数据结构实验报告

2017年12月11日　　23:36

题目:哈夫曼编解码器设计
班级:信息学院二班
学号:PB16150288
姓名:黄志鹏

需求分析:

1. 本道题目主要是想考察通信中哈夫曼编解码的相关实现.
2. 一个完整的哈夫曼编解码系统，需要有初始化,编码，解码等功能.
3. 除了以上功能外，还应该有打印代码文件和哈夫曼树的功能.
4. 本道题目需要一个用户界面，对整个进程进行控制，可以随时结束退出

概要设计：

1. 系统中的所有对象和属性

- 系统控制台(app)
  - show()
  - Cycle()
- 编解码器（coder）
  - Initialization()
  - Encoder()
  - Decoder()
  - Print_code()
  - Print_huffman()
- 哈夫曼树
  - Data
    - Letter
    - weight
  - Parent
  - Lchild
  - Rchild
- 栈
  - Stackempty()
  - Gettop()
  - Pop()
  - Push()

详细设计：

1. 前三个对象（类）中：属于一层套一层的抽象度逐级降低的结构，在app类中有一个coder类的私有成员，这样很好的对底层进行了保护，并且有效的分解了各层的变量，使得编程和调试过程变得十分的简单。
2. App.h文件和app.cpp文件，描述了第一个类app,它负责需求4：呈现一个比较友好的的用户界面，并且可以对整个系统进行指令和控制，这是通过对它的私有成员coder进行操作完成的（PS：这部分只要稍加改造，就能改造成更加友好的图形界面系统，而不影响程序的其他部分）
3. coder.h文件和coder.cpp文件，描述了app的类coder，它对直接操作于一个哈夫曼树，负责完成需求2,3，即一系类哈夫曼编解码器需要有的功能。
4. 其中coder.cpp文件中的huffman_treeprint()函数采用递归的方法，将哈夫曼树打印到终端和文件huffman_print.txt文件中。
   代码如下：

```
void Coder::print_haffmantree(int bt, int levl, bool start)
{
  if(start){
    cout<<"==============the following is the huffmantree created before==========="<<endl;
    int i;              //the variable for cycling
    HT = new HTNode[m+1];
    HC = new char*[n+1];
    infile.open("E:\\CodeblockFile\\homework\\haffman\\hfmtree.txt", ios::in);
    if(!infile) cout<<"can not open hfmtree.txt";
    for(i = 1; i<=2*n-1; i++) infile>>HT[i].weight>>HT[i].parent>>HT[i].lchild>>HT[i].rchild;
    infile.close();
    bt = m;
    outfile.open("E:\\CodeblockFile\\homework\\haffman\\TreePrint.txt", ios::out);
  }
```

```
    if(bt==0)
        return;                 //已经是最后了
    print_haffmantree(HT[bt].rchild,levl+1,false);//递归调用
    for(int i=0;i<levl;++i){
     cout<<"   ";//先输出空格，给后面的继续输入留出余地
     outfile<<"   ";
    }
    if(levl>=1){
     cout<<"--";
     outfile<<"--";
    }
    cout<<"("<<HT[bt].weight<<")";//输出右结点的信息（权值）
    outfile<<"("<<HT[bt].weight<<")";
    if(HT[bt-1].lchild==0)//如果左结点也是叶子结点，输出左结点的信息
    {
        cout<<ch[bt];
        cout<<':'<<HT[bt].weight<<endl;
  outfile<<ch[bt];
        outfile<<':'<<HT[bt].weight<<endl;
    }
    else{
     cout<<endl;
     outfile<<endl;
    }
    print_haffmantree(HT[bt].lchild,levl+1,false);
    if(start){
     outfile.close();
     cout<<"printing complete and the tree's picture is stored in TreePrint"<<endl;
    }
}
```

调试

1. 在编解码中的边界值设置容易出错，
2. 各层次类的封装和功能引出容易错乱.

程序运行效果

E:\CodeblockFile\homework\haffman\bin\Debug\haffman.exe

```
this is the huffman en-decoder communication system
=============tap I to initialize==================
=============tap E to encode the tobetran=========
=============tap D to decode the codefile=====
=============tap P to print what you want=======
=============tap Q to quit=====================

**************i am the split line*********************
I
please input the number you your encode letters
27
please input the letters
 ABCDEFGHIJKLMNOPQRSTUVWXYZ
please input the weights to initialize the huffman tree
186 64 13 22 32 103 21 15 47 57 1 5 32 20 57 63 15 1 48 51 80 23 8 18 1 16 1
the huffmantree is created and its data is written to hfmtree
E
encoder completed and the result is written to codefile
D
decoder completed and the result is written to TextFile
P
Do you want to print the codefile or the huffman tree
 ====tap f for codefile ====tap t for huffman tree
t
==============the following is the huffmantree created before===========
         --(186) :186
      --(342)
            --(80)T:80
         --(156)
                  --(21)F:21
               --(41)
                  --(20)M:20
            --(76)
                  --(18)W:18
               --(35)
                        --(5)K:5
                     --(9)
                           --(1)Z:1
                        --(2)
                           --(1)X:1
                     --(4)
                           --(1)Q:1
                        --(2)?2
                           --(1)J:1
                  --(17)
                     --(8)V:8
   --(592)
               --(32)L:32
            --(64)
               --(32)D:32
         --(128)
            --(64)A:64
      --(250)
            --(63)O:63
         --(122)
                  --(16)Y:16
               --(31)
                  --(15)P:15
            --(59)
                  --(15)G:15
               --(28)
                  --(13)B:13
(1000)
            --(57)N:57
         --(114)
            --(57)I:57
      --(217)
         --(103)E:103
   --(408)
            --(51)S:51
         --(99)
            --(48)R:48
      --(191)
            --(47)H:47
         --(92)
               --(23)U:23
            --(45)
               --(22)C:22
printing complete and the tree's picture is stored in TreePrint
Q

Process returned 0 (0x0)   execution time : 46.569 s
Press any key to continue.
```