实验11:

//11.1 最小堆的各项操作（建堆、插入80、删除9、每次打印最小堆）

//11.2 二叉搜索树（依次输入、建BST、递归中序遍历、打印结果）

```
#include <iostream>
#define MaxHeapSize  100
using namespace std;

typedef int DataType;
typedef struct {
   DataType key;
} HeapElem;

typedef struct {
   HeapElem data[MaxHeapSize];
   int CSize;
} MinHeap;


void siftDown(MinHeap& H,  int start, int EndOfHeap) {
   int i = start, j = 2*i+1;
   HeapElem temp = H.data[i];
   while (j <= EndOfHeap) {
      if (j < EndOfHeap &&
         H.data[j].key > H.data[j+1].key)
         j++;
      if (temp.key <= H.data[j].key) break;
      else {
         H.data[i] = H.data[j];
         i = j;
         j = 2*j+1;
      }
   }
   H.data[i] = temp;
}

void siftUp (MinHeap& H, int start) {
   int j = start,  i = (j-1)/2;
   HeapElem temp = H.data[j];
   while (j > 0) {
      if (H.data[i].key <= temp.key) break;
      else {
         H.data[j] = H.data[i];
         j = i;
         i = (i-1) /2;
      }
   }
```

```
    H.data[j] = temp;
}

void Crt_MinHeap (MinHeap & H, HeapElem arr [], int n) {
    for (int i = 0; i < n; i++) {
        H.data[i] = arr[i];
    }
    H.CSize = n;
    int CPos = (H.CSize-2)/2;
    while (CPos >= 0) {
        siftDown(H, CPos, H.CSize-1);
        CPos--;
    }
}

void printMinHeap(MinHeap & h, int n) {
    for(int i=0;i<n;i++) {
        printf("%d ", h.data[i].key);
    }
    cout << endl;
}

int Insert(MinHeap & H, HeapElem x) {
    if (H.CSize == MaxHeapSize)
    { printf ("Heap is full.\n");   return 0; }
    H.data[H.CSize] = x;
    siftUp(H, H.CSize);
    H.CSize++;
    return 1;
}



int RemoveMin(MinHeap& H, HeapElem &x) {
    if (!H.CSize)
    { cout << "Heap is empty." << endl;  return 0; }
    x = H.data[0];
    H.data[0] = H.data[H.CSize-1];
    H.CSize--;
    siftDown(H, 0, H.CSize-1);
    return 1;
}


typedef int ElemType;

typedef struct node {
    ElemType data;
    struct node *leftChild, *rightChild;
} BstNode, *BST;

void Find (BstNode *t, ElemType x,
          BstNode *&p, BstNode *&pr) {
    if (t != NULL) {
        p = t;  pr = NULL;
        while (p != NULL && p->data != x) {
            pr = p;
            if (p->data < x) p = p->rightChild;
            else p = p->leftChild;
        }
    }
}

void Insert (BstNode *& t, ElemType x) {
    BstNode *pt, *prt, *q;
    Find (t, x, pt, prt);
    if (pt == NULL) {
        q = new BstNode;
        q->data = x;
        q->leftChild = q->rightChild = NULL;
        if (prt == NULL) t = q;
        else if (x < prt->data) prt->leftChild = q;
        else prt->rightChild = q;
    }
}

void inorderBST(BST & bst) {
    if(bst != NULL) {
        inorderBST(bst->leftChild);
        printf("%d ", bst->data);
        inorderBST(bst->rightChild);
    }
}

int main(int argc, const char * argv[]) {

    cout << "Exercise 1:" << endl;
    HeapElem numbers0[8];
    DataType numbers1[8] = {53, 17, 78, 23, 45, 65, 87, 9};
    for(int i=0;i<8;i++) {
        numbers0[i].key = numbers1[i];
    }
    MinHeap minheap;
    cout << "Creating Minimum Heap:" << endl;
    Crt_MinHeap(minheap, numbers0, 8);
    printMinHeap(minheap, 8);
    cout << "Insert 80:" << endl;
    HeapElem h1;
    h1.key = 80;
    Insert(minheap, h1);
    printMinHeap(minheap, 9);
    cout << "Remove 9:" << endl;
    HeapElem h2;
    h2.key = 9;
    RemoveMin(minheap, h2);
    printMinHeap(minheap, 8);

    cout << endl << "Exercise 2:" << endl;
    BST bst = new node();
    ElemType numbers2[8] = {53, 78, 65, 17, 87, 9, 81, 15};
    for(int i=0;i<8;i++) {
        Insert(bst, numbers2[i]);
    }
    cout << "Inorder binary search tree:" << endl;
    inorderBST(bst);

    cout << endl;
    return 0;
}
```

Output:

```
Exercise 1:
Creating Minimum Heap:
9 17 65 23 45 78 87 53
Insert 80:
9 17 65 23 45 78 87 53 80
Remove 9:
17 23 65 53 45 78 87 80

Exercise 2:
Inorder binary search tree:
0 9 15 17 53 65 78 81 87
```