作业10

- **Ex1.**设计程序完成下列功能：设有 **M** 个候选人，**N** 个选举人，每个选举人输入一个候选人姓名，最后输出各人得票数。

```
const int N = 3;
const int S = 20;
struct person
{
      char name[S];   //char name;?
      int count;      //double count;?

}leader[N] = {"Tom", 0, "Jerry", 0, "Mimi", 0};
```

```
const int M = 10;
int i;
char name[S];
for(i=1; i <= M; ++i)
{
     cin.getline(name, S);                    //输入选谁
     for(int j=0; j < N; ++j)
          if(strcmp(name, leader[j].name) == 0)
               leader[j].count++;
}
cout << endl;
for(i=0; i < N; ++i)
     cout << leader[i].name << leader[i].count;
```

- **Ex2.** 设计程序，对一个 N 个节点的单向链表中的一个 int 型数据成员求和（假设和非0），要求用递归函数实现求和功能：int SumR(Node *head);

```
const int N = 10;
struct Node
{
    int data;
    Node *next;
};

Node *InsCreate( );
void Output(const Node *);
int Sum(Node *);
void DeleteList(Node *);
```

```
int main()
{
    Node *list = InsCreate();
    Output(list);
    cout << Sum(list);
    DeleteList(list);

    return 0;
}
```

```
int Sum(Node *head)
{
    if(head == NULL)
            return 0;
    else
            return head->data + Sum(head->next);
}
```

```
Node *InsCreate( )
{
     Node *head = NULL;
     for(int i = 0; i < N; ++i)
     {
          Node *p = new Node;
          cin >> p -> data;
          p -> next = head;
          head = p;
     }
     return head;
}
```

```
void Output(const Node *head)
{      while(head != NULL)
      {            cout << head -> data << " ";
                   head = head->next;
      }
      cout << endl;
}

void DeleteList(Node *head)
{     while(head)
      {     Node *current = head;
            head = head -> next;
            free(current);
      }
}
```

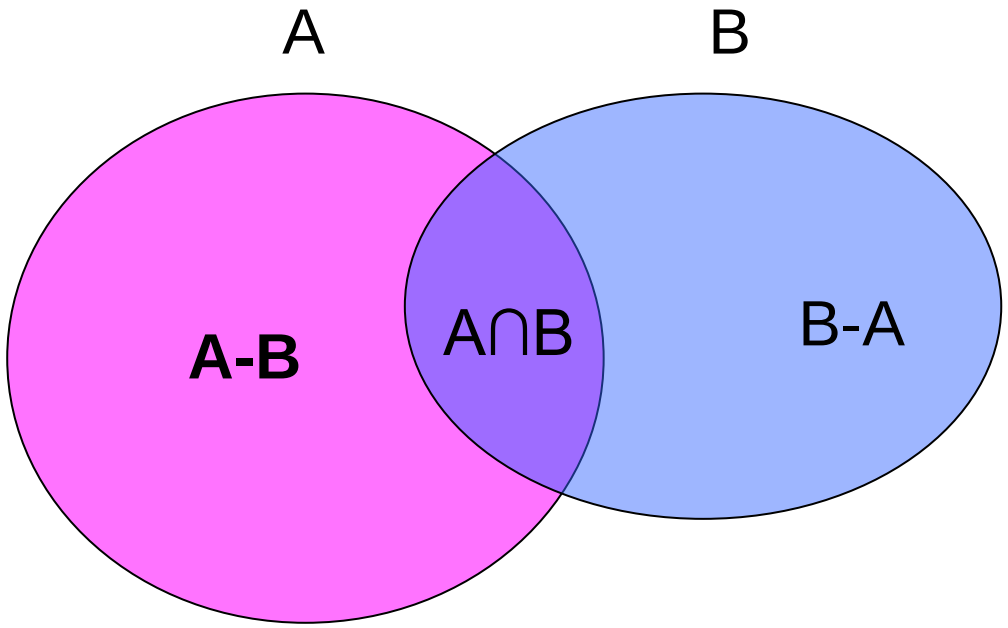
- **Ex3**. 设计程序，实现用链表存储输入的一个字符串，并用一个函数计算字符串的长度。

```
int Length(Node *head)
{
      int len = 0;
      if(head == NULL)
         return 0;
      else
      {
            while(head)
            {
                  ++len;
                  head = head -> next;
            }
            return len;
      }
}
```

```
#include <malloc.h>
struct Node
{
    char data;
    Node *next;
};
Node *InsCreate( );
int Length(Node *);
void DeleteList(Node *);
int main()
{
      Node *list = InsCreate();
      Output(list);
      cout << Length(list);
      DeleteList(list);
      return 0;
}
```

```
Node *InsCreate( )
{
        Node *head = NULL;
        char str[81];
        cin >> str;
        for(int i =0; str[i] != '\0'; ++i)
        {
                Node *p = new Node;
                p -> data = str[i];
                …
```

