# 087和088是在一起的,我们一起做笔记

# 学习大纲

|  |
| --- |
|  |

# 目标:练习使用多个堆进行程序设计的技巧

## GetExceptionCode宏的语法

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 语法 C++复制  DWORD GetExceptionCode(void); parameters 此宏没有参数。 返回值 返回值标识异常的类型。 下表标识了由于常见编程错误而可能发生的异常代码。 这些值在 WinBase.h 和 WinNT.h 中定义。  展开表   | **返回代码** | **说明** | | --- | --- | | **EXCEPTION\_ACCESS\_VIOLATION** | 线程尝试从其无权访问的虚拟地址进行读取或写入。 此值定义为STATUS\_ACCESS\_VIOLATION。 | | **EXCEPTION\_ARRAY\_BOUNDS\_EXCEEDED** | 线程尝试访问超出边界的数组元素，并且基础硬件支持边界检查。 此值定义为STATUS\_ARRAY\_BOUNDS\_EXCEEDED。 | | **EXCEPTION\_BREAKPOINT** | 遇到断点。 此值定义为STATUS\_BREAKPOINT。 | | **EXCEPTION\_DATATYPE\_MISALIGNMENT** | 线程尝试读取或写入未提供对齐的硬件上未对齐的数据。 例如，16 位值必须在 2 字节边界上对齐，在 4 字节边界上对齐 32 位值，等等。 此值定义为 STATUS\_DATATYPE\_MISALIGNMENT。 | | **EXCEPTION\_FLT\_DENORMAL\_OPERAND** | 浮点运算中的一个操作数是反常运算。 非正态值太小，无法表示为标准浮点值。 此值定义为STATUS\_FLOAT\_DENORMAL\_OPERAND。 | | **EXCEPTION\_FLT\_DIVIDE\_BY\_ZERO** | 线程尝试将浮点值除以浮点除数 0 (零) 。 此值定义为STATUS\_FLOAT\_DIVIDE\_BY\_ZERO。 | | **EXCEPTION\_FLT\_INEXACT\_RESULT** | 浮点运算的结果不能完全表示为小数点。 此值定义为STATUS\_FLOAT\_INEXACT\_RESULT。 | | **EXCEPTION\_FLT\_INVALID\_OPERATION** | 此列表中未包括的浮点异常。 此值定义为STATUS\_FLOAT\_INVALID\_OPERATION。 | | **EXCEPTION\_FLT\_OVERFLOW** | 浮点运算的指数大于相应类型允许的量级。 此值定义为STATUS\_FLOAT\_OVERFLOW。 | | **EXCEPTION\_FLT\_STACK\_CHECK** | 由于浮点操作，堆栈已溢出或不足。 此值定义为STATUS\_FLOAT\_STACK\_CHECK。 | | **EXCEPTION\_FLT\_UNDERFLOW** | 浮点运算的指数小于相应类型允许的幅度。 此值定义为STATUS\_FLOAT\_UNDERFLOW。 | | **EXCEPTION\_GUARD\_PAGE** | 线程访问了使用 PAGE\_GUARD 修饰符分配的内存。 此值定义为STATUS\_GUARD\_PAGE\_VIOLATION。 | | **EXCEPTION\_ILLEGAL\_INSTRUCTION** | 线程尝试执行无效指令。 此值定义为STATUS\_ILLEGAL\_INSTRUCTION。 | | **EXCEPTION\_IN\_PAGE\_ERROR** | 线程尝试访问不存在的页面，并且系统无法加载该页。 例如，如果在通过网络运行程序时网络连接断开，则可能会发生此异常。 此值定义为STATUS\_IN\_PAGE\_ERROR。 | | **EXCEPTION\_INT\_DIVIDE\_BY\_ZERO** | 线程尝试将整数值除以 0 (零) 的整数除数。 此值定义为STATUS\_INTEGER\_DIVIDE\_BY\_ZERO。 | | **EXCEPTION\_INT\_OVERFLOW** | 整数运算的结果会创建一个太大而无法由目标寄存器保留的值。 在某些情况下，这将导致执行结果中最重要的位。 某些操作不设置携带标志。 此值定义为 STATUS\_INTEGER\_OVERFLOW。 | | **EXCEPTION\_INVALID\_DISPOSITION** | 异常处理程序将无效处置返回到异常调度程序。 