# 题目二：多叉树

## 头文件 tree.h

```cpp
#include <list>
#include <string>
#include <vector>

#ifndef __NODE_TREE_
#define __NODE_TREE_
#define _DEBUG

struct Node
{
// 假定不会存在相同节点，代码未作限定
public:
  Node(const int value):m_value(value) {}
  ~Node() {}
  Node& push_back(const Node& node)
  {
    child_list.push_back(node);
    return *this;
  }
  bool operator==(const Node& node) const { return this->m_value ==
node.m_value; };
  // 找到输出路径，没找到输出空字符串
  std::string Find(const Node& node, std::vector<int>& path) const;

private:
  int m_value;
  std::list<Node> child_list;
  std::string vec2string(const std::vector<int>& path) const;
};

#endif
```

## 源文件 tree.cpp

```cpp
#include "tree.h"
#include <iostream>
#include <sstream>
#include <assert.h>
#include <cstring>

using namespace std;

string Node::Find(const Node& node, vector<int>& path) const
{
  path.push_back(m_value);
  if (node == *this)
    return vec2string(path);
  else
  {
    list<Node>::const_iterator cit;
    //for(cit = child_list.cbegin(); cit != child_list.cend();
++cit)
    for(cit = child_list.begin(); cit != child_list.end(); ++cit)
    {
      string result = cit->Find(node, path);
      if (!result.empty())
        return result;
    }
  }
  path.pop_back();
  return "";
}

string Node::vec2string(const vector<int>& path) const
{
  ostringstream oss;
  vector<int>::const_iterator cit;
  //for(cit = path.cbegin(); cit != path.cend(); ++cit)
  for(cit = path.begin(); cit != path.end(); ++cit)
    oss << char(*cit);
  return oss.str();
}

// 初始化多叉树
Node Init()
```

```cpp
{
  Node A('A');
  Node B('B');
  Node D('D');
  Node F('F');
  Node H('H');
  Node N('N');
  N.push_back('R').push_back('S').push_back('T');
  H.push_back('O').push_back('P').push_back('Q');
  F.push_back('K').push_back('L').push_back('M').push_back(N);
  D.push_back(H).push_back('I').push_back('J');
  B.push_back('E').push_back(F).push_back('G');
  A.push_back(B).push_back('C').push_back(D);
  return A;
}

string x2y(const string& xp, const string& yp, const string&
separator="->")
{
  string str;
  int xl = xp.length();
  int yl = yp.length();
  int father_index;
  for(int i=0; i< xl && i<yl; ++i)
    if(xp.at(i) == yp.at(i))
       father_index = i;
    else
      break;

  for(int i=xl-1; i>father_index; --i)
    str += xp.at(i);

  for(int i=father_index; i<yl; ++i)
    str += yp.at(i);

  ostringstream oss;
  int length = str.length();
  oss << length << ":";
  for(int i=0;i<length;++i)
    oss << str.at(i) << ((i+1 == length) ? "" : separator);
  return oss.str();
}

int main(int argc, char** argv)
```

```
{
  assert(argc == 3);
  if (argc > 1)
    assert(strlen(argv[1]) == 1);
  if (argc > 2)
    assert(strlen(argv[2]) == 1);

  // 初始化多叉树
  Node head = Init();
  char _x,_y;
  _x = argv[1][0];
  _y = argv[2][0];
  //assert(_x != _y);

  vector<int> path;
  string pathX = head.Find(_x, path);
  path.clear();
  string pathY = head.Find(_y, path);
  cout << pathX << endl << pathY << endl;
  if (pathX.empty() || pathY.empty())
  {
    cerr << "多叉树中不存在输入的某个节点" << endl;
    return 0;
  }
  // 截去两个字符串的公共部分
  cout << x2y(pathX, pathY) << endl;
  return 0;
}
```

# 题目三：Cache 模拟

## 头文件 Item.h

```
#include <stdlib.h>
#include <vector>

#ifndef _CACHE_ITEM__
#define _CACHE_ITEM__

const int _AGE_LIMIT = 10; // 10
```

```cpp
const int _COUNT_LIMIT = 100; // 100
const int _TIMES_LOOP = 200;  // 200

// 用于产生随机数，方案二
static std::vector<int> Zero2Count;

class Item
{
public:
  // 默认在单向链表的头部插入
  Item();
  // 选择位置插入
  Item(const int random , const int age=0);
  ~Item();
public:
  // 返回值：通过判断 age > 10 淘汰掉的 item 的数目
  static int Scan();
  static void Print() ;
  static bool IsFull() { return m_count >= _COUNT_LIMIT;}
  static int GetCount() { return m_count;}

private:
  int m_id;
  int m_age;
  Item* m_next;
  bool AgeUp() { return ( ++m_age > _AGE_LIMIT); }
private:
  static int m_count; // 有效元素个数
  static int m_index; // 所有创建过的元素个数（包括销毁了的）
  static Item* m_head;
};

#endif
```

