第六章　指针与引用

6.1　示范题的求解与剖析

一.　示范题一

读程序写结果。程序中使用了与指针相关的常用运算：取内容运算“*”，取地址运算“&”，指针加减一个常数（结果仍为指针），两个指针进行比较，两个指针求差运算等。

```
#include <iostream.h>
void main() {
    int a[10]={10,11,12};
    int *p=a, *q=&a[9];
    *(p+3)=13;  *(q-5)=*(p+3)+1;
    for (int i=5; i<10; i++)
        *(p+i)=80+i;
    cout<<"*(p+2)="<<*(p+2)<<endl;
    cout<<"*(&a[8])="<<*(&a[8])<<endl;
    for (i=3; i<8; i++)
        cout<<*(a+i)<<"  ";
    cout<<"\nq-a="<<q-a<<endl;
    cout<<"q-p="<<q-p<<endl;
    while (q>&a[4])
        cout<<*q--<<"  ";
    cout<<"\nq-a="<<q-a<<endl;
}
```

【实验目的】

熟悉指针概念、指针的定义方式以及与指针相关的常用运算（*、&，数组指针的算术运算及关系运算、指针求差运算等）。

【实现方法】

1．通过说明语句“int *p=a, *q=&a[9];”，说明了两个指针变量 p 和 q，并给它们赋初值，使 p 指向 a[0]，使 q 指向 a[9]。实际上，数组名 a 是一个常量指针（常量地址），它总等同于&a[0]。

2．由于 p 指向 a[0]而 q 指向 a[9]，所以“*(p+3)=13;”意味着在给数组元素 a[3]赋值，而语句“*(q-5)=*(p+3)+1;”就相当于“a[4]=a[3]+1;”。

3．本题使用了取指针所指内容的运算（如，*q，*(p+i)等）；还使用了两个指针求差的运算（如，q-a 以及 q-p，注意这些指针要指向同一个数组，差为二指针间元素的个数）。

4．程序中使用的“*(&a[8])”就等同于 a[8] -- 先取 a[8]的地址（为一个指针），而后又取该指针所指向的内容。而*(a+i)也总等同于 a[i]，所以“cout<<*(a+i);”也就等同

于“cout<<a[i];”。

5．由于 q 指向了 a 数组，从而可进行“q>&a[4]”这样的指针比较运算（规定元素靠前者其指针值小）。

【程序编制】

具体程序见题目处，从略。

【调试运行】

程序执行后的输出结果为：

```
*(p+2)=12
*(&a[8])=88
13  14  85  86  87
q-a=9
q-p=9
89  88  87  86  85
q-a=4
```

【剖析点评】

注意下面三行中出现的三个“*pi”，它们的使用含义是不相同的：

```
int i=12, *pi=&i;
cout<<*pi;
*pi = 34;
```

第一行的“*pi”处于变量说明处，是说明 pi 为“int*”型的变量，并同时将该指针变量初始化为 i 的地址。不可将此处的“*”理解为“取内容”运算，它与前面的 int 联合起来以说明 pi 为“int*”型即指向 int 型数据的指针型变量。

第二行的“*pi”表示指针变量 pi 所指向的那一变量即 i 的内容（“*”理解为“取内容”运算，输出 i 的内容即 i 值 12）。

第三行的“*pi”为左值（存储空间概念），表示要改变指针变量 pi 所指向的那一变量（即 i）空间中的内容（使用其存储空间），使 i 值改变为 34。

二． 示范题二

编制具有如下原型的函数 findMax，负责在数组 a 的 n 个元素中找出最大值，并返回该最大值数组元素的内存地址（指针值），而且再通过引用 idx 返回具有最大值的元素在数组中的下标。

```
int* findMax(int* a, int n, int& idx);
```

而后编制如下式样的主函数，调用 findMax，以验证其正确性。

```
void main() {
    int b[10] = {34, 7, 23, 89, 11, -2, 100, 11, 5, 66};
    int *maxAddr, maxIdx;
    maxAddr = findMax(b, 10, maxIdx);    //对 findMax 函数进行调用，返回指针值
```

```
    //实参 b 为数组名，是一个指针；实参 maxIdx 对应于引用形参，可带回结果值
    cout << "Max value index: " << maxIdx << endl;
    cout << "Max value address: " << maxAddr << endl;
    cout << "Max value: " << *(b+maxIdx) << endl;
}
```

程序执行后的输出结果样式为：

```
Max value index: 6
Max value address: 0x0012FF70
Max value: 100
```

【实验目的】

1. 设计并编制自定义函数 findMax，完成指定功能；而后在 main 中调用 findMax 验证其正确性。

2. 对 int 型指针参数以及 int 型引用参数的使用：函数第一参数为 int 型指针 a，调用该函数时，对应实参可以为一维整型数组（名）；函数第三参数为 int 型引用 idx，调用时的对应实参必须为 int 型变量（如 maxIdx），调用结果将放入该变量中带回到调用函数。

【实现方法】

函数 findMax 中，对以 a 为首地址的数组各元素依次进行处理：通过比较求出元素最大值，其处理程序“构架”如下：

```
    idx = 0;                    //idx 代表具有最大值的数组元素下标值
    for(int i=1; i<n; i++)
        if ( *(a+i) > *(a+idx) )
            ...;                //找到更大者时，改变 idx 值
    return  ...;                //返回最大值元素所在内存地址
```

【程序编制】

```
#include <iostream.h>
int* findMax(int *a, int n, int &idx);  //函数原型
void main() {
    ...                                 //同题目处，从略
}
int* findMax(int* a, int n, int& idx) {
    idx = 0;                            //idx 代表具有最大值的数组元素下标值
    for(int i=1; i<n; i++)
        if ( *(a+i) > *(a+idx) )
            idx = i;                    //找到更大者时，改变 idx 值
    return (a+idx);                     //返回最大值元素所在内存地址
}
```

【调试运行】

注意：在不同计算机软硬件环境下，findMax 函数的返回值（最大值数组元素的内存地址，即指针值）可能会不相同。本例结果处给出的“Max value address: 0x0012FF70”只是在某机器环境下的取值情况。

【剖析点评】

与非字符型一维数组作为函数参数相同，非字符型指针参数（如 findMax 的第一参数 a）作为数组首地址进行传递时，也需要靠随后的另一个参数（如第二参数 n）来指出要处理多少个数组分量。即是说，实参数组带来的只是在内存中连续存放的那一批数组元素的首地址（指针值）。

三. 示范题三

下面的程序片段首先通过指针配合 new 运算符生成了一个动态的二维数组 a(其行数 lin 及列数 col 通过 cin 临时输入)，而后又为数组各元素赋了值。请在此基础上编制完整程序并上机进行调试运行，并输出结果。

```
int lin,col,i,j;
cin>>lin>>col;            //任意输入行数 lin 及列数 col
int **a;
a = new int*[lin];        //lin（行数）个 int* 指针
for(i=0;i<lin;i++)
    a[i]=new int[col];    //每行有 col（列数）个 int 数
for(i=0;i<lin;i++)
    for(j=0;j<col;j++)
        a[i][j]=i+j;      //注意 a[i][j]等同于*(a[i]+j)，又等同于*( *(a+i)+j)
```

【实验目的】

通过使用 new 运算符来生成具有动态大小的数组空间（动态数组），并对所生成的动态数组进行使用。

【实现方法】

1. 通过 do-while 型循环语句，输入必须为正整数的行列数 lin 及 col 之值。

2. 通过使用“a = new int*[lin];”，动态生成了一组共 lin（行数）个“int*”指针空间，使各分量*(a+i)也即 a[i]均为指向 int 型数据的指针（空间）。

3. 循环语句“for(i=0; i<lin; i++) a[i]=new int[col];”执行后，将为每行动态生成 col（列数）个 int 型存储空间。注意，a[i]即为*(a+i)，为它们分别赋值，将使它们指向各自的一组具有 col 个 int 型数据的存储空间。