使用高级语言（如 C）的程序员不应遇到此异常。 此值定义为 STATUS\_INVALID\_DISPOSITION。 | | **EXCEPTION\_INVALID\_HANDLE** | 线程对内核对象使用了句柄，该句柄 (可能是因为已关闭。) 此值定义为 STATUS\_INVALID\_HANDLE。 | | **EXCEPTION\_NONCONTINUABLE\_EXCEPTION** | 线程在发生不连续异常后尝试继续执行。 此值定义为STATUS\_NONCONTINUABLE\_EXCEPTION。 | | **EXCEPTION\_PRIV\_INSTRUCTION** | 线程尝试使用当前计算机模式中不允许的操作执行指令。 此值定义为STATUS\_PRIVILEGED\_INSTRUCTION。 | | **EXCEPTION\_SINGLE\_STEP** | 跟踪陷阱或其他单一指令机制发出信号，指示执行了一条指令。 此值定义为STATUS\_SINGLE\_STEP。 | | **EXCEPTION\_STACK\_OVERFLOW** | 线程会占用其堆栈。 此值定义为STATUS\_STACK\_OVERFLOW。 | | **STATUS\_UNWIND\_CONSOLIDATE** | 已执行帧合并。 |    备注 **GetExceptionCode** 函数只能从异常处理程序的筛选器表达式或异常处理程序块中调用。 如果在执行 **\_\_try** 块期间发生异常，则计算筛选器表达式，并确定是否执行 **\_\_except** 块。  筛选器表达式可以调用筛选器函数。 筛选器函数不能调用 **GetExceptionCode**。 但是， **GetExceptionCode** 的返回值可以作为参数传递给筛选器函数。 [**GetExceptionInformation**](https://learn.microsoft.com/zh-cn/windows/win32/debug/getexceptioninformation) 函数的返回值也可以作为参数传递给筛选器函数。 **GetExceptionInformation** 返回指向包含异常代码信息的结构的指针。  当存在嵌套处理程序时，将计算每个筛选器表达式，直到将一个筛选器表达式计算为EXCEPTION\_EXECUTE\_HANDLER或EXCEPTION\_CONTINUE\_EXECUTION。 每个筛选器表达式都可以调用 **GetExceptionCode** 来获取异常代码。  返回的异常代码是硬件异常生成的代码，或在 [**RaiseException**](https://learn.microsoft.com/zh-cn/windows/win32/api/errhandlingapi/nf-errhandlingapi-raiseexception) 函数中为软件生成的异常指定的代码。  处理断点异常时，请务必递增上下文记录中的指令指针，以继续此异常。 示例 有关示例，请参阅 [使用异常处理程序](https://learn.microsoft.com/zh-cn/windows/win32/debug/using-an-exception-handler)。 要求 展开表   | **要求** | **值** | | --- | --- | | 最低受支持的客户端 | Windows XP [仅限桌面应用] | | 最低受支持的服务器 | Windows Server 2003 [仅限桌面应用] |  请参阅 [**GetExceptionInformation**](https://learn.microsoft.com/zh-cn/windows/win32/debug/getexceptioninformation)  [**RaiseException**](https://learn.microsoft.com/zh-cn/windows/win32/api/errhandlingapi/nf-errhandlingapi-raiseexception)  [结构化异常处理函数](https://learn.microsoft.com/zh-cn/windows/win32/debug/structured-exception-handling-functions)  [结构化异常处理概述](https://learn.microsoft.com/zh-cn/windows/win32/debug/structured-exception-handling) |

