《数据结构》上机作业

注意：以下作业文件名命名格式为：“学号姓名”+“程序名”，在规定的时间上交到指定的位置。

1. 将下面两个 C 语言程序在 VC++下调试通过并完成以下任务。

程序一：

```
/* algo1-0-1.c 计算 1-1/x+1/x*x… */
 #include<stdio.h>
 #include<sys/timeb.h>
 void main()
 {
   struct timeb t1,t2;
   long t;
   double x,sum=1,sum1;
   int i,j,n;
   printf("请输入 x n：");
   scanf("%lf%d",&x,&n);
   ftime(&t1); /* 求得当前时间 */
   for(i=1;i<=n;i++)
   {
     sum1=1;
     for(j=1;j<=i;j++)
       sum1=-sum1/x;
     sum+=sum1;
   }
   ftime(&t2); /* 求得当前时间 */
   t=(t2.time-t1.time)*1000+(t2.millitm-t1.millitm); /* 计算时间差 */
   printf("sum=%lf 用时%ld 毫秒\n",sum,t);
 }
```

程序二：

```
/* algo1-0-2.cpp 计算 1-1/x+1/x*x…的更快捷的算法 */
 #include<stdio.h>
 #include<sys/timeb.h>
 void main()
 {
   struct timeb t1,t2;
   long t=0;
   double x,sum1=1,sum=1;
```

```
    int i,n;
    printf("请输入 x n: ");
    scanf("%lf%d",&x,&n);
    ftime(&t1); /* 求得当前时间 */
    for(i=1;i<=n;i++)
    {
      sum1=-sum1/x;
      sum+=sum1;
    }
    ftime(&t2); /* 求得当前时间 */
    t=(t2.time-t1.time)*1000+(t2.millitm-t1.millitm); /* 计算时间差 */
    printf("sum=%lf 用时%ld 毫秒\n",sum,t);
}
```

（1）对每个程序，在 x 相同的情况下变换 n 值（如：n=10,n=100,n=1000, n=2000,n=5000,n=10000,.....），在任何一个字处理软件中自行设计一个表格，记录、比较程序运行时间的差别。比较在相同的条件下（x 和 n 相同）程序一和程序二执行时间的差别；

（2）将 C 程序中“求当前时间”的语句去除，改为在循环体中增加一个计数器，计录其执行次数。比较在相同的条件下（x 和 n 相同）两个算法的循环执行次数。将结果记录在自行设计的表中。

将以上记录表保存为：学号+姓名+“algo1-0 实验报告”，提交。

2. 编写一个程序 algo1-1.cpp,计算 $i!\times 2^i$（i＝0,1,…n-1）的值并分别存入数组 a[NUM]的各个分量中（$0!\times 2^0$存入 a[0]，$1!\times 2^1$存入 a[1]，$2!\times 2^2$存入 a[2]，……）。假设计算机中允许的整数最大值为 MAXINT，则当对某个 n，$n!\times 2^n$ >MAXINT 时，应停止计算，结束程序。

3. 选做题：设计一个程序 algo1-2.cpp，输出所有小于等于 n（n 为一个大于 2 的正整数）的素数。要求：（1）每行输出 10 个素数学（2）尽可能采用较优的算法。

4. 下面是线性表的顺序存储结构定义和基本操作函数（见学院网上教学《数据结构》课程中的 HW4-S.rar 文件），请编写一个主程序，调用 bo2-1.c 中的基本操作函数完成如下功能：

（1）初始化顺序表 L；
（2）依次采用尾插法插入 a,b,c,d,e 元素；
（3）输出顺序表 L；
（4）输出顺序表 L 长度；
（5）输出顺序表 L 的第 3 个元素；
（6）在第 4 个元素位置上插入’f’元素；