4. 访问（存取）上述动态生成的第 i 行第 j 列的 int 型存储空间时，应该使用如下的分量表达方式“*(*(a+i)+j)”，但注意，它又等同于“*(a[i]+j)”，而且还等同于“a[i][j]”，所以，可选择自己喜爱的上述三种方式中的某一种去使用。

【程序编制】

```
#include <iomanip.h>
void main() {
    int lin,col,i,j;
    do{
        cout<<"lin, col = ? ";
        cin>>lin>>col;          //输入行数 lin 及列数 col
    } while (lin<=0 || col<=0); //要求输入的行列数必须为正整数
    int **a;                    //a 为指向指针的指针（二重指针）
    a = new int*[lin];          //动态生成一组共 lin（行数）个“int*”指针空间
    for(i=0; i<lin; i++)
        a[i]=new int[col];      //为每行动态生成 col（列数）个 int 型存储空间
    for(i=0; i<lin; i++)        //为数组各元素按指定公式赋值
        for(j=0; j<col; j++)
            a[i][j]=i+j;        // a[i][j]等同于*(a[i]+j)与 *( *(a+i)+j)
    for(i=0; i<lin; i++) {
        for(j=0; j<col; j++)
            cout<<setw(3)<<a[i][j]; //屏幕显示各数组元素（每数域宽为 3 字符）
        cout<<endl;
    }
}
```

【调试运行】

程序执行后的输入输出结果式样如下：

```
lin, col = ? 4 5
  0  1  2  3  4
  1  2  3  4  5
  2  3  4  5  6
  3  4  5  6  7
```

【剖析点评】

C++对数组的说明（定义）必须是静态性的，就是说，数组分量的多少要说明成固定的（在编译阶段，就可确定出数组的大小，为其分配适当大小的存储空间）。如，试图通过“int n; cin>>n; int a[n];”来说明（定义）动态大小之数组 a 的方法是错误的（语法错，要求 n 必须是常量）!

利用本题提供的通过使用 new 运算符的方法，则可实现具有动态大小的数组。实际上，使用此种方法后，则是在执行程序时（运行阶段）方才对数组空间进行分配，从而具有了所谓的动态性。

四． 示范题四

任意输入 k 个整数（k 值通过键盘输入指定），先将它们按从前到后的顺序存放到由结构体形成的链表中；再输入一个整数 x，而后查找 x 是否在所形成的链表中出现；若出现的话，输出它在链表中首次出现时的序号（为第几项），否则输出“不存在”的提示信息。

可定义并使用如下的数据结构：

```
struct item {        //使用 item 结构体来形成链表存放数据
    int num;         //数据放 num
    item* next;      //通过 next“串联”起后项
};
```

例如，输入 8 个数，而后查找数据 72 的首次出现位置号，其输入输出界面可设计为：

```
Input a positive number! k=? 8
Input 8 integers!
-3 55 72 26 0 72 -28 99
Looking for a number x! x=? 72
Fonnd 72 in position 3
```

【实验目的】

1．通过使用结构体来“串联”起一个可以存放数据的动态链表，且该结构体的其中某一分量（如本例的 next 分量）必须为指向本结构体的指针。

2．通过使用 new 运算符依次生成各动态链表项，并通过 next 指针将它们“串联”成为链表；要查找数据是否出现，将对链表进行遍历。

【实现方法】

1．使用如下的程序“构架”来生成一个具有 k 项的链表（而总将新链表项加入到当前已有链表的末尾）。

```
item *head, *tail, *temp;          //head 指向链表首项，tail 指向末项
tail = head = NULL;                //使 head 及 tail 均指向“空”，表示空链表
for(i=0; i<k; i++) {               //形成一个具有 k 个项的链表
    temp=new item;                 //生成一个 item 型的动态新表项
    temp->next=NULL;               //新表项将充当链表末项，将其 next 域置为 NULL
    ...                            //为(*temp)结构变量的其它各分量“定值”
    if(head==NULL)
        head=tail=temp;            //链表为空时，新表项既为首项又为末项
    else {                         //链表非空时
        tail->next=temp;           //新表项加入到原链表的末项之后
        tail=temp;                 //新表项成为链表的新末项
    }
}                                  //i 循环体结束
```

2．使用如下的程序“构架”来遍历链表（从头到尾依次处理一遍各链表项）

```
temp=head;
while (temp!=NULL) {               //基于我们已将链表末项的 next 域置成了 NULL
    ...                            //处理(*temp)结构体即链表项中的数据
    temp=temp->next;               //使 temp 指向下一个链表项
```

```
    }
```

【程序编制】

```
#include <iostream.h>
void main() {
    struct item {                    //自定义 item 结构体类型
        int num;                     //num 分量存放数据
        item* next;                  //next 分量“串联”起后项
    };
    int k;
    do {
        cout<<"Input a positive number!  k=? ";
        cin>>k;
    } while (k<=0);                  //输入正整数 k
    item *head, *tail, *temp;        //head 指向链首，tail 指向链尾
    tail = head = NULL;              //形成空链表
    cout<<"Input "<<k<<" integers!"<<endl;
    for(int i=0; i<k; i++){          //链表共 k 项
        temp=new item;               //生成新表项
        temp->next=NULL;             //新表项将充当链表末项
        cin>>temp->num;              //通过 cin 输入(*temp)结构变量的 num 分量值
        if(head==NULL)               //链表为空时
            head=tail=temp;          //新表项既为首项又为末项
        else {                       //链表非空时
            tail->next=temp;         //加入到末项之后
            tail=temp;               //改变新末项
        }
    }
    cout<<"Looking for a number x!  x=? ";
    int x, idx=0, found=0;
        //idx 记录被查找数据 x 的序号（为第几项），found 标记是否查到
    cin>>x;
    //遍历链表，对链表的各项内容进行处理（查找 x 在链表中的首次出现）
    temp=head;
    while(temp!=NULL) {              //基于我们已将链表末项的 next 域置成了 NULL
        idx++;                       //增长链表项序号
        if (x==temp->num) {          //查到了 x
            found=1;
            break;                   //跳出循环
        }
        temp=temp->next;             //使 temp 指向下一个链表项，接着查找
    }
    if (found)
```

```
        cout<<"Fonnd "<<x<<" in position "<<idx<<endl;
    else
        cout<<"Not fonnd "<<x<<" in the queue!"<<endl;
```

【调试运行】

当输入数据中没出现被查找数据时，程序执行结果式样如下：

```
Input a positive number!  k=? 6
Input 6 integers!
11 22 33 44 55 66
Looking for a number x!  x=? 88
Not fonnd 88 in the queue!
```

【剖析点评】

可对程序进行改进，当被查找的数 x 在所形成的链表中多次出现时，要统计出共出现了多少次，而且要一并输出它们各自在链表中出现的序号（各为第几项）。

例如，输入 10 个数，而后查找-6 出现的次数及各序号，输入输出界面可设计为：

```
Input a positive number!  k=? 10
Input 10 integers!
13 35 -6 0 -6 782 -91 58 4 -6
Looking for a number x!  x=? -6
Fonnd -6 in position: 3 5 10
Total fonnd: count = 3
```

6.2　实践题的求解方法与提示

一.　实践题一

读程序，写出运行结果。注意其中通过指针参数实现了主调函数与被调函数间的“双向传值”。

```
#include <iostream.h>
void split(double x, int *iPart, double *fPart) {
    *iPart = int(x);
    *fPart = x - *iPart;
}
double f(double *a, int n) {
    int i, intPt;
    double fracPt, maxfracPt=0;
    for(i=0; i<n; i++)  {
        split( *(a+i), &intPt, &fracPt );
        if( fracPt > maxfracPt)
            maxfracPt = fracPt;
```

```
        *(a+i) = intPt;
    }
    return maxfracPt;
}
void main() {
    int i;
    double maxfr, a[6] = {1.1, 2.2, 3.3, 9.9, 6.6, 5.0};
    maxfr = f(a, 6);
    cout<<"After call f, a[0..5] = ";
    for(i=0; i<6; i++)  cout<<a[i]<<"  ";
    cout<<"\nmaxfr="<<maxfr<<endl;
}
```