## 异常常量

|  |
| --- |
| /\*  \* Legal values for expression in except().  \*/  #define EXCEPTION\_EXECUTE\_HANDLER 1  #define EXCEPTION\_CONTINUE\_SEARCH 0  #define EXCEPTION\_CONTINUE\_EXECUTION -1 |

# 自定义函数语法说明

## 自定义结构体TREENODE

|  |
| --- |
| //定义节点数据类型  typedef struct \_TREENODE  {  struct \_TREENODE \*Left,\*Right;//二叉树的左右节点指针  TCHAR key[KEYSIZE];  LPTSTR pData;//指向数据堆的指针    }TREENODE,\*LPTNODE,\*\*LPPTNODE; |

## FillTree函数的用法

|  |
| --- |
| LPTNODE FillTree(HANDLE,HANDLE,HANDLE); |

## InsertTree函数的用法

|  |
| --- |
| BOOL InsertTree(LPPTNODE, LPTNODE); |

## KeyCompare函数的用法

|  |
| --- |
| int KeyCompare(LPTSTR,LPTSTR); |

## Scan函数的用法

|  |
| --- |
|  |

# 演练

# 087节

## 1.新建一文件夹取名:Lesson87-88-SortBt,然后在这里新建一个win32控制台空项目,取名SortBt ,然后新建一个cpp文件,取名:SortBt.cpp

## 2.在cpp的源文件路径里面新建一个demo1.txt文本文件,内容如下

|  |
| --- |
| linodnqb. 保信人寿  lamaigze. 池春建设  lazykgqe. 春永建设  specdmgp. 赐芳股份  ranchizl. 大东补习班  savearos. 大钰贸易  romeyapc. 德话食品  piccohej. 顶上系统  tomspdic. 东帝望  anatruwk. 东南实业  galedewa. 东旗  suprdumn. 福星制衣厂股份有限公司  lilasrbv. 福泰人寿  nortslfg. 富同企业  quicklha. 高上补习班  bsbehpi. 光明杂志  grosrovz. 光远商贸  bottmdbj. 广通  aroutlon. 国顶有限公司  blonpnrd. 国皓  franrwcd. 国银贸易  wolzayep. 汉典电机  grealhmv. 仪和贸易  whitcaur. 倚天文化事业  ricsuvjc. 永大企业  koenelnv. 永业房屋  frankxfv. 友恒信托  ricarfgx. 宇欣实业  hilaaosf. 远东开发  maisdnyn. 悦海  ernshmdm.正人资源  lonepwwt. 正太实业  wilmkuhr. 志远有限公司  vaffekut. 中硕贸易  eastcpdh. 中通  morgkiqw. 仲堂企业 |

## 然后再创建一个demo2.txt,内容如下

|  |
| --- |
| e0454199. Record Number: 00000000.abcdefghijklmnopqrstuvwxyz x  2ccfbc5e. Record Number: 00000001.abcdefghijklmnopqrstuvwxyz x  ee8c083f. Record Number: 00000002.abcdefghijklmnopqrstuvwxyz x  9bdce50c. Record Number: 00000003.abcdefghijklmnopqrstuvwxyz x  de1f5e55. Record Number: 00000004.abcdefghijklmnopqrstuvwxyz x  acd7406a. Record Number: 00000005.abcdefghijklmnopqrstuvwxyz x  e942e95b. Record Number: 00000006.abcdefghijklmnopqrstuvwxyz x  4bf945f8. Record Number: 00000007.abcdefghijklmnopqrstuvwxyz x  62448ad1. Record Number: 00000008.abcdefghijklmnopqrstuvwxyz x  2820f936. Record Number: 00000009.abcdefghijklmnopqrstuvwxyz x  97bcc037. Record Number: 00000010.abcdefghijklmnopqrstuvwxyz x  bf78c9a4. Record Number: 00000011.abcdefghijklmnopqrstuvwxyz x  ecb6030d. Record Number: 00000012.abcdefghijklmnopqrstuvwxyz x  e77b72c2. Record Number: 00000013.abcdefghijklmnopqrstuvwxyz x  309328d3. Record Number: 00000014.abcdefghijklmnopqrstuvwxyz x  ab4adc10. Record Number: 00000015.abcdefghijklmnopqrstuvwxyz x  024cc309. Record Number: 00000016.abcdefghijklmnopqrstuvwxyz x  8011f90c. Record Number: 00000017.abcdefghijklmnopqrstuvwxyz x  305d7f2f. Record Number: 00000018.abcdefghijklmnopqrstuvwxyz x  61e9a93c. Record Number: 00000019.abcdefghijklmnopqrstuvwxyz x  027586c5. Record Number: 00000020.abcdefghijklmnopqrstuvwxyz x |

## 3.我们先在main函数里面编写一些代码

|  |
| --- |
| #include<Windows.h>  #include<stdio.h>  #include<stdlib.h>  #define KEYSIZE 10 //KEY的大小是8个字符,还有一个.而且末尾需要添加一个'\0'    //定义节点数据类型  typedef struct \_TREENODE  {  struct \_TREENODE \*Left,\*Right;//二叉树的左右节点指针  TCHAR key[KEYSIZE];  LPTSTR pData;//指向数据堆的指针  }TREENODE,\*LPTNODE,\*\*LPPTNODE;  #define CR 0x0D  #define LF 0x0A  #define NODE\_HEAP\_SIZE 0x8000 //定义堆大小  #define DATA\_HEAP\_SIZE 0x8000  #define MAX\_DATA\_LEN 0x1000  #define NODE\_SIZE sizeof(TREENODE)  #define TKEYSIZE KEYSIZE \* sizeof(TCHAR)  #define TSIZE sizeof(TCHAR)  LPTNODE FillTree(HANDLE,HANDLE,HANDLE);  int main(int argc,LPTSTR argv[])  {  HANDLE hIn = INVALID\_HANDLE\_VALUE;  HANDLE hNode = NULL;//节点堆  HANDLE hData = NULL;//数据堆  LPTNODE pRoot;  if(argc < 2)  {  printf(TEXT("Usage: Lesson87-SortBt1 file1 file2 ...\n"));  return -1;  }  for(int iFile =1;iFile<argc;iFile++)  {  printf(TEXT("%s\n"),argv[iFile]);  hIn = CreateFile(argv[iFile],GENERIC\_READ,FILE\_SHARE\_READ,NULL,OPEN\_EXISTING,FILE\_ATTRIBUTE\_NORMAL,NULL);  if(hIn == INVALID\_HANDLE\_VALUE)  {  printf(TEXT("打开文件%s失败,错误码:%d\n"),argv[iFile],GetLastError());  return 1;  }  hNode = HeapCreate(HEAP\_GENERATE\_EXCEPTIONS|HEAP\_NO\_SERIALIZE,NODE\_HEAP\_SIZE,0);//创建节点堆  hData = HeapCreate(HEAP\_GENERATE\_EXCEPTIONS|HEAP\_NO\_SERIALIZE,DATA\_HEAP\_SIZE,0);//创建数据堆  pRoot = FillTree(hIn,hNode,hData);    }  //system("pause");  return 0;  } |