- **Ex4.** 设计程序，首先用链表建立两个整数**集合**（从键盘输入集合的元素，以-1结束，集合中的元素没有-1），然后计算这两个集合的交集、并集与一个差集，最后输出计算结果。




```
struct Node
{
      int content;
      Node *next;
};
```

```
Node *AppCreate( )
{
    Node *head = NULL, *tail = NULL;
    int a;
    cin >> a;
    while(a != -1)
    {
        Node *p = new Node;
        p -> content = a;
        p -> next = NULL;
        if(head == NULL)
            head = p;
        else
            tail -> next = p;
        tail = p;
        cin >> a;
    }
    return head;
```

可以修改为：

```
Node *o1()
{Node *head=NULL;
int a;cin>>a;
while(a!=-1)
{Node *p=new Node;
p->content=a;
if(head==NULL)
{head=p;
p->next=NULL;
}
else
{
Node *q=head;
while(q->next!=NULL)
q=q->next;
```

```
Node *Create()
{
    Node *head = NULL;
    int a;
    cin >> a;
    while(a != -1)
    {
        Node *p = new Node;
        p->content = a;
        if(head == NULL)
        {   head = p;
            p->next = NULL;
        }
        else
        {   Node *q = head;
```

缩进
空格
一行只写一句
标识符命名

交集

```
Node * Sintersection(Node *head1, Node *head2)
{
      Node *head = NULL;
      for(Node *p=head1; p != NULL; p=p->next)
            for(Node *q=head2; q!=NULL; q=q->next)
                  if(p->content == q->content)
                  {     Node *r = new Node;
                        r->content = p->content;
                        r->next = head;
                        head = r;
                  } //建链表，头部插入节点，A、B中都有的值
      return head;
}
```

差集 A-B

```
Node *Sdifference(Node *head1, Node *head2) // A-B
{     Node *head = NULL;
      for(Node *p=head1; p != NULL; p=p->next)
      {     bool flag = true;
            for(Node *q=head2; q != NULL; q=q->next)
                  if(p->content == q->content) //B中有该值
                        flag = false;
            if(flag == true)
            {     Node *r = new Node;
                  r->content = p->content;
                  r->next = head;
                  head = r;
            } //建链表，插入节点，B中没有、A中有的值
      }
      return head;
}
```

并集

```
Node *Sunion(Node *head1, Node *head2)
{
     Node *head = NULL;
     head = Sdifference (head1, head2); //先求A-B
     for(Node *q=head2; q != NULL; q=q->next)
     {     Node *r = new Node;
           r->content = q->content;
           r->next = head;
           head = r;
     } //头部插入B
     return head;
}
```

```
int main()
{
    Node *list1 = AppCreate();
    Node *list2 = AppCreate();
    Node *list_I = Sintersection(list1, list2);
    Output(list_I);
    Node *list_D = Sdifference(list1, list2);
    Output(list_D);
    Node *list_U = Sunion(list1, list2);
    Output(list_U);
    DeleteList(list1);
    DeleteList(list2);
    DeleteList(list_I); //如果不是新建的链表，不能重复删除
    DeleteList(list_D); //如果不是新建的链表，不能重复删除
    DeleteList(list_U); //如果不是新建的链表，不能重复删除
    return 0;
}
```

有的同学用模板？
#include <string>
#include <list>
不符合本课程要求。

第14周自主训练任务

1．调试课件例子程序，熟悉结构类型的构造、定义、初始化和操作方法，验证结构类型变量的赋值和值传递特性。

2．应用结构类型重新实现字符串压缩和解压任务，以便对含有数字字符的英文字符串正确解压（该方法利于扩展，但压缩效果略差，为节省空间，字符的个数可定义为**char**类型代替**int**类型）。

```
struct CompString
{     char ch;
      int count;                  //为节省空间，字符的个数可定义为char count;
}compStr[N] = { {0,0} }; //最后一个元素为{0,0}
```