【实验目的】

1．通过设置指针参数来实现主调函数与被调函数间的“双向传值”。

2．编制出由 main 调用 f、而 f 又调用 split 的嵌套式函数调用的处理程序。

【实现方法】

1．函数 split 负责“分离”出 x 的整数部分与小数部分，分别放于*iPart 与*fPart 处，由于形参 iPart 与 fPart 都是指针，从而可实现“双向传值”而将这两个结果同时“带回去”。

2．函数 f 负责从指针 a 为首地址的位置开始，对最前面的 n 个 double 型数进行处理：通过调用 split 改变数组各元素值（只留下其整数部分），并返回“分离”后之小数部分的最大者。

3．主函数 main 中使用“maxfr = f(a, 6); ”形式的调用语句，通过函数 f 将数组 a 的 6 个元素值进行了更改（a 作为实参指针即数组元素的首地址被传给了 f，函数 f 中改变数组元素值并通过“双向传递”将结果仍然放在了 a 数组中）。

【编程提示】

要通过指针形参的使用来实现所谓的“双向传值”的话，必须在被调函数中更改此形参指针所指向的变量值（注意，并不是更改形参指针变量本身的值）。

例如，split 函数中具有指针形参 iPart，而且在函数体中使用“*iPart = int(x);”，就意味着是更改主调函数中相应实参指针所指向的变量值。而 f 函数中具有指针形参 a，而且在函数体中使用“*(a+i) = intPt;”，也意味着是更改主调函数中相应实参数组的元素值。

二． 实践题二

编制具有如下原型的函数 FindPlace，该函数返回字符串 str 中第一次出现字符 c 的位置（地址值，即指向字符 c 的指针）。如果不出现 c 字符则返回空指针 NULL。

```
char * FindPlace(char *str, char c);
```

并编制主函数对它进行调用以验证其正确性（输出从返回指针开始直到串尾的那一子串；或输出“No match found”提示串，当 FindPlace 返回空指针 NULL 时）。

【实验目的】

编制并使用返回指针类型值的自定义函数。

【实现方法】

1. FindPlace 函数中，从头到尾（直到结束符“\0”）逐个查看各 str[i]是否等于 c，若相等的话，要返回那一 str[i]字符的地址（即 &str[i]）。若整个串中根本不出现 c 字符的话，返回空指针 NULL。

2. 主函数 main 中，可首先进行说明：

```
char s[80], *p, c;
```

而后输入字符串 s，以及一个字符 c，之后进行如下的调用并将返回结果赋值给字符指针 p。

```
p = FindPlace(s, c);
```

最后再将结果输出。

【编程提示】

1. 执行程序后的输入输出界面样式可设计为：

```
Input a string:
Hello world!
Input a character:o
Sub_string =>o world!
```

2. 注意：若返回的字符指针值为 p 的话，那么，通过“cout<<p;”形式的语句输出的将是从 p 指针所指字符开始的子串（系统对输出字符指针值如 p 之值是特殊对待的，对其他类型指针值的输出则只是输出一个地址值）。

3. 思考：若寻找并返回 c 字符在字符串 str 中最后一次的出现位置时，FindPlace 函数中，只需倒着从尾部向前逐个查看各 str[i]即可，实现方式相类似。

三. 实践题三

编制具有如下原型的函数：

```
void chgStr(char *ip, char *op);
```

负责将 ip 所指输入串中的各字符按照指定规则进行变换后、存放到由 op 指向的输出串中。

即，要从头到尾逐字符地对 ip 所指向的字符串进行如下处理：

（1）若 ip 所指当前字符为字母，则将其改变大小写后存放到结果串 op 中。

（2）若 ip 所指当前字符是一个数字字符，则将其变换为另一个数字字符后存放到结果串 op 中：字符 0 变换为字符 9，字符 1 变换为字符 8，字符 2 变换为字符 7，...，字符 9 变换为字符 0。

（3）若 ip 所指当前字符为其他字符，则将该字符复制到结果串 op 中。

例如：若调用时通过 ip 带来的字符串为“Nankai 1918-2004, x+y-5*6/37=? OK!”，则结果字符串 op 应为“nANKAI 8081-7995, X+Y-4*3/62=? ok!”。

最后编制主函数，对 chgStr 进行调用，以验证其正确性。

【实验目的】

1．设计并编制自定义函数 chgStr，完成指定功能；而后在 main 中调用 chgStr 验证其正确性。

2．通过指针参数的使用实现“双向传值”：函数参数为字符指针；调用该函数时，对应实参为一维字符数组（名）；通过指针参数 op 将结果带回到调用函数中去。

【实现方法】

1．函数 chgStr 中，对以 ip 所指输入串中的各字符依次进行处理，并将结果存放到由 op 指向的输出串中（而后带回到 main 中）。处理程序“构架”如下：

```
while (*ip)                          //从头到尾逐字符地对 ip 串进行处理
    if (*ip>='A' && *ip<='Z')        //当前字符为大写字母时
        *op++ = *ip++ +32;
    else
        if (...)
            ...
        else
            ...
 *op = '\0';                         //添加 op 串尾结束符
```

2．主函数 main 中，调用 chgStr，完成规定任务。

```
    char istr[80], ostr[80];
    ...
    chgStr(istr, ostr);              //对 istr 串进行变换，结果放入 ostr 中
    cout<<"ostr=> "<<ostr<<endl;     //输出改变后（传回来）的 ostr 字符串
```

【编程提示】

1．看第四章的示范题二以及第五章的示范题四，它们处理与该题相类似的问题，实现方法可以相互借鉴与参考。

2．第五章使用的是函数实现方法，其函数参数为一维字符数组，与本题使用字符型指针参数都可以实现“数组值”的“双向传递”，从而可看出指针与数组的密切关系。

3．通过指针参数的使用来实现“双向传值”的方法：在被调函数内，对形参指针所指元素的使用与改变，就意味着是对实参指针所指元素的直接使用与改变。即是说，若在被调函数内改变了形参指针所指元素的值，则返回主调函数后，相应实参指针所指元素的值也进行了相同的改变。

对本实践题，main 中使用的函数调用语句为“chgStr(istr, ostr);”，其实参为两个数组名，代表着两个常量地址即指针值。而 chgStr 中对形参指针 op 所指元素的值进行了改变，从而其对应实参指针 ostr 所指元素的值也发生了变化（传回了结果值）！

4．注意，若函数参数为非字符型的数组参数时（或为非字符型的指针参数时），一般要同时再设计另外一个参数 n，告诉被调函数有多少个数组元素需要处理。但函数参数为字符型数组参数（或为字符型的指针参数）时，通常不必再设计那样的参数 n，因为实参字符数组通常为字符串，被调函数中通过判断结尾符“\0”则可处理完其所有的数组元素（各字

符）。

四． 实践题四

编程序，含有如下形式的循环语句，用于理解并测试 new 与 delete 的功能。

```
double *p;  unsigned long n;  cin>>n;
for (int i=0; i<n; i++) {
    p = new double[32767];
    cout<<"p="<<p<<endl;
    …
    delete p;
}
```

执行后，当输入不同的 n 时，输出的各 P 值有什么不同吗？若将循环改写为“for (int i=0; ; i++){…}”即改成为无限循环的话，程序会无限循环而永不停止吗？