## 4.然后我们来实现FillTree自定义函数,我们先看看能否把文件里面的数据读取并且显示出来,初步的代码如下

|  |
| --- |
| LPTNODE FillTree(HANDLE hIn,HANDLE hNode,HANDLE hData)  {  LPTNODE pRoot=NULL,pNode;  DWORD dwRead;  TCHAR valueData[MAX\_DATA\_LEN];//存放数据的缓冲区  BOOL isCR; //是否是回车的标记  DWORD i=0;  while(TRUE)  {  pNode = (LPTNODE)HeapAlloc(hNode,HEAP\_ZERO\_MEMORY,NODE\_SIZE);  //读取节点名称  if(!ReadFile(hIn,pNode->key,TKEYSIZE,&dwRead,NULL)||dwRead!=TKEYSIZE)  return pRoot;  pNode->key[KEYSIZE-1]='\0';  isCR = FALSE;  for(i=0;i<MAX\_DATA\_LEN;i++)  {  ReadFile(hIn,&valueData[i],TSIZE,&dwRead,NULL);//用win32Api读取一个字符的写法  //因为我们不知道value的具体长度又不想浪费内存,所以需要一个一个字符来读取,每一次都需要判断是否是回车符号,如果是  //回车,并且它的下一个是换行换行,就跳出循环  if(isCR && valueData[i] == LF) break;  isCR = (valueData[i] == CR);    }  valueData[i-1] = '\0';  printf(TEXT("%s: %s\n"),pNode->key,valueData);  }    } |

### 以上就是087节的内容

# 088节,下面我们接着编写程序,这一节需要在节点对里面创建二叉树和把数据放到数据堆里面

## 5.我们需要继续完善FillTree函数,代码如下

|  |
| --- |
|  |
| LPTNODE FillTree(HANDLE hIn,HANDLE hNode,HANDLE hData)  {  LPTNODE pRoot=NULL,pNode;  DWORD dwRead;  TCHAR valueData[MAX\_DATA\_LEN];//存放数据的缓冲区  BOOL isCR; //是否是回车的标记  DWORD i=0;  //088  LPTSTR lpStr;  while(TRUE)  {  pNode = (LPTNODE)HeapAlloc(hNode,HEAP\_ZERO\_MEMORY,NODE\_SIZE);  //读取节点名称  if(!ReadFile(hIn,pNode->key,TKEYSIZE,&dwRead,NULL)||dwRead!=TKEYSIZE)  return pRoot;  pNode->key[KEYSIZE-1]='\0';  isCR = FALSE;  for(i=0;i<MAX\_DATA\_LEN;i++)  {  ReadFile(hIn,&valueData[i],TSIZE,&dwRead,NULL);//用win32Api读取一个字符的写法  //因为我们不知道value的具体长度又不想浪费内存,所以需要一个一个字符来读取,每一次都需要判断是否是回车符号,如果是  //回车,并且它的下一个是换行换行,就跳出循环  if(isCR && valueData[i] == LF) break;  isCR = (valueData[i] == CR);    }  valueData[i-1] = '\0';  //088 在数据堆里面分配内存,注意,我们把key保存在节点堆里面,把key和值一整行保存在数据堆里面  lpStr = (LPTSTR)HeapAlloc(hData,HEAP\_NO\_SERIALIZE|HEAP\_ZERO\_MEMORY,(SIZE\_T)(KEYSIZE+\_tcslen(valueData)+1)\*TSIZE);  memcpy(lpStr,pNode->key,TKEYSIZE);//先拷贝key  lpStr[KEYSIZE]='\0';  \_tcscat(lpStr,valueData); //然后在后面拼接值  //需要让节点的一个指针指向数据堆  pNode->pData = lpStr;  //节点插入二叉树,调用InsertTree函数,我们需要实现这个函数  InsertTree(&pRoot,pNode);  //memcpy(lpStr,valueData,KEYSIZE+\_tcslen(valueData)+1);  printf(TEXT("%s: %s\n"),pNode->key,valueData);  }    } |