（7）输出顺序表 L；
（8）删除 L 的第 3 个元素；
（9）输出顺序表 L；
（10）释放顺序表 L。

提示：根据下列给出的源代码建立两个头文件和一个 c 文件，自己编写实现指定功能的 main 函数，具体过程如下：

首先，在磁盘上建立一个文件夹，如：“学号姓名+ZY4”，在其下建立以下文件：

(1)用以下源代码建立头文件“c1.h”（该文件中包含《数据结构》中常用常量的定义）

```
/* c1.h (程序名) */
 #include<string.h>
 #include<ctype.h>
 //#include<malloc.h>       /* malloc()等  */
 #include<limits.h>         /* INT_MAX 等  */
 #include<stdio.h>          /* EOF(=^Z 或 F6),NULL */
 #include<stdlib.h>         /* atoi() */
// #include<io.h>            /* eof() */
 #include<math.h>           /* floor(),ceil(),abs() */
// #include<process.h>       /* exit() */
 /*  函数结果状态代码  */
 #define TRUE 1
 #define FALSE 0
 #define OK 1
 #define ERROR 0
 #define INFEASIBLE -1
 /* #define OVERFLOW -2  因为在 math.h 中已定义 OVERFLOW 的值为 3,故去掉此行 */
 typedef int Status; /* Status 是函数的类型,其值是函数结果状态代码，如 OK 等  */
 typedef int Boolean; /* Boolean 是布尔类型,其值是 TRUE 或 FALSE */
```

(2)用以下源代码建立头文件“c2-1.h”（该文件是线性表的顺序存储结构定义）

```
/* c2-1.h  线性表的动态分配顺序存储结构  */
 #define LIST_INIT_SIZE 10 /*  线性表存储空间的初始分配量  */
 #define LISTINCREMENT 2 /*  线性表存储空间的分配增量  */
 typedef struct
 {
    ElemType *elem;         /*  存储空间基址  */
    int length;             /*  当前长度  */
    int listsize;           /*  当前分配的存储容量(以 sizeof(ElemType)为单位) */
 }SqList;
```

