各种排序算法C++模版类实现

闲来无事，于是把常用的排序算法自己写了一遍，也当做是复习一下。

```
/*****************************************************************
 *              *
 *              *
 *       Date   : 2012. 05. 03          *
 *    Author : 397090770        *
 *    Email  : wyphao.2007@163.com       *
 *              *
 *              *
 *****************************************************************/

#include <iostream>
#include <iomanip.h>

#define LEN 100  //排序数的个数
#define NUM 10  //每行输出的字数个数

using namespace std;

////////////////////////////////////////////////////////////////////
//树节点
template <class T>
class Node{
public:
 Node *left;
 Node *right;
 T data;

 Node() : left(NULL), right(NULL), data(NULL){}
 ~Node(){}
};

////////////////////////////////////////////////////////////////////
class Sort{
public:
 Sort(){};
 ~Sort(){};
 //快速排序
 template <class T>
 void QuickSort(T arr[], int low, int hight);
```

```
 //选择排序
 template <class T>
 void SelectSort(T arr[], int len);

 //冒泡排序
 template <class T>
 void BubbleSort(T arr[], int len);

 //插入排序
 template <class T>
 void InsertSort(T arr[], int len);

 //堆排序
 template <class T>
 void HeapSort(T arr[], int len);

 //二叉排序树排序
 template <class T>
 void TreeSort(T arr[], int len);

private:
 //快速排序中选择中心点
 template <class T>
 int Quick(T arr[], int left, int right);

 //建立堆
 template <class T>
 void CreateHeap(T arr[], int root, int len);

 //建立二叉排序树
 template <class T>
 Node<T>* BuildTree(Node<T>*root, T data);

 //中序遍历二叉排序树
 template <class T>
 void InTree(Node<T> *root, T arr[]);
};

//////////////////////////////////////////////////////////////////////
int main(){
 Sort sort;
 int *arr;        //需要排序的数组
 int width = 0;       //最大数的位数，用于排列输出结果
 int len = LEN;       //用来求最大数的位数
 arr = (int *)malloc(LEN * sizeof(int)); //分配空间
```

```
 if(arr == NULL){     //空间分配失败
  cout << "Malloc failed!" << endl;
  exit(1);
 }

 srand(time(NULL));      //设置种子
 for(int i = 0; i < LEN ;i++){   //随机生成数字
  arr[i] = (rand() % (LEN * 10)) + 1;
 }

 //sort.SelectSort(arr, LEN);
 sort.TreeSort(arr, LEN);

 //求得最大数的位数，用于排列输出结果
 while(len){
  width++;
  len /= 10;
 }

 for(int i = 0; i < LEN; i++){   //输出排序后的数字
  cout << setw(width) << arr[i] << " ";
  cout << fixed;
  if((i + 1) % NUM == 0){    //每行输出的数字个数
   cout << endl;
  }
 }

 cout << endl;
 return 0;
}

//////////////////////////////////////////////////////////////////////
//快速排序
template <class T>
void Sort::QuickSort(T arr[], int low, int hight){
 int pivot = -1;
 if(low <= hight){
  pivot = Quick(arr, low, hight);
  QuickSort(arr, low, pivot - 1);
  QuickSort(arr, pivot + 1, hight);
 }
}

//返回中轴点的下标
template <class T>
int Sort::Quick(T arr[], int left, int right){
```

```
 int i = left + 1, j = right;
 int flag = left;
 int temp;

 while(i <= j){
  while(i <= j && arr[i] < arr[flag]){
   ++i;
  }
  while(i <= j && arr[j] > arr[flag]){
   --j;
  }
  if(i < j){
   temp = arr[i];
   arr[i] = arr[j];
   arr[j] = temp;
   ++i;
   --j;
  }
 }

 temp = arr[flag];
 arr[flag] = arr[j];
 arr[j] = temp;
 return j;
}

//////////////////////////////////////////////////////////////////////
//选择排序
template <class T>
void Sort::SelectSort(T arr[], int len){
 int index;
 T temp;
 for(int i = 0; i < len - 1; i++){
  index = i;
  for(int j = i + 1; j < len; j++){