## 6.接着我们需要实现InsertTree自定义函数

|  |
| --- |
|  |
| BOOL InsertTree(LPPTNODE ppRoot, LPTNODE pNode)  {  //如果二叉树是空的,那么插入的节点就可以中文根节点  if(\*ppRoot == NULL)  {  \*ppRoot = pNode;  return TRUE;  }  //如果已经有根节点,需要判断这个节点是比根节点大还是比根节点小  //如果小在根节点的左边,大就在根节点的右边  if(KeyCompare(pNode->key,(\*ppRoot)->key)<0) //KeyCompare也是自定义函数,也需要自己来写  InsertTree(&((\*ppRoot)->Left),pNode);//这里需要使用递归调用  else  InsertTree(&((\*ppRoot)->Right),pNode);  } |

## 7.接着我们来实现KeyCompare函数,它很简单其实就是调用c语言函数

|  |
| --- |
| static int KeyCompare(LPTSTR key1,LPTSTR key2)//静态函数  {  return \_tcsncmp(key1,key2,KEYSIZE);  } |

## 8.然后我们可以查看一排序后是文件内容,需要在main函数里面添加遍历代码,这里有一个自定义Scan函数

|  |
| --- |
| #include<Windows.h>  #include<stdio.h>  #include<stdlib.h>  #include<tchar.h>  #define KEYSIZE 10 //KEY的大小是8个字符,还有一个.而且末尾需要’\0’  //定义节点数据类型  typedef struct \_TREENODE  {  struct \_TREENODE \*Left,\*Right;//二叉树的左右节点指针  TCHAR key[KEYSIZE];  LPTSTR pData;//指向数据堆的指针  }TREENODE,\*LPTNODE,\*\*LPPTNODE;  #define CR 0x0D  #define LF 0x0A  #define NODE\_HEAP\_SIZE 0x8000 //定义堆大小  #define DATA\_HEAP\_SIZE 0x8000  #define MAX\_DATA\_LEN 0x1000  #define NODE\_SIZE sizeof(TREENODE)  #define TKEYSIZE KEYSIZE \* sizeof(TCHAR)  #define TSIZE sizeof(TCHAR)  LPTNODE FillTree(HANDLE,HANDLE,HANDLE);  BOOL InsertTree(LPPTNODE, LPTNODE);//088  int KeyCompare(LPTSTR,LPTSTR);  BOOL Scan(LPTNODE);  int main(int argc,LPTSTR argv[])  {  HANDLE hIn = INVALID\_HANDLE\_VALUE;  HANDLE hNode = NULL;//节点堆  HANDLE hData = NULL;//数据堆  LPTNODE pRoot;  if(argc < 2)  {  printf(TEXT("Usage: Lesson87-SortBt1 file1 file2 ...\n"));  return -1;  }  for(int iFile =1;iFile<argc;iFile++)  {  printf(TEXT("%s\n"),argv[iFile]);  hIn = CreateFile(argv[iFile],GENERIC\_READ,FILE\_SHARE\_READ,NULL,OPEN\_EXISTING,FILE\_ATTRIBUTE\_NORMAL,NULL);  if(hIn == INVALID\_HANDLE\_VALUE)  {  printf(TEXT("打开文件%s失败,错误码:%d\n"),argv[iFile],GetLastError());  return 1;  }  hNode = HeapCreate(HEAP\_GENERATE\_EXCEPTIONS|HEAP\_NO\_SERIALIZE,NODE\_HEAP\_SIZE,0);//创建节点堆  hData = HeapCreate(HEAP\_GENERATE\_EXCEPTIONS|HEAP\_NO\_SERIALIZE,DATA\_HEAP\_SIZE,0);//创建数据堆  pRoot = FillTree(hIn,hNode,hData);  \_tprintf(\_T("排序后的文件%s\n"), argv[iFile]);//088  //遍历二叉树  Scan(pRoot);//这个也是一个自定义函数需要自己编写  }  //system("pause");  return 0;  } |