(3)用以下源代码建立文件“bo2-1.c”（该文件包含线性表的顺序存储结构的基本操作函数）

```
/* bo2-1.c 顺序表示的线性表(存储结构由 c2-1.h 定义)的基本操作(12 个) */
 Status InitList(SqList *L) /* 算法 2.3 */
 { /* 操作结果：构造一个空的顺序线性表 */
   (*L).elem=(ElemType*)malloc(LIST_INIT_SIZE*sizeof(ElemType));
   if(!(*L).elem)
     exit(OVERFLOW); /* 存储分配失败 */
   (*L).length=0; /* 空表长度为 0 */
   (*L).listsize=LIST_INIT_SIZE; /* 初始存储容量 */
   return OK;
 }

 Status DestroyList(SqList *L)
 { /* 初始条件：顺序线性表 L 已存在。操作结果：销毁顺序线性表 L */
   free((*L).elem);
   (*L).elem=NULL;
   (*L).length=0;
   (*L).listsize=0;
   return OK;
 }

 Status ClearList(SqList *L)
 { /* 初始条件：顺序线性表 L 已存在。操作结果：将 L 重置为空表 */
   (*L).length=0;
   return OK;
 }

 Status ListEmpty(SqList L)
 { /* 初始条件：顺序线性表 L 已存在。操作结果：若 L 为空表，则返回 TRUE，否则返回 FALSE */
   if(L.length==0)
     return TRUE;
   else
     return FALSE;
 }

 int ListLength(SqList L)
 { /* 初始条件：顺序线性表 L 已存在。操作结果：返回 L 中数据元素个数 */
   return L.length;
```

```
}

Status GetElem(SqList L,int i,ElemType *e)
{ /* 初始条件：顺序线性表 L 已存在，1≤i≤ListLength(L) */
  /* 操作结果：用 e 返回 L 中第 i 个数据元素的值 */
  if(i<1||i>L.length)
    exit(ERROR);
  *e=*(L.elem+i-1);
  return OK;
}

int LocateElem(SqList L,ElemType e,Status(*compare)(ElemType,ElemType))
{ /* 初始条件：顺序线性表 L 已存在，compare()是数据元素判定函数(满足为 1,否则为 0) */
  /* 操作结果：返回 L 中第 1 个与 e 满足关系 compare()的数据元素的位序。 */
  /*           若这样的数据元素不存在，则返回值为 0。算法 2.6 */
  ElemType *p;
  int i=1; /* i 的初值为第 1 个元素的位序 */
  p=L.elem; /* p 的初值为第 1 个元素的存储位置 */
  while(i<=L.length&&!compare(*p++,e))
    ++i;
  if(i<=L.length)
    return i;
  else
    return 0;
}

Status PriorElem(SqList L,ElemType cur_e,ElemType *pre_e)
{ /* 初始条件：顺序线性表 L 已存在 */
  /* 操作结果：若 cur_e 是 L 的数据元素，且不是第一个，则用 pre_e 返回它的前驱，*/
  /*           否则操作失败，pre_e 无定义 */
  int i=2;
  ElemType *p=L.elem+1;
  while(i<=L.length&&*p!=cur_e)
  {
    p++;
    i++;
  }
  if(i>L.length)
    return INFEASIBLE;
```

```
    else
    {
      *pre_e=*--p;
      return OK;
    }
  }

  Status NextElem(SqList L,ElemType cur_e,ElemType *next_e)
  { /* 初始条件：顺序线性表 L 已存在 */
    /* 操作结果：若 cur_e 是 L 的数据元素，且不是最后一个，则用 next_e 返回它的后继， */
    /*              否则操作失败，next_e 无定义 */
    int i=1;
    ElemType *p=L.elem;
    while(i<L.length&&*p!=cur_e)
    {
      i++;
      p++;
    }
    if(i==L.length)
      return INFEASIBLE;
    else
    {
      *next_e=*++p;
      return OK;
    }
  }

  Status ListInsert(SqList *L,int i,ElemType e) /* 算法 2.4 */
  { /* 初始条件：顺序线性表 L 已存在，1≤i≤ListLength(L)+1 */
    /* 操作结果：在 L 中第 i 个位置之前插入新的数据元素 e，L 的长度加 1 */
    ElemType *newbase,*q,*p;
    if(i<1||i>(*L).length+1) /* i 值不合法 */
      return ERROR;
    if((*L).length>=(*L).listsize) /* 当前存储空间已满,增加分配 */
    {
      newbase=(ElemType
*)realloc((*L).elem,((*L).listsize+LISTINCREMENT)*sizeof(ElemType));
      if(!newbase)
        exit(OVERFLOW); /* 存储分配失败 */
      (*L).elem=newbase; /* 新基址 */
```

```
    (*L).listsize+=LISTINCREMENT; /* 增加存储容量 */
  }
  q=(*L).elem+i-1; /* q 为插入位置 */
  for(p=(*L).elem+(*L).length-1;p>=q;--p) /* 插入位置及之后的元素右移 */
    *(p+1)=*p;
  *q=e; /* 插入 e */
  ++(*L).length; /* 表长增 1 */
  return OK;
}

Status ListDelete(SqList *L,int i,ElemType *e) /* 算法 2.5 */
{ /* 初始条件：顺序线性表 L 已存在，1≤i≤ListLength(L) */
  /* 操作结果：删除 L 的第 i 个数据元素，并用 e 返回其值，L 的长度减 1 */
  ElemType *p,*q;
  if(i<1||i>(*L).length) /* i 值不合法 */
    return ERROR;
  p=(*L).elem+i-1; /* p 为被删除元素的位置 */
  *e=*p; /* 被删除元素的值赋给 e */
  q=(*L).elem+(*L).length-1; /* 表尾元素的位置 */
  for(++p;p<=q;++p) /* 被删除元素之后的元素左移 */
    *(p-1)=*p;
  (*L).length--; /* 表长减 1 */
  return OK;
}

Status ListTraverse(SqList L,void(*vi)(ElemType*))
{ /* 初始条件：顺序线性表 L 已存在 */
  /* 操作结果：依次对 L 的每个数据元素调用函数 vi()。一旦 vi()失败，则操作失败
*/
  /*              vi()的形参加'&'，表明可通过调用 vi()改变元素的值 */
  ElemType *p;
  int i;
  p=L.elem;
  for(i=1;i<=L.length;i++)
    vi(p++);
  printf("\n");
  return OK;
}
```

(4)请你自己编写“main2-1.c”，在其中编写 main 函数，调用 bo2-1.c 中定义的基本函数来实现题目 4 要求的功能（以下是“main2-1.c”的程序框架）。

```
/* main2-1.c 检验 bo2-1.c 的主程序 */
 #include "c1.h"
 typedef char ElemType;
 #include "c2-1.h"
 #include "bo2-1.c"

 int main()
{   SqList L;
    ElemType e,a[5]={'a','b','c','d','e'};
    int i,j;

    printf("初始化 L 前：L.elem=%p L.length=%d L.listsize=%d\n",L.elem,L.length,
L.listsize);

    /*(1) 初始化顺序表 L； */
    i=InitList(&L);
    printf("初始化 L 后：L.elem=%p L.length=%d L.listsize=%d\n",L.elem,L.length,
L.listsize);

    /*(2)依次采用尾插法插入’a’,’b’,’c’,’d’,’e’元素； */
    for(j=1;j<=5;j++)
       i=ListInsert(&L,j,a[j-1]);

    //在此编写上机题目 4 要求实现的其它功能!
    /*(3)输出顺序表 L； */

    /*(4)输出顺序表 L 长度； */

    /*(5)输出顺序表 L 的第 3 个元素； */

    /*(6)在第 4 个元素位置上插入’f’元素； */

    /*(7)输出顺序表 L； */

    /*(8)删除 L 的第 3 个元素； */

    /*(9)输出顺序表 L； */

    /*(10)释放顺序表 L。*/

    return(1);
```

```
}
```

