汇编实验三

完成实验后，需用实验报告纸撰写实验报告。

一、实验报告包含以下内容

- 实验序号
- 实验内容
- 算法描述
- 汇编程序
- 运行结果

二、实验目的

1. 掌握分支与循环结构的汇编表示。
2. 掌握子以栈传递参数方式编写程序或函数
3. 掌握递归编程
4. 熟悉调试器 ollydbg 的使用。

三、课堂练习

1. 试编程实现正整数的素数分解。例如：　72=2^3*3^2

```
int factorNumber(int *array,int n)
len=0;
if isPrime(n) then {
    save(n,1) to array;
    return len++;
}
p=2;
while (p<=n/2) {
    If  isPrime(p) then {
        if k= maxExp (n, p) > 0 then {
        save(p,k) to array;
        len++;
        }
          }
    p=p+1;
}
return len;
}
```

算法 isPrime(n)：

```
if  n == 1 then  return 0;
for (i=2; i*i<n; i++)
    if n%i ==0 then return 0;
}
return 1;
```

算法 maxExp (n, p)：

```
k=0;
```

```
while ( n%p==0) {
    k=k+1;
    n=n/p;
}
return k;
```

2.汉诺塔问题

有三根杆子 A，B，C。A 杆上有 N 个(N>1)穿孔圆盘，盘的尺寸由下到上依次变小。要求按下列规则将所有圆盘移至 C 杆：

1、每次只能移动一个圆盘；

2、大盘不能叠在小盘上面。

提示：可将圆盘临时置于 B 杆，也可将从 A 杆移出的圆盘重新移回 A 杆，但都必须遵循上述两条规则。

问：如何移？最少要移动多少次？

汉诺算法基本思想：

第一次移动，要把 A 柱子上的前 n-1 个移动到 B 柱子上；

第二次移动，直接把 A 柱子上的最后一个移动到 C 柱子上；

第三次移动，把 B 柱子上的 n-1 个柱子通过柱子 A 移动到柱子 C 上。

接着通过递归算法就完成了。

```
void Hanoi(int n, char one,char two,char three){
    if(n==1) move(one,three)
    else{
        Hanoi(n-1,one,three,two);
        Move(one,three);
        Hanoi(n-1,two,one,three);
    }
}

void Move(char x,char y){
    print("%c-->%c",x,y);
}

int main(){
    int n;
    printf("Input Your Number");
    scanf("%d",&n);
    Hanoi(n,'A','B','C')

}
```

3. 试编程实现快速排序。

算法 C 描述

```
void swap(int *a, int *b){
    int tmp = *a;
```

```
    *a = *b;
    *b = tmp;
}

int partition(int a[], int low, int high){
    int Key = a[low]; //基准元素
    while(low < high){ //从表的两端交替地向中间扫描
        while(low < high && a[high] >= Key) --high;
        swap(&a[low], &a[high]);
        while(low < high && a[low] <= Key ) ++low;
        swap(&a[low], &a[high]);
    }
    return low;
}

void quickSort(int a[], int low, int high){
    if(low < high){
        int Loc = partition(a, low, high); //将表一分为二
        quickSort(a, low, Loc -1); //递归对低子表递归排序
        quickSort(a, Loc + 1, high); //递归对高子表递归排序
    }
}
```