进一步，将其中的“delete p;”语句删除掉后再执行（仅循环 n 次），输出的各 P 值会有什么不同吗？

再进一步，不仅删除 delete，而且改为无限循环的话，又会出现什么情况呢？

【实验目的】

理解并测试 new 与 delete 的功能;为防止出现“内存泄漏”的情况，new 与相应的 delete 应该匹配出现。

【实现方法】

1．程序中的 for 循环要执行 n 次，而每一次循环总是先通过 new 来申请一块很大的动态存储空间,而后输出该块空间的首地址,随后则通过 delete 立即将此块动态空间释放掉(归还给了系统，使得下次还可以对此块空间进行重新分配使用）。所以，n 次循环中输出的 p 值（地址即指针值）都完全相同（虽然在不同机器环境下的所输出的地址即指针值的大小可能有所不同）。

2．将 delete 语句删除掉以后再执行，屏幕输出的 n 个 p 值则会互不相同，其原因是只分配不归还，系统只能重新分配另一块空间供使用。

3．若删除 delete 且将 for 语句改成为无限循环的话，程序最终将出现“无存储空间可供分配”的错误。其原因是，由于总通过 new 分配新内存空间，但又从不归还它们，而可供编译器分配的动态存储空间又是有限的，从而当再没有动态存储空间可供分配的时候（此时通过 new 返回的指针值为 NULL），那时将会出现运行错误。程序中可通过判断返回的 p 值是否为 NULL 而对这种情况进行处理（如，跳出 for 循环而终止程序等）。

【编程提示】

注意：程序中凡通过 new 动态申请的存储空间，都应该在使用结束后，用相应的 delete 将它们释放掉（归还给系统），否则就有可能产生所谓的“内存泄漏”。

五． 实践题五

编程序，按如下方法求 A 矩阵的转置矩阵 B：输入两个正整数 m 和 n，而后通过使用指针配合 new 运算符生成一个 m 行 n 列的二维动态数组 A 以及另一个 n 行 m 列的二维动态数组 B，之后为 A 输入数据（A 矩阵数据），进而求出其转置矩阵 B（数据放动态数组 B 中）并输出结果。

【实验目的】

通过使用运算符 new 生成动态数组，并对这些动态数组进行使用。

【实现方法】

1．说明两个二重指针“int **A, **B;”，并参照本章示范题三的编程模式，动态生成具有 m 行 n 列的二维动态数组 A 以及另一个具有 n 行 m 列的二维动态数组 B。

2．为 A 的数组元素输入数据，并求出其转置矩阵 B（B[j][i]=A[i][j]），而后输出结果。

【编程提示】

程序执行后的输入输出界面可设计为：

```
m, n = ? 3 4
A=? 1 2 3 4 5 6 7 8 9 10 11 12
----- A -----
  1  2  3  4
  5  6  7  8
  9 10 11 12
----- B -----
  1  5  9
  2  6 10
  3  7 11
  4  8 12
```

六． 实践题六

编程序，通过使用如下类型的结构来形成动态链表：

```
struct item {                  //结构类型，用于形成链表项
    int data;                  //存放数据
    item * next;               //指向本结构的指针，由它“串联”起后项
};
```

而后从键盘输入 10 个 int 型数据，并将它们存放在通过指针和 new 而动态生成的各链表项之中（要求总是将新链表项“插入”到当前已有链表的头位置处）；最后再从链表中取出各数据并显示在屏幕上（后放入的必然先取出，从而实现了反序输出问题）。

【实验目的】

通过使用结构类型、指针和 new 运算符，动态生成一个链表，并将输入数据存放在动态生成的链表项中而后使用。

【实现方法】

“输入 10 个数据并将它们存放在生成的各链表项之中”的程序“构架”：

```
item *first=NULL, *temp;
int i, x;
cout<<"Enter 10 integer numbers:"<<endl;
for(i=0; i<10; i++) {
    cin>>x;
    temp=new item;
    temp->data = ...
    temp->next = ...
    first = ...
}
```

【编程提示】

程序执行后的输入输出界面可设计为：

```
Enter 10 integer numbers:
1 3 5 7 9 0 2 4 6 8
---- The result ----
8 6 4 2 0 9 7 5 3 1
```

七. 实践题七

通过使用如下的 person 结构类型，可形成用于管理同类人员（职员，学生或居民等）档案资料的一个链表。

```
struct person {
    char name[12];
    int age;
    char sex;       //  M/m -- 男，  F/f -- 女
    person* next;
};
```

链表可长可短，其功能可通过使用指针以及动态创建和撤消数据对象的运算符 new 和 delete 来完成（参看教材 6.5.2 小节的“构建人员档案链表”及其处理程序）。

编制类似的处理程序，用来完成如下的四项具体工作：

1） 读入若干个人员的档案资料（约定读入的 name 为“*”符号时结束输入），动态生成链表项，并将输入的档案资料存放于链表项之中，而后总将新链表项加入到原链表的末尾；

2） 在链表首加入一项，其 name="wang ping"，age=20，sex='m'；

3） 遍历链表，输出整个链表的各项内容；

4） 统计出当前链表中共有多少男士，并计算出他们的平均年龄。

例如，程序执行后的输入输出界面可设计为：

```
person 1=>name(ending if name='*'):li gang
       1=>age,sex:36 m
person 2=>name(ending if name='*'):guo jie
       2=>age,sex:25 f
person 3=>name(ending if name='*'):zhao qi
       3=>age,sex:30 m
person 4=>name(ending if name='*'):*
---------------
wang ping 20 m
li gang 36 m
guo jie 25 f
zhao qi 30 m
Male-num=3,  average-age=28.6667
```

【实验目的】

链表的生成、使用与管理：创建链表，插入新项，遍历链表，对数据进行指定处理。

【实现方法】

1．说明指向链表首项的指针 head 和指向链表末项的指针 tail，并置初值 NULL，表示空链表。

```
person *head=NULL, *tail=NULL;
```

2．循环读入各人员的档案资料（读入的 name 为“*”符号时结束循环），动态生成链表项用来存放输入数据，且总将链表项加入到以 tail 所指向的链表尾部。该部分程序的“构架”如下：

```
for (i=1; ;i++) {                  //无限循环的 for 语句，要靠 break 跳出循环体
    gets(name);                    //输入第 i 个人的 name
    if (name[0]!='*') {            //若非结束符“*”，则往链表末加入一个表项
        temp=new person;           //动态生成一个新表项
        ...                        //给新表项的 name 域赋值
        ...                        //给新表项的 age 及 sex 域输入值
        ...                        //将新表项加入到原链表的末项之后
     }
     else
        break;                     //输入结束符“*”时跳出循环
}
```

3．要往链表首加入一项，可通过 new 生成新项（如*temp），赋予数据，加入时还要考虑当前的链表是否为“空”：

```
if(head==NULL)
    head=tail=temp;                //链表为空时，新表项既为首项又为末项
else                               //链表非空时，将新表项插入链首
    {
        temp->next=head;
```

```
        head=temp;                    //注意，此 2 句的顺序不可颠倒!
    }
```

4. 输出链表内容、对链表数据进行某些处理，都要对以 head 指向其首项的链表进行“遍历”（从头到尾依次使用与处理一遍链表中的各数据）而完成所指定的任务。

【编程提示】

由于要使用“gets(name);”来输入允许带空格的名字 name，所以程序开头应包含头文件<stdio.h>。

八. 实践题八

读下述程序，其中使用了引用型参数以及返回引用的函数，请给出程序执行后的输出结果并上机进行验证。

```
#include <iostream.h>
int k=56;
void f1(int a, int& b) {
    cout<<"In f1: a, b => "<<a<<", "<<b<<endl;
    a+=10;  b+=10;
}
int& f2(int& a, int& b) {
    cout<<"In f2: a + b => "<<a+b<<endl;
    if( (a+b) %2 ==0 )  return a;
        else  return b;
}
void main() {
    int x=1, y=2, z=3, w=0;
    f1(x,y);
    cout<<"main1: x, y => "<<x<<", "<<y<<endl;
    cout<<"main1: z, k, w => "<<z<<", "<<k<<", "<<w<<endl;
    w=f2(z,k)++;
    cout<<"main2: z, k, w => "<<z<<", "<<k<<", "<<w<<endl;
    w=f2(z,k)++;
    cout<<"main3: z, k, w => "<<z<<", "<<k<<", "<<w<<endl;
}
```