5. 在给定的框架 main2-2.c（见学院网上教学《数据结构》课程中的 HW5-S.rar 文件）中，补充完成单链表的各种基本运算，并在此基础上设计一个主程序完成如下功能：

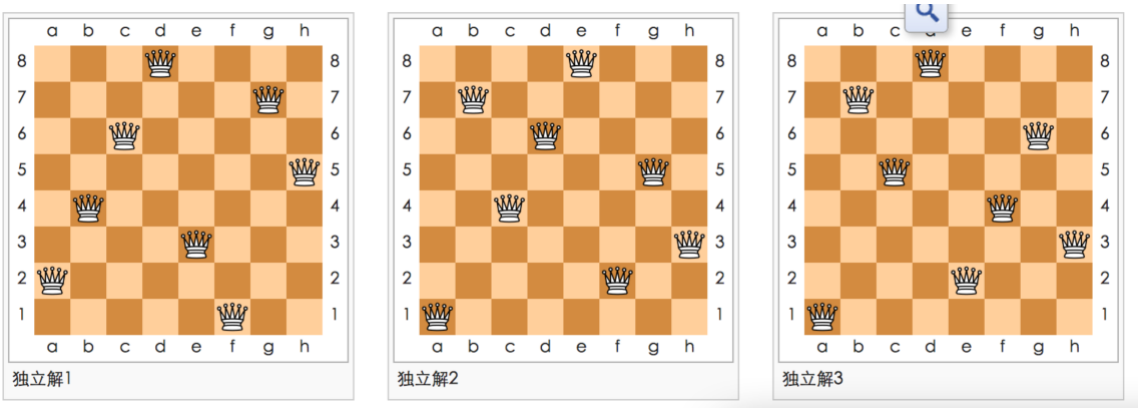
（1）初始化单链表 L；
（2）依次采用头插法插入 1,2,3,4,5 元素；
（3）输出单链表 L；
（4）输出单链表 L 长度；
（5）输出单链表 L 的第 3 个元素；
（6）在第 4 个元素位置上插入 888 元素；
（7）输出单链表 L；
（8）删除 L 的第 3 个元素；
（9）输出单链表 L；
（10）释放单链表 L。

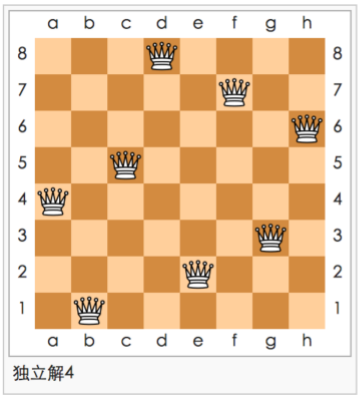
6. 设计一个程序 algo2-3.c，设计一个包含 n 个元素的带头结点的循环单链表，不调用教材中提及的基本运算，自己编写完整的程序将其逆序，要求输出逆序前后的序列。

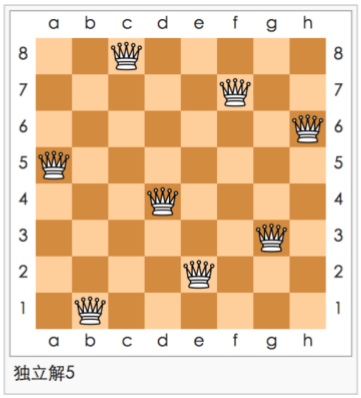
7. 设计一个程序 algo3-1.c，求解 N 皇后问题（N=3～8）。可使用循环或递归方法；尽可能使用图形界面演示 N 皇后的各种摆放方案。

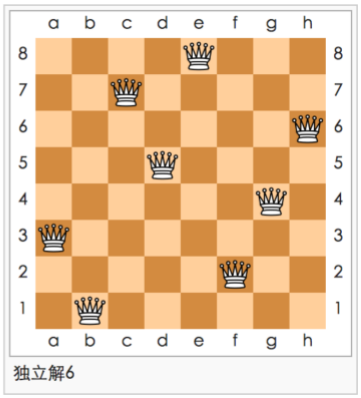
提示：八皇后问题是一个以国际象棋为背景的问题：如何能够在 8×8 的国际象棋棋盘上放置八个皇后，使得任何一个皇后都无法直接吃掉其他的皇后？为了达到此目的，任两个皇后都不能处于同一条横行、纵行或斜线上。八皇后问题可以推广为更一般的 n 皇后摆放问题：这时棋盘的大小变为 n×n，而皇后个数也变成 n。当且仅当 $n = 1$ 或 $n \geq 4$ 时问题有解。

