（接上）实验内容（算法、程序、步骤和方法）

在第一版思路的基础上，为了快速实现新的，更加简便的哈夫曼树，这里只利用的结构体而未使用类，但实现的方式大同小异。

且为了方便优先队列的定义事先定义了一个 Compare 的结构体，其中的函数对象来实现比较功能。

源代码如下：

```
#include<iostream>
#include<queue>
#include<cstdlib>
using namespace std;

struct TreeNode
{
    char data;
    unsigned freq;
    string co;

    TreeNode* lchild;
    TreeNode* rchild;

    TreeNode(char data,unsigned freq)
    {
        lchild=rchild=NULL;
        this->data=data;
        this->freq=freq;
    }
};

struct Compare
{
    bool operator()(TreeNode* l,TreeNode* r)
    {
        return (l->freq>r->freq);
    }
};

//void printCodes(TreeNode *root)
```

```
TreeNode* createHuff(char data[],int freq[],int size)
{
    TreeNode *top, *left, *right;
    priority_queue<TreeNode *, vector<TreeNode *>, Compare> theTree;
    for(int i=0;i<size;i++)
    {
        theTree.push(new TreeNode(data[i], freq[i]));
    }

    while(theTree.size()>1)
    {
        left=theTree.top();
        theTree.pop();
        right=theTree.top();
        theTree.pop();

        top = new TreeNode('$', left->freq + right->freq);
        top->lchild=left;
        top->rchild=right;
        theTree.push(top);
    }

    return theTree.top();
}
//setcodes(example,"");
void setcodes(TreeNode *node,const string &str)
{
    if(!node)
    {
        return;
    }
    if(node!=NULL)
    {

        node->co=str;
        //cout << node->co << "::" << node->data << endl;
        setcodes(node->lchild, str + "0");
        setcodes(node->rchild, str + "1");
```

```
    }
}

void preShow(TreeNode* node)
{
    if (node!=NULL)
    {
        if (node->data!='$')
        cout<<node->co<<" : "<<node->data<<endl;
        preShow(node->lchild);
        preShow(node->rchild);
    }
}

int main()
{
    char data[]={'a','b','c','d','e','f'};
    int freq[] = { 3, 52, 23, 24, 15, 77};
    int size = sizeof(data) / sizeof(data[0]);
    TreeNode *flag = createHuff(data, freq, size);
    setcodes(flag, "");
    preShow(flag);
    return 0;
}
```

除了一个前序遍历来输出哈夫曼树，另外的主函数为 setcodes 和 createHuff 分别是对已有的生成树进行编码赋值和产生目的哈夫曼树。createHuff 的核心思想与之前的一样，不断从队列中提取最大的两个元素，（这是第一个提出的作为左结点，第二个作为右结点），然后将权值付给新的临时 TOP，再将其压入队列继续 while 循环即可，其中的队列采用的是 STL 的，不过快速排序实现起来也并不麻烦。

编码的方式也是采用递归，考虑到 string 的特殊性，传递参数时采用 const string& 进行操作，确保安全。如果它是当前结点的左结点则把当前的 str+=“0”，否则+=“1”即可。