## 源文件 item.cpp

```cpp
#include <iostream>
#include <algorithm>
#include <vector>
#include <string>
#include <stdlib.h>
```

```cpp
#include <time.h>
#include <assert.h>
#include "Item.h"

using namespace std;

int Item::m_count;
int Item::m_index;
Item* Item::m_head;

Item::Item():m_id(m_index++),m_age(0), m_next(m_head)
{
  // TODO 构造中抛出异常?
  assert(m_count < _COUNT_LIMIT);
  m_head = this;
  ++m_count;
}

Item::Item(const int random , const int
age):m_id(m_index++),m_age(age)
{
  assert(m_count < _COUNT_LIMIT);
  int pos = random % (m_count+1);
  if (m_count == 0 || pos == 0)
  {
    m_next = m_head;
    m_head = this;
  }
  else
  {
    Item* pre = NULL;
    Item* tmp = m_head;
    for (int i=0;i<pos;++i)
    {
      pre = tmp;
      tmp = tmp->m_next;
    }
    m_next = tmp;
    if (pre != NULL)
      pre->m_next = this;
  }

  ++m_count;
}
```

```cpp

Item::~Item()
{
  if (m_head == this)
    m_head = this->m_next;
  else
  {
    Item* tmp, *pre;
    pre = tmp = m_head;
    while(tmp != this)
    {
      pre = tmp;
      tmp = tmp->m_next;
    }
    if (pre != NULL)
      pre->m_next = this->m_next;
  }
  --m_count;
}

void Item::Print()
{
  printf("[id(age)]: ");
  for (Item* tmp= m_head; tmp!=NULL; tmp = tmp->m_next)
    printf("%d(%d) ", tmp->m_id, tmp->m_age);
  printf("(live[%d], total[%d])\n", Item::m_count, Item::m_index);
}

int Item::Scan()
{
  int count = 0;
  Item* earliest;
  for (Item* tmp= m_head; tmp!=NULL; tmp = tmp->m_next)
  {
    earliest= tmp;
    if(tmp->AgeUp())
    {
      printf("age 到期，淘汰 item [id=%d] 。\n", tmp->m_id);
      delete tmp;
      ++count;
    }
  }
  // TODO > 至少淘汰一个 与之后的淘汰条件并不等价
  if(count == 0 && m_count >= _COUNT_LIMIT)
  {
    // TODO 第一个到底只收尾哪个?
    // earliest = m_head;
    printf("Cache 已满，淘汰第一个 item[id=%d] 。\n", earliest->m_id);
    delete earliest;
  }
  return count;
}

int randperm()
{
  random_shuffle(Zero2Count.begin(), Zero2Count.end());
  return Zero2Count.at(0);
}

int NewRandom()
{
  // 产生随机数方案一
  srand((unsigned)time(NULL));
  int random =  rand();
  //cout << "rand() returns: " << random << endl;
  for(int i=0;i<random%3 +1;++i)
    if (Item::IsFull() == false)
    {
      int value = randperm();
      new Item(value);
      int count = Item::GetCount();
      fprintf(stdout, "Cache [%d]位置新增 item 成功。\n", value %
count);
    }
    else
      cerr << "Cache 已满，新增失败。" << endl;
}

int Init50(const int count=50)
{
  for(int i=0;i<count;++i)
    if (Item::IsFull() == false)
      new Item;
}

int main()
{
  for (int i = 0; i <= _COUNT_LIMIT; ++i)
```

```
    Zero2Count.push_back(i);
  // 已有 50 个 Item
  Init50();
  for(int i=0;i<_TIMES_LOOP;++i)
  {
    sleep(1);
    // 新增
    NewRandom();
    cout << "新增之后 && 淘汰之前:" << endl;
    Item::Print();
    Item::Scan();
    cout << "淘汰之后:" << endl;
    Item::Print();
    cout << "=================" << endl;
  }
  return 0;
}
```