   if(arr[index] > arr[j]){
    index = j;
   }
  }
  if(index != i){
   temp = arr[index];
   arr[index] = arr[i];
   arr[i] = temp;
  }
 }
}
```

```
//////////////////////////////////////////////////////////////////////
//冒泡排序
template <class T>
void Sort::BubbleSort(T arr[], int len){
 T temp;
 bool flags = true;
 for(int i = len - 1; i > 0; i--){
  if(flags){
   flags = false;
   for(int j = 0; j < i; j++){
    if(arr[j] > arr[j + 1]){
     flags = true;
     temp = arr[j];
     arr[j] = arr[j + 1];
     arr[j + 1] = temp;
     for(int k = 0; k < LEN; k++){
      cout << arr[k] << " ";
     }
     cout << endl;
    }
   }
  }else{
   break;
  }
 }
}

//////////////////////////////////////////////////////////////////////
//插入排序
template <class T>
void Sort::InsertSort(T arr[], int len){
 T temp;
 int i, j;
 for(i = 1; i < len; i++){
  temp = arr[i];
  for(j = i - 1; j >= 0; j--){
   if(temp < arr[j]) {
    arr[j + 1] = arr[j];
   }else{
    break;
   }
  }
  arr[j + 1] = temp;
 }
}
```

```
//////////////////////////////////////////////////////////////////////
//堆排序
template <class T>
void Sort::HeapSort(T arr[], int len){
 int i;
 T buff;
 T *temp = (T *)malloc(sizeof(T) * (len + 1));
 if(temp == NULL){
  cout << "Malloc Error!" << endl;
  exit(1);
 }
 for(i = 1; i < len + 1; i++){ //复制数组，使得偏移从1开始，这样好计算左孩子和右孩子坐标
  temp[i] = arr[i - 1];
 }

 //建立子堆
 for(i = len / 2; i >= 1; i--){
  CreateHeap(temp, i, len);
 }

 for(i = len - 1; i >= 1; i--){
  buff = temp[1];
  temp[1] = temp[i + 1];
  temp[i + 1] = buff;

  CreateHeap(temp, 1, i);
 }

 for(i = 1; i < len + 1; i++){
  arr[i - 1] = temp[i];
 }
}

//建立堆
template <class T>
void Sort::CreateHeap(T arr[], int root, int len){
 int j = 2 * root;      //root's left child, right (2 * root + 1)
 T temp = arr[root];
 bool flags = false;

 while(j <= len && !flags){
  if(j < len){
   if(arr[j] < arr[j + 1]){ // Left child is less then right child
    ++j;      // Move the index to the right child
   }
```

```
  }

  if(temp < arr[j]){
   arr[j / 2] = arr[j];
   j *= 2;
  }else{
   flags = true;
  }
 }
  arr[j / 2]  = temp;
}

//////////////////////////////////////////////////////////////////////
//二叉排序树排序
template <class T>
void Sort::TreeSort(T arr[], int len){
 Node <T>*root = NULL;
 for(int i = 0; i < len; i++){
  root = BuildTree(root, arr[i]);
 }

 InTree(root, arr);
}

//建立二叉排序树
template <class T>
Node<T>* Sort::BuildTree(Node<T>*root, T data){
 Node<T> *tempNode = root;
 Node<T> *parentNode = NULL;

 Node<T> *newNode = new Node<T>;
 newNode->data = data;
 newNode->left = NULL;
 newNode->right = NULL;

 if(root == NULL){//空树的时候
  return newNode;
 }else{
  while(tempNode != NULL){
   parentNode = tempNode;
   if(tempNode->data >= data){
    tempNode = tempNode->left;
   }else{
    tempNode = tempNode->right;
   }
  }
```

```
  if(parentNode->data >= data){
   parentNode->left = newNode;
  }else{
   parentNode->right = newNode;
  }
 }

 return root;
}

//中序遍历二叉排序树，将二叉树的节点存储在数组中
template <class T>
void Sort::InTree(Node<T> *root, T arr[]){
 static int index = 0;
 if(root != NULL){
  InTree(root->left, arr);
  arr[index++] = root->data;
  InTree(root->right, arr);
 }
}
//////////////////////////////////////////////////////////////////////////
```