3．实现链表的冒泡法排序函数：Node *ListBubbleSort(Node *head)；

```
void BubbleSort(Node *head)
{
        Node *cur;
        for(int i = 0; i < N-1; ++i)
                for(int j = 0; j < N-1-i; ++j)
                {
                        Node *pre = head;
                        for(int k = 0; k < j; ++k)
                                pre = pre->next;
                        cur = pre->next;
                        if(pre->data > cur->data)
                        {
                                int temp = pre->data;
                                pre->data = cur->data;
                                cur->data = temp;
                        }
                }
}
```

仅交换数据成员

```
void ListBubbleSort(Node *head)
{   bool flag = false;
    Node *previous = NULL;
    Node *current = NULL;
    while(!flag)
    {   flag = true;     //相邻两个元素全都已经满足顺序
        previous = head;
        current = head->next;
        while(current)
        {   if(previous->data > current->data)
            {   int temp = current->data;
                current->data = previous->data;
                previous->data = temp;
                flag = false;
            }
            previous = current;
            current = current->next;
        }
```

仅交换数据成员

```
void sort(Node *h)
{
    if(h == NULL || h->next == NULL)
        return;
    for(Node *p2 = h; p2 != NULL; p2 = p2->next)
    {
        Node *p = h;
        for(Node *p1 = p->next; p1 != NULL; p1 = p1->next)
        {
            if(p1->data < p->data)
            {
                int temp = p1->data;
                p1->data = p->data;
                p->data = temp;
            }
            p = p1;
        }
    }
}
```

做了部分无用功

仅交换数据成员

```
void ListBubbleSort(Node *head)
{
      int temp;
      for(Node *p1 = head; p1 ; p1 = p1->next)
            for(Node *p2 = p1->next; p2 ; p2 = p2->next)
                  if(p2->data < p1->data)
                  {
                        temp = p2->data;
                        p2->data = p1->data;
                        p1->data = temp;
                  }
}
```

不是相邻两个数据比较
不是标准的冒泡法

仅交换数据成员

解法三

```
Node *ListBubbleSort(Node *head)
{
    bool t;
    do {t = false;  //相邻两个元素全都已经满足顺序
        if(head->next->data < head->data)
        {   Node *temp = head->next;
            head->next = temp->next;
            temp->next = head;
            head = temp;
            t = true;
        }//先处理前两个节点
        Node *previous = head;
        while(previous->next->next != NULL)//剩下不止两个节点
        {   if(previous->next->next->data < previous->next->data)//从二三两个节点开始比较
            {   Swap(previous);
                t = true;
            }
            previous = previous->next;
        }
    } while(t);     return head;
}
```

```
void Swap(Node *previous)
{   Node *temp = previous->next->next;
    previous->next->next = temp->next;
    temp->next = previous->next;
    previous->next = temp;
    return;
}//交换previous后面的两个节点
```

解法三’

```
Node* list_bubbleSort(Node* head)
{   Node *end = NULL;
    while(end != head->next)
    {   Node *current = head, *follow = head->next, *pre;
        if(current->data > follow->data)
        {   current->next = follow->next;
            follow->next = current;
            head = follow;
        } //先处理前两个节点
        pre = head, current = pre->next, follow = current->next;
        while(follow != end)  //剩下不止两个节点
        {   if(current->data > follow->data)//从二三两个节点开始比较
            {   pre->next = follow;
                current->next = follow->next;
                follow->next = current;
            }// 交换 pre 后面的两个节点
            pre = pre->next;
            current = pre->next;
            follow = current->next;
        }
        end = current;
    }    return head;}
```

几种不按要求实现的做法

```
Node *ListBubbleSort(Node *head, int n);
```

```
for(int i = 1; i < n; ++i)
        head = ListBubbleSort(head); //小函数未实现排序
```

