr[i] =   goal.pnode->link[i]; // 紀錄指標
        buf[i].rSet = ( i == goal.pidx) ? oneB.slot.rSet : goal.pnode->data[idx].rSet; //
```

```
buffer 是排序過後的 set
            buf[i].key = ( i == goal.pidx) ? oneB.slot.key : goal.pnode->data[idx++].key;
         } // for
         goal.pnode->data[0].rSet = buf[0].rSet; //  留下最左紀錄
        goal.pnode->data[0].key = buf[0].key;
         goal.pnode->link[2] = NULL; //  右指標去除
         goal.pnode->link[1] = oneB.link; //  原本 oneB 的葉子值


       for( int i = 1; i < 2 ; i++ ){ //  剩下的 unused slots
          goal.pnode->data[i].rSet.clear();
          goal.pnode->data[i].key.clear();
       } //

         oneB.link = createNode(ptr[1],ptr[2],goal.pnode->parent,buf[2]); //  兄弟節點  非葉
的最大節點    接上中右鍵點
         oneB.slot.rSet = buf[1].rSet;
         oneB.slot.key = buf[1].key;
    } // splitNonLeaf

};

class avltree{
    avlNode *root;
    public:
    avltree(){
       root = NULL;
    } //

    avlNode* rotateLL( avlNode *x) {    // right rotate
       avlNode *y = x->left;
       x->left = y->right;
       y->right = x;
       y->parent = x->parent;
       x->parent = y;
       return y;
    } // rotateLL
```

```
avlNode* rotateLR( avlNode *x) { // left rotate then right rotate
   x->left = rotateRR(x->left);
   return rotateLL(x);
} // rotateLR

/* lR
   y = x-left;
   z = y->right;
   y->right = z->left;
   x->left = z->right;
   z->right = x;
   z->left = y;
   return y;
*/
avlNode* rotateRR( avlNode *x) {    //    right rotate
   avlNode *y = x->right;
   x->right = y->left;
   y->left = x;
   y->parent = x->parent;
   x->parent = y;
   return y;
} // rotateLL


avlNode* rotateRL( avlNode *x) {    // left rotate then right rotate
   x->right = rotateLL(x->right);
   return rotateRR(x);
} // rotateRL


int calTreeH( avlNode *node){       // calculate tree height
    if( node == NULL )
       return 0;
   return max(calTreeH(node->left), calTreeH(node->right)) + 1;
} //

int getBalanceFactor(avlNode* node) { //  得到平衡係數  BF
```

```
    if (node == NULL)
          return 0;
    return calTreeH(node->left) - calTreeH(node->right);
} // if

bool search( avlNode* &node, avlNode* &parent,    int key, int &LR ){
    if( node == NULL )
        return false;
     else if ( node->key == key )
        return true;
     parent = node;
     if ( node-> key > key ) {
        LR = 1;
        return search(node->left, parent , key, LR);
     } // if
     else if ( node-> key < key ){
        LR = 2;
        return search( node->right, parent , key, LR);
     }// else if
} // search

void insert_tree(int key, int number){
     int LR = 0;
     avlNode *temp = root;
     avlNode *parent = NULL;
    if ( search(temp, parent ,    key , LR) ) { // have key?
        temp->rSet.push_back(key);   // push back to tree
        temp->rSet.push_back(number);
        return;
    } // if
    else if(    ! search(temp, parent ,    key , LR ) ) { // no
        if( root == NULL ){
           root = createNode(key,number);
           return;
        } // if

        if( LR == 1 ) { // go left
           temp = createNode(key,number);
```

```
            parent->left = temp;
            temp->parent = parent;
          } // if

          else if( LR == 2 ) { // go right
            temp = createNode(key,number);
            parent->right = temp;
            temp->parent = parent;
          } // else if
        } // else if

        for( temp = parent ; temp != NULL ; temp = temp->parent) {
          int balance = getBalanceFactor(temp);
          if( balance >= 2 ){
            if( getBalanceFactor(temp->left) >= 0 ) //  做 LL       BF(x->left) = 0 or 1
              temp = rotateLL(temp);
            else if( getBalanceFactor(temp->left) < 0 )   //  做 LR     BF(x->left) = -1
               temp = rotateLR(temp);
          } // if


          else if ( balance <= -2) {
            if( getBalanceFactor(temp->right) <= 0 ) //  做 RR       BF(x->right) = 0 or -1
              temp = rotateRR(temp);
            else if( getBalanceFactor(temp->right) > 0 ) //  做 RL     BF(x->right) = 1
              temp = rotateRL(temp);
          } // else if
        } // for
    } // insert_tree

    avlNode *createNode( int key, int number){ //
        avlNode *newNode = NULL;
        try {
          newNode = new avlNode; // create a new node
          newNode->key = key;
          newNode->rSet.push_back(number);
          newNode->left = NULL;
          newNode->right = NULL;
```

```
        newNode->parent = NULL;
    } // try
    catch(std::bad_alloc& ba){ // unable to allocate space
        std::cerr << endl << "bad_alloc caught." << ba.what() << endl;
    } // catch
    return newNode;
} // createnode

void caltreeH_and_node(int &height, int &node_num){        // calculate tree height
   height = 0;
   node_num = 0;
   int j = 0, k = 0, l = 0 ;
    if( root == NULL )
        return;
    queue<avlNode*> q; //  用 bfs 找樹高和節點數
    q.push(root);
    while( !q.empty()){
        int size = q.size();
        for( int i = 0 ; i < size ; i++ ){
            avlNode* temp = q.front();
            q.pop();
            for(k = 0 ; k < temp->rSet.size() ; k++ )     //  算節點數量
               node_num++;
            if( temp->left != NULL) //  有 child 的話
                q.push(temp->left); //  把其 child 放入
             if(temp->right != NULL )
                q.push(temp->right);
        } // for
        height++;
    } // while
} // caltreeH_and_node


void Get_RootSet( vector<int> &r1Set){
   if( root == NULL )
        return;
    r1Set = root->rSet;
} // rootSet
```