## 9.然后我们需要实现这个函数

|  |
| --- |
| //遍历二叉树  static BOOL Scan(LPTNODE pNode) //静态函数只能够在本模块里面使用,外面不能使用  {  //如果二叉树是空的,就直接返回  if(pNode == NULL) return TRUE;  Scan(pNode->Left);//递归用法,扫描左子树  \_tprintf(\_T("%s\n"),pNode->pData);//输出数据堆里面的数据  Scan(pNode->Right);  return TRUE;  } |

## 10.完成后的代码

### SortBt.cpp

|  |
| --- |
| #include<Windows.h>  #include<stdio.h>  #include<stdlib.h>  #include<tchar.h>  #define KEYSIZE 10 //KEY的大小是8个字符,还有一个.而且末尾需要添加一个'\0'  //定义节点数据类型  typedef struct \_TREENODE  {  struct \_TREENODE \*Left,\*Right;//二叉树的左右节点指针  TCHAR key[KEYSIZE];  LPTSTR pData;//指向数据堆的指针  }TREENODE,\*LPTNODE,\*\*LPPTNODE;  #define CR 0x0D  #define LF 0x0A  #define NODE\_HEAP\_SIZE 0x8000 //定义堆大小  #define DATA\_HEAP\_SIZE 0x8000  #define MAX\_DATA\_LEN 0x1000  #define NODE\_SIZE sizeof(TREENODE)  #define TKEYSIZE KEYSIZE \* sizeof(TCHAR)  #define TSIZE sizeof(TCHAR)  LPTNODE FillTree(HANDLE,HANDLE,HANDLE);  BOOL InsertTree(LPPTNODE, LPTNODE);//088  int KeyCompare(LPTSTR,LPTSTR);  BOOL Scan(LPTNODE);  int main(int argc,LPTSTR argv[])  {  HANDLE hIn = INVALID\_HANDLE\_VALUE;  HANDLE hNode = NULL;//节点堆  HANDLE hData = NULL;//数据堆  LPTNODE pRoot;  if(argc < 2)  {  printf(TEXT("Usage: Lesson87-SortBt1 file1 file2 ...\n"));  return -1;  }  for(int iFile =1;iFile<argc;iFile++)  {  printf(TEXT("%s\n"),argv[iFile]);  hIn = CreateFile(argv[iFile],GENERIC\_READ,FILE\_SHARE\_READ,NULL,OPEN\_EXISTING,FILE\_ATTRIBUTE\_NORMAL,NULL);  if(hIn == INVALID\_HANDLE\_VALUE)  {  printf(TEXT("打开文件%s失败,错误码:%d\n"),argv[iFile],GetLastError());  return 1;  }  hNode = HeapCreate(HEAP\_GENERATE\_EXCEPTIONS|HEAP\_NO\_SERIALIZE,NODE\_HEAP\_SIZE,0);//创建节点堆  hData = HeapCreate(HEAP\_GENERATE\_EXCEPTIONS|HEAP\_NO\_SERIALIZE,DATA\_HEAP\_SIZE,0);//创建数据堆  pRoot = FillTree(hIn,hNode,hData);  \_tprintf(\_T("排序后的文件%s\n"), argv[iFile]);//088  //遍历二叉树  Scan(pRoot);//这个也是一个自定义函数需要自己编写  }  //system("pause");  return 0;  }  LPTNODE FillTree(HANDLE hIn,HANDLE hNode,HANDLE hData)  {  LPTNODE pRoot=NULL,pNode;  DWORD dwRead;  TCHAR valueData[MAX\_DATA\_LEN];//存放数据的缓冲区  BOOL isCR; //是否是回车的标记  DWORD i=0;  //088  LPTSTR lpStr;  while(TRUE)  {  pNode = (LPTNODE)HeapAlloc(hNode,HEAP\_ZERO\_MEMORY,NODE\_SIZE);  //读取节点名称  if(!ReadFile(hIn,pNode->key,TKEYSIZE,&dwRead,NULL)||dwRead!=TKEYSIZE)  return pRoot;  pNode->key[KEYSIZE-1]='\0';  isCR = FALSE;  for(i=0;i<MAX\_DATA\_LEN;i++)  {  ReadFile(hIn,&valueData[i],TSIZE,&dwRead,NULL);//用win32Api读取一个字符的写法  //因为我们不知道value的具体长度又不想浪费内存,所以需要一个一个字符来读取,每一次都需要判断是否是回车符号,如果是  //回车,并且它的下一个是换行换行,就跳出循环  if(isCR && valueData[i] == LF) break;  isCR = (valueData[i] == CR);    }  valueData[i-1] = '\0';  //088 在数据堆里面分配内存,注意,我们把key保存在节点堆里面,把key和值一整行保存在数据堆里面  lpStr = (LPTSTR)HeapAlloc(hData,HEAP\_ZERO\_MEMORY,(SIZE\_T)(KEYSIZE+\_tcslen(valueData)+1)\*TSIZE);  memcpy(lpStr,pNode->key,TKEYSIZE);//先拷贝key  lpStr[KEYSIZE]='\0';  \_tcscat(lpStr,valueData); //然后在后面拼接值  //需要让节点的一个指针指向数据堆  pNode->pData = lpStr;  //节点插入二叉树,调用InsertTree函数,我们需要实现这个函数  InsertTree(&pRoot,pNode);  printf("%s %s\n",pNode->key,valueData);  }    return NULL;  }  //088  BOOL InsertTree(LPPTNODE ppRoot, LPTNODE pNode)  {  //如果二叉树是空的,那么插入的节点就可以中文根节点  if(\*ppRoot == NULL)  {  \*ppRoot = pNode;  return TRUE;  }  //如果已经有根节点,需要判断这个节点是比根节点大还是比根节点小  //如果小在根节点的左边,大就在根节点的右边  if(KeyCompare(pNode->key,(\*ppRoot)->key)<0) //KeyCompare也是自定义函数,也需要自己来写  InsertTree(&((\*ppRoot)->Left),pNode);//这里需要使用递归调用  else  InsertTree(&((\*ppRoot)->Right),pNode);  return TRUE;  }  static int KeyCompare(LPTSTR key1,LPTSTR key2)//静态函数  {  return \_tcsncmp(key1,key2,KEYSIZE);  }  //遍历二叉树  static BOOL Scan(LPTNODE pNode) //静态函数只能够在本模块里面使用,外面不能使用  {  //如果二叉树是空的,就直接返回  if(pNode == NULL) return TRUE;  Scan(pNode->Left);//递归用法,扫描左子树  \_tprintf(\_T("%s\n"),pNode->pData);//输出数据堆里面的数据  Scan(pNode->Right);  return TRUE;  } |