【实验目的】

1. 通过使用引用参数来实现调用函数与被调函数间数据的“双向传值”。

2. 函数返回值为引用，则可将函数调用结果当作“变量”来进行使用（为作为“左值”的存储空间）。

【实现方法】

1．引用参数是其对应实参变量的“替身”，意味着：被调函数中对引用参数值的使用与更改就是对调用语句的对应实参变量值的直接使用与更改，从而可实现调用函数与被调函数间数据的“双向传值”。

2．若函数的返回值为引用，则返回的不仅是一个值，而且还代表一个可以作为“左值”的存储空间，从而可在函数调用结果上直接进行诸如“f2(z,k)++”这样的运算。

3．通过引用参数与指针参数都可以实现调用函数与被调函数间数据的“双向传值”，但通过引用参数更“直接”— 它是实参变量的一个“替身”；而通过指针形参则是“间接”的— 被调函数中要更改形参指针所指向的变量值（而并不是更改形参指针变量本身的值）。

【编程提示】

注意函数 f1 的赋值参数 a 与引用参数 b 的使用区别，以及对返回引用的 f2 函数的使用方法，进而理解程序执行后的如下输出结果：

```
In f1: a, b => 1, 2
main1: x, y => 1, 12
main1: z, k, w => 3, 56, 0
In f2: a + b => 59
main2: z, k, w => 3, 57, 56
In f2: a + b => 60
main3: z, k, w => 4, 57, 3
```

6.3 自立题的求解提示与注意

一. 自立题一

读程序写结果并上机进行验证。注意，并非只要使用指针参数就必然能实现“双向传值”。

```
#include <iostream.h>
#include <string.h>
void f1(int* p, int* q) {
        p=new int;
        *p=66;  *q=66;
        cout<<"*p,*q =>"<<*p<<", "<<*q<<endl;
}
void f2(char* s) {
        cout<<"s+2 =>"<<s+2<<endl;
        char a[20]="C++ Programming!";
        s=a;
        cout<<"s =>"<<s<<endl;
}
void main() {
        int a=2, b=5;
        f1(&a, &b);
        cout<<"a,b =>"<<a<<", "<<b<<endl;
```

```
    char e[10]="123abc#";
    cout<<"e =>"<<e<<endl;
    f2(e);
    cout<<"e =>"<<e<<endl;
}
```

【实验目的】

深入理解指针参数的使用与函数间“双向传值”的关系。

【要点提示】

1．注意，被调函数 f1 中，通过“p=new int;”改变了形参指针 p 的值，使它指向了另外的位置，从而使随后的“*p=66;”无法实现“双向传值”！而被调函数 f2 中也通过“s=a;”改变了形参指针的值，使它指向了另外的位置，这样也不可实现“双向传值”！

就是说，要想真正实现“双向传值”的话，只有在被调函数中改变形参指针所指变量的值（而不是改变形参指针变量本身的值!），那时改变的才是调用函数中实参指针所指变量的值。

2．注意函数 f1 中对指针参数 p 和 q 的不同处理，以及 f2 函数中对 s 指针值的改变，进而理解程序执行后的如下输出结果：

```
*p,*q =>66, 66
a,b =>2, 66
e =>123abc#
s+2 =>3abc#
s =>C++ Programming!
e =>123abc#
```

二． 自立题二

假设在 main 函数中有如下的说明（数组 a 中存放了 n 个字符串，即名字）：

```
const int n=10;
char a[n][31] = {"guo li", "li na",  "li qi", "liu yan", "ma jing","sun li juan", "wang le", "wu da", "yang ke", "zhang yi fu" };
```

编制具有如下原型的自定义函数：

```
int search(char (*p)[31], int n, char* name);
```

负责在字符串数组 p 的前 n 个字符串（名字）中，查找给定串 name 的出现位置（下标值）并返回，若 p 中不出现 name 的话，返回-1。

并编制主函数，输入要查找的某个名字（字符串）name，而后通过如下形式的语句对上述函数进行调用，之后输出相关结果（出现位置，即下标值。若没查到时，给出提示）：

```
int idx = search(a, n, name);      //从 a 数组的第一个名字（0 下标）开始查找
int idx = search(a+3, n, name);    //从 a 数组的第四个名字（3 下标）开始查找
```

【实验目的】

函数参数为指向一维数组的指针时，其使用含义与二维形参数组相类似。

【要点提示】

1．函数 search 的第一个参数 p 被说明为指向一维数组的指针，按照“char (*p)[31]”所说明的 p 实际上可看作是一个二维字符数组，即是说，p[i]即*(p+i)为一个一维字符数组（字符串，至多可存放 31 个字符）；要查找给定串 name 在 p 中的出现位置（下标值），可使用如下形式的 if 语句来实现“if(strcmp(p[i], name) == 0) return (i);”。

2．若将 search 函数对形参 p 的说明“char (*p)[31]”修改为“char p[][31]”，而且其他一切都不必修改，同样可实现本题所给出的求解任务，从而可体会到指向一维数组的指针形参 p 与二维形参数组 p 的密切关系。

3．若将 a 数组中的名字（字符串）按照“字典序”进行了升序排列的话，search 函数中除可使用顺序查找方式之外，还可按照折半法（二分法）来进行查找。

4．程序执行后的输入输出界面可设计为：

```
searching-name=? li qi
search(a, n, name) => idx=2
search(a+3, n, name) => No such name from a[3]!
```

三． 自立题三

编制具有如下原型的函数 findFirst 以及 findLast：

```
char * findFirst(char * sourceStr, char * subStr);
char * findLast(char * sourceStr, char * subStr);
```

findFirst 函数所实现的功能为：返回源串 sourceStr 中第一次出现 subStr 子字符串的头字符位置（地址值）。如果源串 sourceStr 的长度小于 subStr 子字符串的长度，或者源串 sourceStr 中根本就不出现 subStr 子字符串的话，返回空指针值 NULL。

findLast 函数则要返回源串 sourceStr 中最后一次出现 subStr 子字符串的头字符位置。

而后编制主函数，任意输入两个字符串，将它们用作实参来调用这两个函数，以验证其正确性。要求实现程序中不可使用“string.h”头文件内有关寻找子串的标准库函数。

【实验目的】

字符指针的使用以及所定义字符串函数的具体实现。

【要点提示】

1.两个函数都首先求出源串与子字符串的长度，如果源串 sourceStr 的长度小于 subStr 子字符串的长度，则肯定找不到子串而返回 NULL；否则的话，求出源串长度与子串长度的差 count，可知源串 sourceStr 中出现子串的可能性共有 count+1 种。再通过“for (i=0; i<=count; i++) ...”的循环去“从左往右”（findFirst 函数中）或通过“for (i=count; i>=0; i--) ...”的循环去“从右往左”（findLast 函数中）寻找子串的出现位置。

2. findFirst 函数中“从左往右”寻找子串的含义是：首先把 subStr 子串的头位置 j0=0（j0 表示 subStr 字符数组的下标）与 sourceStr 的头位置 j1=0（j1 表示 sourceStr 字符数组的下标）“对齐”，而后逐字符地对两个串进行比较，若一直比到了子串的结束各字符均相

同时，则意味着在源串中找到了子串，此时返回子串在源串中的头字符位置（地址值）；若逐字符进行比较时发现有不相同，则要进一步将 subStr 子串的头位置 j0=0 与 sourceStr 的新头位置 j1=1“对齐”，而后又一次逐字符地对两个串进行比较（相当于子串整体“右平移”一个字符后再一次比较），如此等等。