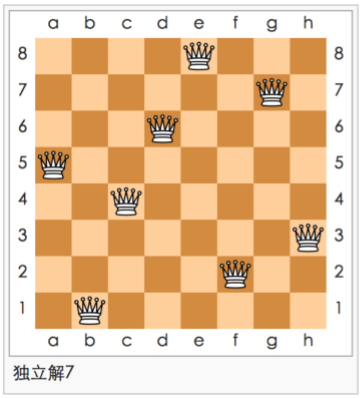
八皇后问题一共有 92 个互不相同的解。如果将旋转和对称的解归为一种的话，则一共有 12 个独立解，具体如下：

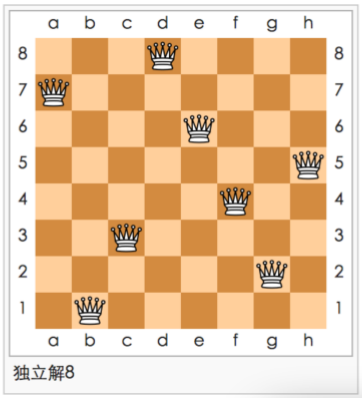


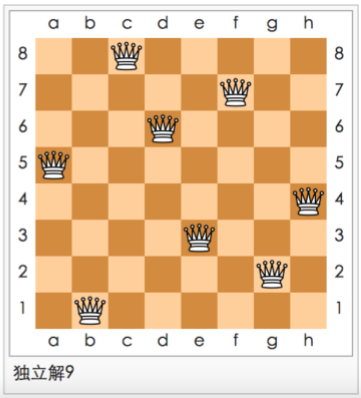


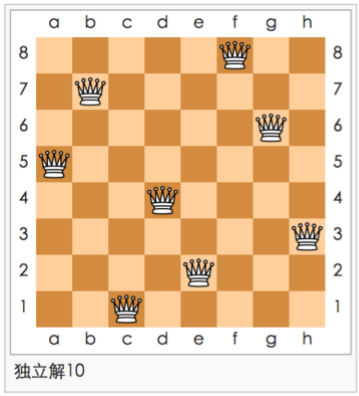


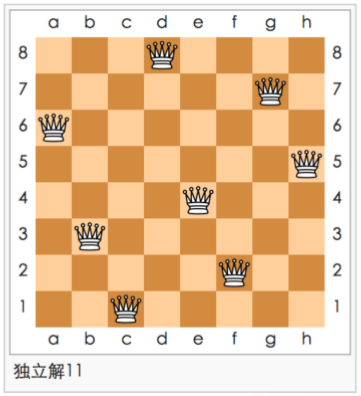


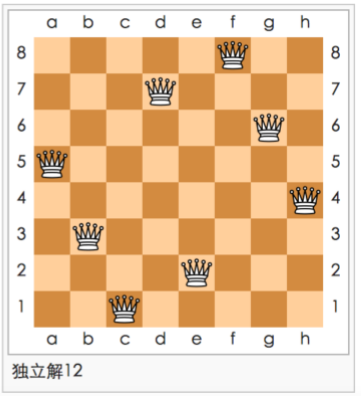












8. 设计一个程序 algo5-1.c,将数组 A[n]的两段数据对换，两段数据的下标分别为：p0，p1 与 p2，p3，且 0≤p0≤p1<p2≤p3≤n-1。要求程序使用的附加空间与 n 无关。

9. 设计一个程序 algo6-1.c，实现二叉树的各种运算，并实现中序遍历二叉树的非递归算法以及先序、中序和后序遍历的递归算法。

10. 设计一个程序 algo9-1.c，输出在顺序表{1，2，3，4，5，6，7，8，9，10}中采用二分查找法查找关键字 9 的过程。

11. 奇偶交换排序如下所述：第一趟对所有奇数 i，将 a[i]和 a[i+1]进行比较；第二趟对所有的偶数 i，将 a[i]和 a[i+1]进行比较，若 a[i]>a[i+1]，则将两者交换，第三趟对奇数 i;第四趟对偶数 i，…，依次类推直至整个文件有序为止。设计一个程序

algo10-1.cpp 实现奇偶排序算法。