## 11.为了使我们的代码更加完美,我们可以适当做一些优化,在容易有问题的代码处添加\_\_try和\_\_finally等等代码

|  |
| --- |
| #include<Windows.h>  #include<stdio.h>  #include<stdlib.h>  #include<tchar.h>  #define KEYSIZE 10 //KEY的大小是8个字符,还有一个.而且末尾需要添加一个'\0'  //定义节点数据类型  typedef struct \_TREENODE  {  struct \_TREENODE \*Left,\*Right;//二叉树的左右节点指针  TCHAR key[KEYSIZE];  LPTSTR pData;//指向数据堆的指针  }TREENODE,\*LPTNODE,\*\*LPPTNODE;  #define CR 0x0D  #define LF 0x0A  #define NODE\_HEAP\_SIZE 0x8000 //定义堆大小  #define DATA\_HEAP\_SIZE 0x8000  #define MAX\_DATA\_LEN 0x1000  #define NODE\_SIZE sizeof(TREENODE)  #define TKEYSIZE KEYSIZE \* sizeof(TCHAR)  #define TSIZE sizeof(TCHAR)  #define STATUS\_FILE\_ERROR 0x0000001  LPTNODE FillTree(HANDLE,HANDLE,HANDLE);  BOOL InsertTree(LPPTNODE, LPTNODE);//088  int KeyCompare(LPTSTR,LPTSTR);  BOOL Scan(LPTNODE);  int main(int argc,LPTSTR argv[])  {  HANDLE hIn = INVALID\_HANDLE\_VALUE;  HANDLE hNode = NULL;//节点堆  HANDLE hData = NULL;//数据堆  LPTNODE pRoot;  if(argc < 2)  {  printf(TEXT("Usage: Lesson87-SortBt1 file1 file2 ...\n"));  return -1;  }  for(int iFile =1;iFile<argc;iFile++)  \_\_try  {  printf(TEXT("%s\n"),argv[iFile]);  hIn = CreateFile(argv[iFile],GENERIC\_READ,FILE\_SHARE\_READ,NULL,OPEN\_EXISTING,FILE\_ATTRIBUTE\_NORMAL,NULL);  if(hIn == INVALID\_HANDLE\_VALUE)  RaiseException(STATUS\_FILE\_ERROR,0,0,NULL);//抛异常  \_\_try  {  hNode = HeapCreate(HEAP\_GENERATE\_EXCEPTIONS|HEAP\_NO\_SERIALIZE,NODE\_HEAP\_SIZE,0);//创建节点堆  hData = HeapCreate(HEAP\_GENERATE\_EXCEPTIONS|HEAP\_NO\_SERIALIZE,DATA\_HEAP\_SIZE,0);//创建数据堆    pRoot = FillTree(hIn,hNode,hData);    \_tprintf(\_T("排序后的文件%s\n"), argv[iFile]);//088  //遍历二叉树  Scan(pRoot);//这个也是一个自定义函数需要自己编写  }  \_\_finally  {  //不要在这里释放内存或者关闭文件,否则只能够处理一个文件,但是这个finally不能少  }  }  \_\_except((GetExceptionCode()==STATUS\_FILE\_ERROR||GetExceptionCode()==STATUS\_NO\_MEMORY) ?  EXCEPTION\_EXECUTE\_HANDLER:EXCEPTION\_CONTINUE\_SEARCH)//太长换行  {  \_tprintf(\_T("Error on file :%s\n"),argv[iFile]);  }  //最后用完了才释放比较好  if(hNode !=NULL) HeapFree(hNode,HEAP\_NO\_SERIALIZE,hNode);  if(hData !