3．findLast 函数中“从右往左”寻找子串的含义是：首先把 subStr 子串的头位置 j0=0（j0 表示 subStr 字符数组的下标）与 sourceStr 的头位置 j1=count（j1 表示 sourceStr 字符数组的下标）“对齐”，而后逐字符地对两个串进行比较，若一直比到了子串的结束各字符均相同时，则意味着在源串中找到了子串，此时返回子串在源串中的头字符位置（地址值）；若逐字符进行比较时发现有不相同，则要进一步将 subStr 子串的头位置 j0=0 与 sourceStr 的新头位置 j1=count-1“对齐”，而后又一次逐字符地对两个串进行比较（相当于子串整体“左平移”一个字符后再一次比较），如此等等。

4．主函数对两个函数各调用一次，而后输出源串 sourceStr 中从 subStr 子字符串头字符开始直到串尾的所有字符作为结果。程序“构架”如下：

```
char s0[81], s1[81], *p;
...          //输入源串 s0 以及子串 s1
p = findFirst(s0, s1);
if(p)
    ...      //输出结果
else
    cout<<"No match found -- from first!"<<endl;
p = findLast(s0, s1);
...
```

5．程序执行后的输入输出界面可以设计为：

```
Input a source_string:
ABC! 123 +-*! 1350K. 2005!
Input a sub_string:
! 1
Sub_str from first '! 1' =>! 123 +-*! 1350K. 2005!
Sub_str from last '! 1' =>! 1350K. 2005!
```

四． 自立题四

编程序，按如下要求来求解 n 元一次线性方程组（假设方程组具有唯一解）。

（1）方程个数 n 之值由用户通过键盘输入；

（2）方程组存放在“增广矩阵”A 之中，而 n 行 n+1 列的 A 存储空间通过 new 来动态分配，且 A 的各元素值也由用户通过键盘输入；

（3）方程组的解存放于“向量”B 之中，而具有 n 个元素的 B 存储空间也通过 new 来动态分配。

【实验目的】

1．通过 new 运算符生成一维和二维的动态数组，分别用于存放“增广矩阵”A 和方程组的解“向量”B。

2．编制相关函数，用于求解任意的 n 元一次线性方程组，并将结果显示在屏幕上。

【要点提示】

1．说明如下的全局变量，它们将能够在随后的每一个自定义函数中使用：

```
int n;                  //矩阵行数 n
double **A;             //二维数组首地址，“存放”增广矩阵（系数矩阵 + 方程组右端向量）
double *B;              //一维数组首地址，“存放”方程组的解
```

2．将整个求解任务进行“分解”，设计出如下各负其责的多个自定义函数：

```
void Linequ();          //通过 new 动态分配 A 存储空间以及 B 存储空间
void InMatrix();        //输入增广矩阵 A 的初值
void PrintL();          //显示线性方程组（增广矩阵 A 的数据）
void Solve();           //对增广矩阵 A 进行“同解变换”，将求出的解存放于 B 之中
void ShowX();           //输出方程组的解（B 向量数据）
void _Linequ();         //释放二维数组 A 以及一维数组 B 的存储空间
```

3．编制主函数，输入方程个数 n，而后按照上述给出的顺序对每一个自定义函数进行一次调用，从而完成所给方程组的求解任务。

4．上述的 Solve 函数，要对增广矩阵 A 进行“同解变换”，并最终将求出的解存放于 B 数组之中。所谓“同解变换”，主要使用如下的变换方法：“将某一行的各数据乘以适当的倍数加到另一行的对应各元素上去”，从而可首先将系数矩阵消为“上三角”，而后再进行所谓的“回代过程”，最后完成求解任务。

5．程序执行后的输入输出界面可以设计为：

```
Input positive n : 3
Input 12 numbers that will form the Line_equation:
2.2 1.5 -4.1 22.8 -2 1.7 12 33.9 11 -3 -5 8.2
The Line_equation is:
---------------------------------
    2.2     1.5    -4.1    22.8
     -2     1.7      12    33.9
     11      -3      -5     8.2
---------------------------------
The Result is:
---------------
X[0]=5.085
X[1]=12.8171
X[2]=1.85674
```

6．可进一步思考，在上述的求解方法基础上（消解时不选主元），而改进成为列主元消去法的求解方式。其消去过程概述如下：

```
for (k=1; k<n; k++) {
     ...
     /* 先在第 k-1 行到 n-1 行的范围内进行行互换，使 A[k-1][k-1]在第 k-1 列中为最大元，而后再进行消元过程。也即，先找列中最大元所在行 ind，而后将第 k-1 行与第 ind 行进行整行互换，之后再将系数矩阵消为“上三角”，并进行回代过程。
     */
}
```

五． 自立题五

编程序，按如下要求来求解任意阶数满秩矩阵的逆矩阵。

（1）矩阵行数（阶数）n 之值由用户通过键盘输入；

（2）将欲求逆的“原始矩阵”另加一个“单位矩阵”存放在于数组 A 之中，而 n 行 2n 列的 A 存储空间通过 new 来动态分配，且“原始矩阵”的各元素值也由用户通过键盘输入；

（3）利用行初等变换设法将 A 左半的“原始矩阵”化为“单位矩阵”，此时，右半的原“单位矩阵”则变成了欲求的结果逆矩阵。

【实验目的】

1.通过 new 运算符生成 n 行 2n 列的二维动态数组 A,并在 A 中通过初等变换来求取“原始矩阵”的逆矩阵。

2．编制相关函数，用于实现从输入、求解到输出结果等一系列的规划功能。

【要点提示】

1．说明如下的全局变量，它们将能够在随后的每一个自定义函数中使用：

```
int n;            //矩阵行数（阶数）n
double **A;       //二维数组首地址，“盛放”矩阵。初始时为“原始矩阵” +“单位矩阵”
```

2．将整个求解任务进行“分解”，设计出如下各负其责的多个自定义函数以完成各子任务。

```
void Matrix();            //通过 new 动态分配具有 n 行 2n 列的 A 存储空间（矩阵 A）
void InMatrix();          //输入“原始矩阵”数据，并形成矩阵 A“右半”的“单位矩阵”
void printSourceMatrix();    //按格式输出“原始矩阵”
void Solve();                //求解矩阵的逆（解处于 A 矩阵的“右半”）
void printResultMatrix();    //输出结果矩阵（A 矩阵“右半”的数据）
void _Matrix();              //释放动态空间
```

3．编制主函数，输入矩阵行数（阶数）n，而后按照上述给出的顺序对每一个自定义函数进行一次调用，从而完成所给的满秩矩阵的求逆任务。

4．上述的 Solve 函数中，可以使用如下具体做法来求取逆矩阵：利用行初等变换（如将某一行的各数据乘以适当的倍数加到另一行的对应各元素上去），设法将 A 左半的“原始矩阵”化为“单位矩阵”（首先将“左半”消为“上三角”；又将“左半”主对角线消为 1；最后将“左半”消为单位矩阵），此时，右半的原“单位矩阵”则变成了欲求的逆矩阵。

5．程序执行后的输入输出界面可以设计为：

```
Input positive n : 3
Input 9 numbers that will form the matrix:
1 1 1 2 1 0 3 -1 0
The source Matrix is:
       1        1        1
       2        1        0
       3       -1        0
The result Matrix is:
```

```
0       0.2       0.2
0       0.6      -0.4
1      -0.8       0.2
```

六. 自立题六

编程序，使用链表来实现如下问题：有 12 人围坐成一圈（沿顺时针方向依次编号为 1 到 12），按规则淘汰其中 11 人后（沿顺时针方向每当数到 k 时，那一人员就被“淘汰出局”，而其中的 k>1 由用户通过键盘输入指定），输出最后所剩那一个人的编号，并输出淘汰过程的“中间结果数据”。