四、实验内容

1.试编程实现堆排序。

2.试编程实现二分查找。

3.试编程实现 输出集合{1，2，…,n}全排列。

4.试编程实现八皇后问题。

提示：

1. 堆排序算法（Heap Sort）

堆是一种数组对象具有以下性质：

任意的叶子节点小于（或大于）它所有的父节点。对此，又分为大顶堆和小顶堆：

大顶堆要求节点的元素都要大于其孩子。

小顶堆要求节点元素都小于其左右孩子。

两者对左右孩子的大小关系不做任何要求。

利用堆排序，就是基于大顶堆或者小顶堆的一种排序方法。下面，通过大顶堆来实现。

基本思想：堆排序可以按照以下步骤来完成：

1.首先将序列构建称为大顶堆；（这样满足了大顶堆那条性质：

位于根节点的元素一定是当前序列的最大值）

2. 取出当前大顶堆的根节点，将其与序列末尾元素进行交换；

（此时：序列末尾的元素为已排序的最大值；由于交换了元素，

当前位于根节点的堆并不一定满足大顶堆的性质）

3. 对交换后的 n-1 个序列元素进行调整，使其满足大顶堆的性质；
4. 重复 2.3 步骤，直至堆中只有 1 个元素为止

下面是基于大顶堆的堆排序算法代码：

```
void print(int a[], int n){
    for(int j= 0; j<n; j++){
        cout<<a[j] <<" ";
}
cout<<endl;
}
/**
已知 H[s…m]除了 H[s] 外均满足堆的定义
调整 H[s],使其成为大顶堆.即将对第 s 个结点为根的子树筛选,
@param H 是待调整的堆数组
@param s 是待调整的数组元素的位置
@param length 是数组的长度
**/
void HeapAdjust(int H[],int s, int length)
{
    int tmp = H[s];
    int child = 2s+1; //左孩子结点的位置。(i+1 为当前调整结点的右孩子结点的位置)
    while (child < length) {
        if(child+1 <length && H[child]<H[child+1]) {
        // 如果右孩子大于左孩子(找到比当前待调整结点大的孩子结点)
            ++child ;
        }
        if(H[s]<H[child]) { // 如果较大的子结点大于父结点
            H[s] = H[child]; // 那么把较大的子结点往上移动，替换它的父结点
            s = child; // 重新设置 s ,即待调整的下一个结点的位置
            child = 2*s+1;
        } else { // 如果当前待调整结点大于它的左右孩子，则不需要调整，直接退出
            break;
        }
        H[s] = tmp; // 当前待调整的结点放到比其大的孩子结点位置上
    }
    print(H,length);
}
/**
初始堆进行调整
将 H[0..length-1]建成堆
调整完之后第一个元素是序列的最小的元素
**/
void BuildingHeap(int H[], int length){
//最后一个有孩子的节点的位置 i= (length -1) / 2
```

```
    for (int i = (length -1) / 2 ; i >= 0; --i)
        HeapAdjust(H,i,length);
}

/*堆排序算法*/
void HeapSort(int H[],int length){
    //初始堆
    BuildingHeap(H, length);
    //从最后一个元素开始对序列进行调整
    for (int i = length - 1; i > 0; --i){
    //交换堆顶元素 H[0]和堆中最后一个元素
        int temp = H[i];
        H[i] = H[0];
        H[0] = temp;
        //每次交换堆顶元素和堆中最后一个元素之后，都要对堆进行调整
        HeapAdjust(H,0,i);
    }
}
int main(){
    int H[10] = {3,1,5,7,2,4,9,6,10,8};
    cout<<"初始值：";
    print(H,10);
    HeapSort(H,10);
    cout<<"结果：";
    print(H,10);

}
```

2. 二分查找算法 c 描述

把数据分成两半，再判断所查找的 key 在哪一半中，再重复上述步骤知道找到目标 key;

```
#include<stdio.h>
int BiSearch(int arr[],int len,int key){ //二分法
 int low=0;         //定义初始最小
 int high=len-1;    //定义初始最大
 int mid;               //定义中间值
 while(low<=high) {
    mid=(low+high)/2;  //找中间值
    if(key==arr[mid]) //判断 min 与 key 是否相等
        return mid;
    else if(key>arr[mid]) //如果 key>mid 则新区间为[mid+1,high]
        low=mid+1;
    else                  //如果 key<mid 则新区间为[low,mid-1]
        high=mid-1;
```

```
 }
 return -1;          //如果数组中无目标值 key，则返回 -1 ；
}

int main(){
 int arr[]={1,2,3,4,5,6,7,8,9,10,11};   //首先要对数组 arr 进行排序
 printf("%d \n",BnSearch(arr,(sizeof(arr)/sizeof(arr[0])),7));
 return 0;
}
```

3.全排列算法
基本思想：
1.把第 1 个数换到最前面来（本来就在最前面），准备打印 1xx，再对后两个数 2 和 3 做全排列。
2.把第 2 个数换到最前面来，准备打印 2xx，再对后两个数 1 和 3 做全排列。
3.把第 3 个数换到最前面来，准备打印 3xx，再对后两个数 1 和 2 做全排列。
这是一个递归的过程，把对整个序列做全排列的问题归结为对它的子序列做全排列的问题

```
include <stdio.h>
void Swap(int *lhs, int *rhs){
    int t = *lhs;
    *lhs = *rhs;
    *rhs = t;
}

/*****************************************************************************/
/* 功能：实现全排列功能
/* 参数：
/*       source--整数数组，存放需要全排列的元素
/*       begin --查找一个排列的开始位置
/*       end   --查找一个排列的结束位置，当 begin=end 时，表明完成一个排列
/*****************************************************************************/
void FullPermutation(int source[], int begin, int end){
    int i;
    if (begin >= end){ // 找到一个排列
        for(i = 0; i < end; i++){
            printf("%d", source[i]);
        }
        printf("\n");
    }
```