数组

课件插入法

4．设计并实现一个简单的机票管理系统。用单链表存储某航班已售机票信息（已售机票流水号、乘客姓名、机票价格、含日期的出售时间），系统业务功能包括售票、退票、按出售价格排序、客户查找等功能。

```
struct Ticket
{
      char name[20];          //乘客姓名
      double price;           //机票价格
      int count;              //序号
      …
      Ticket* next;           //指向下一节点的指针
};
```

```
Ticket* createList(int n)
{     Ticket *head = NULL, *tail = NULL;
      for(int i = 0; i < n; ++i)
      {     Ticket *p = new Ticket; //创建新节点
            cin >> p -> name; //给新节点的数据成员输入值
            cin >> p -> price;
            p -> count = i+1;
            p -> next = NULL; //给新节点的指针成员赋值
            if(head == NULL) //已有链表为空链表的情况
                  head = p;
            else //已有链表不空的情况
                  tail -> next = p;
            tail = p;
      }
      return head;
}
```

```
void PrintList(Ticket * head)
{
      ……
}

void deCrease(Ticket *h)          //退票善后处理
{
      Ticket *p = h;
      while(p)
      {
            p -> count = p -> count - 1;
            p = p -> next;
      }
}
```

```
void deleteTicket(Ticket *&head, char name[20])
{
      Ticket *previous = head; // previous 指向头节点
      if(!strcmp(previous -> name, name)) //删除头节点
      {
            Ticket *current = head; // current 指向头节点
            head = head->next; // head 指向新的头节点
            delete current; //释放删除节点的空间
            deCrease(head);
            return ;
      }
      ……
}
```

```
void deleteTicket(Ticket *&head, char name[20])
{
      ……    //最早买票的那个人退票，见上页
      while( previous -> next != NULL
                        && strcmp(previous->next->name, name) )
            previous = previous -> next;  //查找
      if(previous -> next == NULL)//没找到
            cout << "此人未购票" << endl;
      else //删除中间节点
      {
            Ticket *current = previous -> next;
            previous -> next = current -> next;
            delete current;
            deCrease(current-> next);
      }
}
```

```
void sortList(Ticket *&head)
{     ……
}

void DeleteList(Ticket * head)
{     ……
}

int main()
{     int n;
      cin >> n;
      Ticket *head = createList(n);
      PrintList(head);
      char name[20];
      cin >> name;
      deleteTicket(head, name);PrintList(head);
      sortList(head);PrintList(head);     DeleteList(head);
      return 0;
}
```

```
Node *p,*q;
for(p = head; p->next != 0; p = p->next)
{    for(q = p->next; q != 0; q = q->next)
     {        if(q->val > p->val)
              {    int t = q->val;
                   q->val = p->val;
                   p->val = t;
              }
     }
}
return head;        //选择法排序
```

5．判断一个单向链表中是否有环（即最后一个节点的next指针是否指向了链表中的某个节点），对无环的非空单向链表建立一个环（从链表最后一个节点指向第M个节点，其中，M可以是1、2、3、...），操作成功，返回true，否则（链表空），返回false。函数原型分别为：bool HasLoop(Node *head);及bool CreateLoop(Node *head, int m);

节点类型Node定义如下：

```
struct Node
{
    int  data;//节点的值
    Node* next;
    //链表中下一个节点的指针，无环时最后一个节点为0
}
```

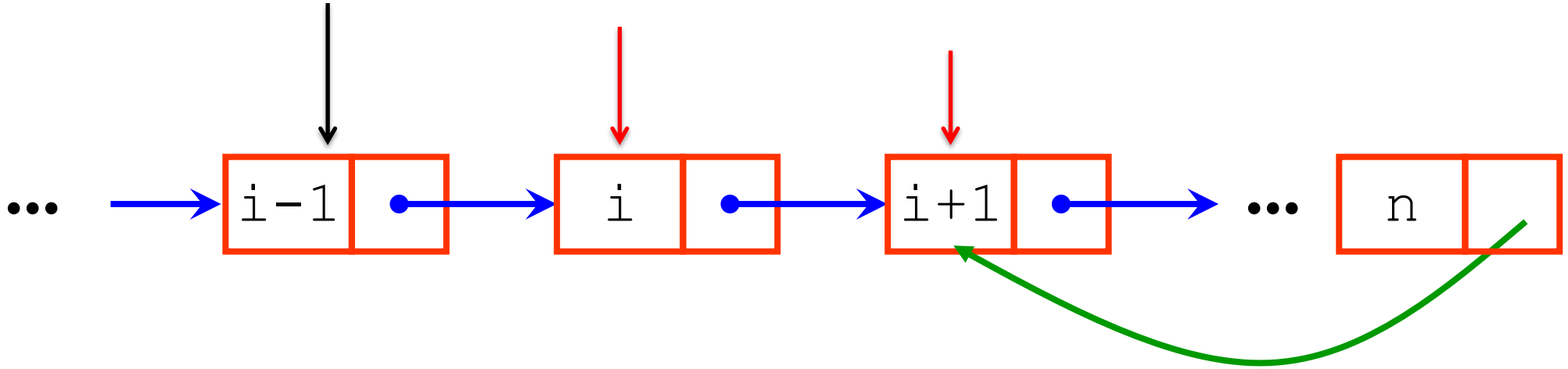
```
const int N = 10;
Node *InsCreate( );  (略)
void Output(const Node *);(略)
bool hasLoop(Node *);
Node *Get(Node *, int);   //定位某个节点
Node *Tail(Node *);       //定位最后一个节点
```

```
int main()
{
    Node *list = InsCreate();
    Output(list);
    if(!hasLoop(list))
        cout<<"At first, the list does not have a loop.\n";

    Node *fifth = Get(list, 4); //定位第4个节点
    Node *tail  = Tail(list); //定位最后一个节点
    tail->next = fifth;          // 创建环

    if(hasLoop(list))
        cout<<": Then, we create a loop.\n";

        return 0;
}
```

```
Node *Get(Node *head, int m)
{
    while(head != NULL && m > 1)
    {
        head = head -> next;
        --m;
    }
    return head;
}//定位第m个节点
```

```
Node *Tail(Node *head)
{
    //if(head == NULL)
    //{
    //    return NULL;
    //}
    while(head -> next != NULL)
    {
        head = head->next;
    }
    return head;
}//定位最后一个节点
```