注：第四章的自立题七通过使用数组方式来实现该相同问题。

【实验目的】

单向环形链表的生成、使用与相应的数据处理。

【要点提示】

1. 说明一个如下的结构体类型 itemType，而后靠使用 new 来动态生成这种类型的各结构体项，并“串接”形成一个单向环形链表（末项的后继为首项）来存放各数据（从 1 到 n 的人员编号）。

```
struct itemType {
    int num;            //人员编号
    itemType* next;     //通过 next“串接”后项
};
```

2. 通过如下样式的程序片断，来形成所需的单向环形链表（其 num 编号依次为 1 到 n）

```
itemType *first, *last, *tmp;
first=new itemType;
last=first;
first->num=1;               //first 指向链表首项（编号最小的那一项）
for(i=2; i<=n; i++) {       //依次形成编号从 2 到 n 的项，并“串接”成环形链表
    tmp = new itemType;
    tmp->num = i;
    tmp->next = first;
    last->next = ...
    last = ...
}
```

3. 编制“while(delCount<n-1) {…}”形式的循环，共循环 n-1 次，实现淘汰 n-1 个人的目标（每循环一次淘汰 1 人。要淘汰编号为 i 的人，只需将链表中含有该人员编号的项删除掉即可）。

4. 在“while(delCount<n-1) {…}”的循环体内，为计数（从 1 数到 k）确定出下一个被淘汰者，可通过多次使用类似于“tmp=tmp->next;”这样的语句，找到那一个要被删除的项，而后再通过指针值的修改来实现项的删除操作(注意链表是环状的，“tmp=tmp->next;”的操作总不会出问题)。

5．要输出淘汰过程的“中间结果数据”，只需将当时的环状链表各项的 num 分量值显示出来。

6．程序执行后的输入输出界面式样可设计为：

```
12 人围坐一圈，被数到 k(k>1)的人出局，k=? 8
init-Data:  =>   1  2  3  4  5  6  7  8  9 10 11 12
delNnm= 8;  =>   1  2  3  4  5  6  7  9 10 11 12
delNnm= 4;  =>   1  2  3  5  6  7  9 10 11 12
delNnm= 1;  =>   2  3  5  6  7  9 10 11 12
...
delNnm= 2;  =>   5  9
delNnm= 9;  =>   5
ansNum= 5
```

7．思考：若将人数 12 改为 10（或其他正整数）的话，则可以对 10（或其他指定个数）人的情况进行处理。

七． 自立题七

按下述方法设计一个排序程序：使用由结构形成的链表来存放数据；总保障，在把第 i 个数据插入到链表以前，链表中当前已有的 i-1 个数据已经是有序的（i=1，2，…，即是说，每次总是在原有有序链表的适当位置插入新数据，而保障插入后的链表仍有序。注意 i=1 时，链表为“空”显然满足条件）。

之后输入若干个数据，并将它们依次插入到上述链表中（并已排好了序），最后再将有序链表中的各数据作为排序结果逐一显示在屏幕上。

例如，程序执行后的输入输出界面可以设计为：

```
Input a positive number n (n>0):6
input 6 INTEGERs:
66 -2 35 100 0 -28
The sorted results are:
-28 -2 0 35 66 100
```

【实验目的】

链表的形成与使用：在已有链表中找到欲插入数据的适当位置，而后插入数据，并保障结果链表依然有序。

【要点提示】

1.可定义如下的结构类型 item 以及指针变量 first 与 last，而后从键盘输入 n 个数据，依次“插入”到以 first 指向其首、以 last 指向其末的有序链表之中，最终实现 n 个 int 型数据的从小到大排序，并输出排序结果。

```
struct item {                  //使用 item 结构来形成链表项
    int dat;                   //存放欲排序的数据
    item *next;                //指向其后项，“串联”成为链表
};
```

```
    item *first=NULL, *last=NULL;
```
由 first 指向链表首项，last 指向末项，赋予初值 NULL 意味着起始时链表为“空”。

2.“输入 n 个数据并将它们依次插入到有序链表的适当位置”的程序片断样式

```
for ( i=1; i<=n; i++ ) {                    //共处理 n 个数据
    cin>>number;                            //输入数据 number
    item *temp = new item;                  //为新数据生成一个链表表项
    temp->num = number;                     //新数据 number 放入 num 域
    temp->next = NULL;                      //next 域置为 NULL
    if (head == NULL) {                     //若当前链表为“空”
        ...                                 //temp 项既为 head 又为 tail
    }
    else
        if(temp->num <= head->num) {        //比首项小，将 temp 项插入链首
            ...
        }
        else
            if(temp->num >= tail->num){ //比末项大，将 temp 项插入链末
                ...
            }
            else {                          //将 temp 项插入链中的适当位置
                ...
            }
}
```

3. 对有序链表进行“遍历”，输出排序结果的程序片断样式

```
item *tmp=head;
while ( tmp!=NULL ) {
    ...
    tmp = tmp->next;
}
```

4. 思考：假设已经有两个都按升序排列的有序链表，现要将两者合并后形成一个新的有序链表（仍按升序排列，且保持原有的两个有序链表不被破坏）。则可考虑利用上述形成有序链表的方法以及“二路归并”的技术来实现。

八. 自立题八

利用由结构类型 chNode 形成的“链表”来存储字符串（串中的每一个字符占用一个链表项，字符本身存放在链表项的 c 分量中，而使用链表项的 next 分量去“串接”后继项，并总在串尾添加一个结束标志项，该项的 c 分量值为'\0'，而其 next 分量值为 NULL）。

```
struct chNode {
    char c;                 //存放一个字符
    chNode* next;           //“串接”后继项
};
```

并编制具有如下功能的函数，对此种不同链表中的字符串（如称为“链串”）进行处理。

（1） void putStr (char* a, chNode* p);

将 a 字符串（数组）中的所有字符“重置”到以 p 指向其首的“链串”中（替换掉原有的链串内容）。

（2） void catStr (char* a, chNode* p);

将 a 字符串（数组）中的所有字符“串接”到以 p 指向其首的原有“链串”字符的后面（尾部）。

（3） void displayStr (chNode* p);

屏幕输出以 p 指向其首的“链串”中的所有字符（一个字符串）。

（4） void getStr (char *b, chNode* p);

将以 p 指向其首的“链串”中的所有字符（一个字符串）取出放入字符数组 b 之中。

【实验目的】

链表的生成与使用：使用一个链表来存储一个不同的字符串，并自定义不同的函数，对此种不同链表中的字符串（如称为“链串”）进行某些设定的具体处理。

【要点提示】

1．使用上述的所谓“链串”方式，可以存放任意长度的字符串。但要对存储在这种结构中的字符串进行处理时，每次都要涉及到对链表的“遍历”过程（因为每次都只能从链首“走到”链尾才能找出相应字符串的各字符，方可进行某种指定的处理）。

2．可编制如下的示例性主函数，对上述各自定义函数进行调用以验证它们的正确性。

```
void main() {                       //主函数
    char a[30]="ABC321tdk+-*!$/&%", b[80];
    chNode *p = new chNode;         //生成一个由 p 指向其链首的空“链串”
    p->c = '\0';
    p->next = NULL;
    putStr("888", p);               //将“888”置入链串
    catStr(a, p);                   //将 a 串“串接”到以 p 指向其首的链串尾部
    displayStr(p);                  //输出以 p 指向其首的链串内容
    putStr("555", p);               //将“555”置入链串（重置，即替换）
    catStr("!!123!!", p);           //将“!!123!!”串接到以 p 指向其首的链串尾部
    displayStr(p);                  //输出以 p 指向其首的链串内容
    getStr(b, p);                   //将以 p 指向其首的链串内容取出放入字符数组 b 中
    cout<<b<<endl;                  //输出 b（字符串）
}
```