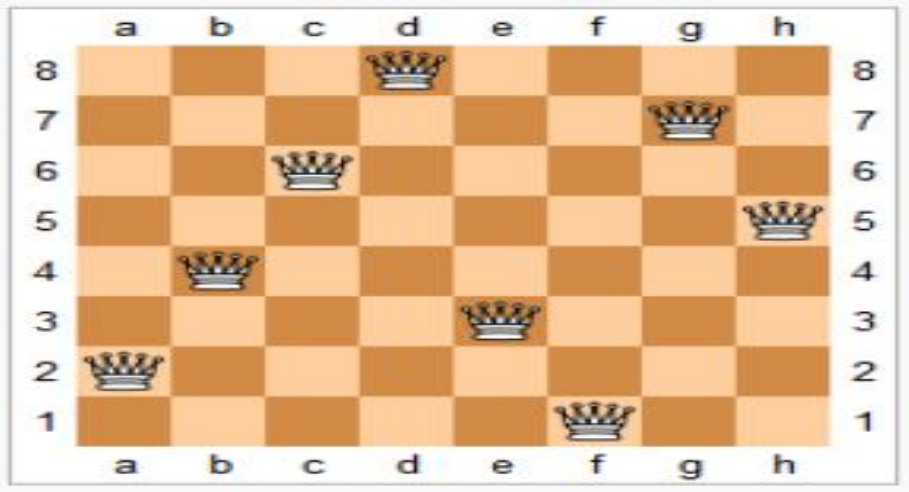
```
    else{ // 没有找完一个排列，则继续往下找下一个元素
        for (i = begin; i < end; i++){
            if (begin != i){
            Swap(&source[begin], &source[i]); // 交换
            }
            // 递归排列剩余的从 begin+1 到 end 的元素
            FullPermutation(source, begin + 1, end);

            if (begin != i){
            Swap(&source[begin], &source[i]); // 回溯时还原
            }
        }
       }
}

int main( ){
    int source[30];
    int i, count;
    scanf("%d", &count);
    for (i = 0; i < count; i++){
        source[i] = i + 1;
    }
    FullPermutation(source, 0, count);
    return 0;
}
```

4.八 皇后问题

八皇后问题，是一个古老而著名的问题，是回溯算法的典型案例。该问题是国际西洋棋棋手马克斯·贝瑟尔于 1848 年提出：在 8× 8 格的国际象棋上摆放八个皇后，使其不能互相攻击，即任意两个皇后都不能处于同一行、同一列或同一斜线上，问有多少种摆法。 高斯认为有 76 种方案。 1854 年在柏林的象棋杂志上不同的作者发表了 40 种不同的解，后来有人用图论的方法解出 92 种结果。



```
#include <stdio.h>

void PutQueen(int n);
int  chess[8][8]={0};
int  a[8],b[15],c[15];
int  sum=0;

int main(int argc, char *argv[]){
    int i;
    for(i=0;i<8;i++ )
        a[i]=1;
    for(i=0;i<15;i++){
        b[i]=1;
        c[i]=1;
    }
    PutQueen(0);
    printf("八皇后摆法总数： %d\n",sum);
    return 0;
}

void PutQueen(int n){
    int col,i,j;
    for(col=0;col<8;col++){
        if(a[col]&& b[n+col] && c[n-col+7]){//if_1
            chess[n][col]=1;
            a[col]=0;
            b[n+col]=0;
            c[n-col+7]=0;
            if(n==7){ //if_2
                sum++;
                printf("第%d 种可能摆法：\n",sum);
                for(i=0;i<8;i++){
                    printf("\t\t");
                    for(j=0;j<8;j++)
                        printf("%d ",chess[i][j]);
                    printf("\n");
                }
                printf("\n");
                if(sum%10==0){
                    printf("按回车键继续...");
                    getchar();
                }
            }else{PutQueen(n+1);
            //end if_2
```

```
            chess[n][col]=0;
            a[col]=1;
            b[n+col]=1;
            c[n-col+7]=1;
        }//end if_1
    }//end for
}
```