```
};

class theCollegeList{
   vector<CollegeType> college_list ; // the set of CollegeType
   string fileID; // number of file ID
   twothreetree two_three;
   avltree avl;
   public:
      bool readF(int mission){ // read file
         this->ClearUp();
         int check = 0, i = 0, j = 0 ;
         string Fin,str,temp;
         while(check == 0){
            printf("\n Input a file number ([0] Quit): ");
             cin >> fileID; // cin in it
             cin.ignore(); // ignore the weird input
             Erase_Whitespace(fileID);
             if ( fileID == "0"    )     // out the loop
                return false;
             else {
                ifstream in;
                Fin = "input" + fileID + ".txt"; // input file name
                in.open(Fin.c_str());
                if (in) {
                     for( i = 0 ; ! in.eof() && i < 3 ; i++ ) // Read the three title
                      getline(in,str);
                   for( i = 0 ; ! in.eof() ; i++ ){ // read and write it to vector
                      getline(in,str);
                       if( str == "\n" || str == "\t" || str == "\0") // avoid generating a excess
saving
                          break;
                       college_list.push_back(CollegeType());    // put the all data to vector
                       college_list[i].Init();
                       college_list[i].Set_num(i+1); // set 序號
                       for( int j = 0 ; str.find('\t', 0) != -1; j++    ) { // split
                           temp = str.substr(0, str.find("     "));                      // the
                           str = str.substr(str.find("     ")+1, str.size() );    // string
```

```
                        college_list[i].Set_It(temp,j );                    // set it in the
vector
                        if( str.find('\t', 0) == -1 )                  // and final
                           college_list[i].Set_It(str,j+1 );                  // set it
                      } // for

                      if( mission == 1 ){
                        two_three.insert23tree( college_list[i].Get_number(),
college_list[i].Get_cn() ); //    cn 是學校名稱
                      } //

                      else if( mission == 2){
                        avl.insert_tree(college_list[i].Get_number(), college_list[i].Get_ls()); //
ls  是上學期畢業生人數
                      } //


                    } // for
                    check = 1;
                    in.close();
                 } // if
                else if (!in)
                   cout << ( "\n### "+ Fin +" does not exist ###\n" );
              }   // else
            } // while
            return true;
        } // readF()

       void ClearUp(){
          college_list.clear();
           fileID.clear();
        } // ClearUp(){




        void Print_all(int mission ){
           int height = 0;
```

```
        int node_num = 0;
        int i = 0,j = 0;
        vector<int> r1,r2;

        avl.Get_RootSet(r1);
        if( mission == 1){
           two_three.caltreeH_and_node(height, node_num);
           two_three.Get_RootSet(r1, r2);
         } // if

        else if( mission == 2){
             avl.Get_RootSet(r1);
             avl.caltreeH_and_node(height, node_num);
         } // else if

         cout << "Tree height = " << height << endl;
         cout << "Number of nodes = " << node_num << endl;
         for( i = 0 ; i < r1.size() ; i++ ){   // number 序號  cn 學校名稱  dn 科系名稱  sn 日夜別
level 等級別   ls 上學期畢業生人數
            cout << i+1 << ": [" << college_list[r1[i]-1].Get_number() << "] " <<
college_list[r1[i]-1].Get_cn() << ", " <<   college_list[r1[i]-1].Get_dn();
            cout << ", " << college_list[r1[i]-1].Get_sn() << ", " << college_list[r1[i]-
1].Get_level() << ", " <<   college_list[r1[i]-1].Get_ls() << endl ;
         } // for

         for( j = 0 ; j < r2.size() ; j++){
            i++;
            cout << i+1 << ": [" << college_list[r2[j]-1].Get_number() << "] " <<
college_list[r2[j]-1].Get_cn() << ", " <<   college_list[r2[j]-1].Get_dn();
            cout << ", " << college_list[r2[j]-1].Get_sn() << ", " << college_list[r2[j]-
1].Get_level() << ", " <<   college_list[r2[j]-1].Get_ls() << endl ;
         } // for

         cout << endl;
       } //


}; // the ClassList
```

```
class Mission{
   public:
      theCollegeList Heap_class;
      void mission1(){
         if( Heap_class.readF(1) ){
            Heap_class.Print_all(1);
          } // if
   } // mission1


   void mission2(){
      if( Heap_class.readF(2) ){
         Heap_class.Print_all(2);
       } // if
   } // mission 2


   void mission3(){

   } // mission 2
}; // Mission class


int main(){
   Mission mission_test;
   int command = -1, M = 0 ;
   string temp;
   while( command != 0 ) {
       printf("\n ** Search Tree Utilities ***\n ");
       printf("* 0. QUIT                          *\n ");
       printf("* 1. Build 2-3 tree              *\n ");
```

```
        printf("* 2. Build AVL tree              *\n ");
        printf("****************************\n ");
        printf("Input a choice(0,1,2): ");
       cin >>    temp;
       cin.ignore(); // remove the weird command for english input
       command = strtoi2(temp);
        if( command == 0 ) {
           break; // out the loop
        } // if
        else if( command == 1 ) {
           mission_test.mission1();
        } // else if

        else if( command == 2 ) {
          mission_test.mission2();
        } // else if

        else if( command == 3 ) {
        } // else if



        else {    //  無效指令
           printf("\n Command does not exist!\n");
        } // else    ( invalid command )

    }    // while     ( command == 0 stop the program )

    system("pause");
    return 0;
} // main
```