该主函数执行后的输出结果应该为：

```
888ABC321tdk+-*!$/&%
555!!123!!
555!!123!!
```

3．可进一步考虑添加其他自定义函数，用于实现另外关心的字符串处理功能。如，添加 lenStr 函数来获取“链串”的长度，添加 cmpStr 函数来对两个“链串”进行比较等。

九． 自立题九

自定义如下的结构类型 stuType：

```
struct stuType {
    char name[20];          //姓名
    char sex;               //性别，用字母 M 表示男士，用字母 F 表示女士
    int age;                //年龄
    double score;           //成绩
};
```

编制具有如下原型的 5 个自定义函数，实现各自的规定功能：

（1）void InputStuInfo(stuType* pStu)；

为 pStu 所指向的结构体输入有关数据（各分量），使之可返回到调用函数中（实现数据的“双向传递”）。

（2）void DisplayStuInfo(stuType* pStu)；

将 pStu 所指向的结构体的有关数据（各分量）显示在屏幕上。

（3）double AveAge(stuType* a, int n)；

求结构体数组 a 的前 n 个分量之 age 域的平均值并返回该值。

（4）stuType* MinScoreStu(stuType* a, int n)；

求结构体数组 a 之前 n 个分量中具有最小成绩（score）的首数组元素（为结构体），并返回指向该元素的指针。

（5）stuType* searching(stuType* a, int n, char* Name)；

在结构数组 a 的前 n 个元素中查找名字为 Name 的出现位置（指针值）并返回。若数组 a 中没有该名字的话，返回 NULL。

并编制主函数，首先进行如下说明：

```
const int n=3;
stuType a[n], *pMinScoStu;
```

而后对上述自定义函数进行调用：输入 a 数组的 n 个结构体数据；屏幕显示 a 数组的各分量数据；求出数组 a 各分量之 age 域的平均值 aveAge；求 a 中具有最小 score 域值的元素并显示；输入一个名字，而后在数组 a 中查找，并输出相关的查找结果信息（数组元素下标值 idx 以及此人的性别、年龄与成绩）。

例如，程序执行后的输入输出交互界面可以设计为：

```
-- Input a[0]=>a[2] --
name, sex, age, score:
ZhangLi f 22 92.5
name, sex, age, score:
LinJing f 23 87.5
name, sex, age, score:
GuoJian m 22 89
--- Display ---
ZhangLi  f  22  92.5
LinJing  f  23  87.5
GuoJian  m  22  89
--- AveAge ---
aveAge=22.3333
-- minScoreStudent --
LinJing  f  23  87.5
```

```
searching-name=? GuoJian
idx=2
about inf : m, 22, 89
```

【实验目的】

指向结构变量的指针及其使用；函数参数为指向结构的指针；函数的返回值为指向结构的指针；结构类型变量、结构数组等语法实体的使用。

【要点提示】

1. 自定义的结构类型 stuType 被说明为全局性的，这样可以在多个不同函数中使用该类型。

2. 函数 InputStuInfo 的参数被说明成指针型的，函数体中通过改变指针所指向的结构变量之内容，可以实现数据的“双向传递”，从而将被调函数中输入的数据传递到调用函数中实参指针所指向的结构变量中。例如，可以使用类似于如下的语句来实现对结构分量值的输入：

```
cin>>pStu->name>>pStu->sex>>pStu->age>>pStu->score;
```

3. 函数 DisplayStuInfo 的参数也为指向结构的指针，但其函数体中只是将结构体数据（各分量）显示在屏幕上，并不改变它们的值，也不需要再将它们“传回”到调用函数中去。

4. 函数 AveAge、MinScoreStu、searching 的第一个形参 a 均被说明为“stuType* a”，也即为指向结构的指针。实际上，这与将 a 说明为一维数组“stuType a[]”所起的作用是相同的，其实参都可以为数组，且实参数组带来的仅是在内存中连续存放的那一批数组元素的首地址（指针概念）。

另外注意，指针参数本身并无法指出数组的大小，所以这三个函数中都要靠随后的另一个参数（如 n）来指出要处理多少个数组分量。

5. 函数 MinScoreStu 与 searching 的返回值都为指向结构的指针，函数体中可以通过使用类似于“return a+idx;”形式的语句来实现所指定的功能（其中的 a 为结构数组名，是一个常量指针，idx 为某一所需的数组下标值）。

6. 主函数 main 中，假设通过如下形式的调用语句返回了欲查找的 searchName 的出现位置（指针值）pStu，那么，与该出现位置相关的数组元素下标值 idx 则可通过“pStu-a”来获得。

```
stuType *pStu = searching(a, n, searchName);
```

十. 自立题十

编制具有如下原型的自定义函数 fun，它的最后一个参数为指向函数的指针 pf：

```
void fun(double* point, double* value, int n, double(*pf)(double));
```

实现功能：计算出由 pf 所指向的那一函数在各 point 点（共 n 个点）处的函数值放入 value 数组中返回。

调用 fun 函数时，pf 的对应实参可以为某个具体的函数指针，用来指向某一自定义函数、或者指向某一标准库函数（注意，要求这些实参函数的原型必须与此处的形参 pf 相同，必须具有“double〈函数名〉(double);”这种函数形式）。

并编制出如下两个与形参 pf 具有相同函数原型的自定义函数 f1 与 f2，以便在调用 fun

函数时充当实参：

```
double f1 (double x){
    return x*x+x+1;
}
double f2 (double x){
    return 2*x-12;
}
```

而后编制主函数 main，通过调用 fun（依次使形参 pf 对应于自定义函数 f1、f2 及标准库函数 sqrt），从而可计算出这三个函数在一批自变量点 x 处的函数值（共 n 个，如 n=4），并将相应结果显示在屏幕上。例如，程序执行后的输出结果可以设计为：

```
x=0,  f1(0)=1
x=1.2,  f1(1.2)=3.64
x=25,  f1(25)=651
x=100.88,  f1(100.88)=10278.7
--------------------------------
x=0,  f2(0)=-12
x=1.2,  f2(1.2)=-9.6
x=25,  f2(25)=38
x=100.88,  f2(100.88)=189.76
--------------------------------
x=0,  sqrt(0)=0
x=1.2,  sqrt(1.2)=1.09545
x=25,  sqrt(25)=5
x=100.88,  sqrt(100.88)=10.0439
```

【实验目的】

说明并使用指向函数的指针；使用指向函数的指针作为函数的参数可使函数的通用性更强（到底要对哪一个函数进行处理，是由调用语句处的实参所决定的!）。

【要点提示】

1．自定义函数 fun 中，可通过循环依次求出 pf 所对应函数的各 value[i]。例如，可使用语句“double x = point[i]; value[i] = (*pf)(x);”，它等同于语句“value[i] = (*pf)(point[i]);”，也等同于 “*(value+i) = (*pf)(*(point+i));”，其中的“(*pf)”代表的正是调用 fun 函数时所带来的对应的具体实参函数。

2．主函数 main 中，可使用类似于如下的程序“构架”：

```
double p[4]={0, 1.2, 25, 100.88}, v[4];
double (*pf1)(double) = &f1;      //说明函数指针 pf1，并初始化使它指向函数 f1
fun(p, v, 4, pf1);                //由第四实参指定计算 f1 的一批函数值
...                               //输出各 v[i]
pf1 = &f2;                        //使函数指针 pf1 指向函数 f2
fun(p, v, 4, pf1);                //计算 f2 函数的一批函数值
...                               //输出各 v[i]
```

```
pf1 = &sqrt;                    //使函数指针 pf1 指向函数 sqrt
fun(p, v, 4, pf1);              //计算 sqrt 函数的一批函数值
...                             //输出各 v[i]
```

3．与数组名类似，函数名本身也为一个常量地址（函数的入口地址）。所以对 fun 函数进行调用时，其第四实参处，可以直接使用具体的函数名。例如：

```
fun(p, v, 4, f1);
fun(p, v, 4, sqrt);
```

也都是正确的函数调用语句。