=NULL) HeapFree(hNode,HEAP\_NO\_SERIALIZE,hData);  hNode =NULL;  hData =NULL;  if(hIn != INVALID\_HANDLE\_VALUE) CloseHandle(hIn);//关闭文件  hIn = INVALID\_HANDLE\_VALUE;  return 0;  }  LPTNODE FillTree(HANDLE hIn,HANDLE hNode,HANDLE hData)  {  LPTNODE pRoot=NULL,pNode;  DWORD dwRead;  TCHAR valueData[MAX\_DATA\_LEN];//存放数据的缓冲区  BOOL isCR; //是否是回车的标记  DWORD i=0;  //088  LPTSTR lpStr;  while(TRUE)  {  pNode = (LPTNODE)HeapAlloc(hNode,HEAP\_ZERO\_MEMORY,NODE\_SIZE);  //读取节点名称  if(!ReadFile(hIn,pNode->key,TKEYSIZE,&dwRead,NULL)||dwRead!=TKEYSIZE)  return pRoot;  pNode->key[KEYSIZE-1]='\0';  isCR = FALSE;  for(i=0;i<MAX\_DATA\_LEN;i++)  {  ReadFile(hIn,&valueData[i],TSIZE,&dwRead,NULL);//用win32Api读取一个字符的写法  //因为我们不知道value的具体长度又不想浪费内存,所以需要一个一个字符来读取,每一次都需要判断是否是回车符号,如果是  //回车,并且它的下一个是换行换行,就跳出循环  if(isCR && valueData[i] == LF) break;  isCR = (valueData[i] == CR);    }  valueData[i-1] = '\0';  //088 在数据堆里面分配内存,注意,我们把key保存在节点堆里面,把key和值一整行保存在数据堆里面  lpStr = (LPTSTR)HeapAlloc(hData,HEAP\_ZERO\_MEMORY,(SIZE\_T)(KEYSIZE+\_tcslen(valueData)+1)\*TSIZE);  memcpy(lpStr,pNode->key,TKEYSIZE);//先拷贝key  lpStr[KEYSIZE]='\0';  \_tcscat(lpStr,valueData); //然后在后面拼接值  //需要让节点的一个指针指向数据堆  pNode->pData = lpStr;  //节点插入二叉树,调用InsertTree函数,我们需要实现这个函数  InsertTree(&pRoot,pNode);  printf("%s %s\n",pNode->key,valueData);  }    return NULL;  }  //088  BOOL InsertTree(LPPTNODE ppRoot, LPTNODE pNode)  {  //如果二叉树是空的,那么插入的节点就可以中文根节点  if(\*ppRoot == NULL)  {  \*ppRoot = pNode;  return TRUE;  }  //如果已经有根节点,需要判断这个节点是比根节点大还是比根节点小  //如果小在根节点的左边,大就在根节点的右边  if(KeyCompare(pNode->key,(\*ppRoot)->key)<0) //KeyCompare也是自定义函数,也需要自己来写  InsertTree(&((\*ppRoot)->Left),pNode);//这里需要使用递归调用  else  InsertTree(&((\*ppRoot)->Right),pNode);  return TRUE;  }  static int KeyCompare(LPTSTR key1,LPTSTR key2)//静态函数  {  return \_tcsncmp(key1,key2,KEYSIZE);  }  //遍历二叉树  static BOOL Scan(LPTNODE pNode) //静态函数只能够在本模块里面使用,外面不能使用  {  //如果二叉树是空的,就直接返回  if(pNode == NULL) return TRUE;  Scan(pNode->Left);//递归用法,扫描左子树  \_tprintf(\_T("%s\n"),pNode->pData);//输出数据堆里面的数据  Scan(pNode->Right);  return TRUE;  } |

## 说真的,这些代码并没有多大用处.