技巧



技巧

```
bool hasLoop(Node *head)
{
    Node *slowNode, *fastNode1, *fastNode2;
    slowNode = fastNode1 = fastNode2 = head;
    while(slowNode && (fastNode1 = fastNode2->next)
          && (fastNode2 = fastNode1->next))
    {
        if(slowNode == fastNode1 || slowNode == fastNode2)
        {
            cout << slowNode->data;
            return true;
        }
        slowNode = slowNode->next;
    }
    return false;
}
```

6. 实现findFirstCross函数：查找两个无环单向链表首个重合节点的位置，若无重合返回NULL。查找函数原型和链表的节点类型为“const Node *findFirstCross(const Node *headA, const Node *headB);”。

```
Node *Create() //创建链表
{
    Node *head=NULL, *tail=NULL;
    int x;
    cin >> x;
    while(x != -1)
    {
        Node *p = new Node;
        p->data = x;
        p->next = NULL;
        if(head == NULL)
            head=p, tail = p;
        else
            tail->next = p, tail=p;
       cin >> x;
    }
    return head;
}
```

```
// 找到两个链表相交的第一个交点
const Node* findFirstCross(const Node* headA, const Node* headB)
{     const Node *pA = headA;
      const Node *pB = headB;
      计算两个链表的长度（略）
      int offset;     //    计算长度差
      在较长的链表上先走offset步（略）
      之后pA、pB一起走
      直到任一个为NULL(不相交)
      或直到二者相等(即为第一个交点)
      while(pA && pB)
      {     if(pA==pB)
                  return pA;
            pA = pA->next;
            pB = pB->next;
      }
      return NULL;
}
```

```
int main()
{    Node *headA = Create();
     Node *headB = Create();
     Node *headC = Create();
     Node *p = headA;
     while(p->next)
           p = p->next;     //在A的尾部
     p->next =  headC;      //接上C
     p = headB;
     while(p->next)
           p = p->next;     //在B的尾部
     p->next =  headC;      //接上C
     const Node *pos = findFirstCross(headA, headB);
     if (pos)                       //判断是否空
         printf("yes: %d\n", pos->data);
     else
         printf("no\n");
      return 0;
}
```

让两个链表具有部分重合节点

7．应用栈结构重新实现圆括号是否配对的检查任务。

```
int main()
{
      char str[81];
      gets(str);
      if( Matchcheck(str) )
                  cout << "配对";
      else
                  cout << "不配对";
      return 0;
}
```

```
bool Matchcheck(char *str)
{     Stack st;
      init(st);
      int ch;
      for(; *str != '\0'; ++str)
      {     if(*str == '(')              //若是左括号则入栈
                  push(st, *str);
            else if(*str == ')')         //若是右括号则弹出左括号
                  if( ! pop(st, ch) )    //没有足够的左括号
                        return false;
      }
      if(st.top == -1)
            return true;
      else                               //循环结束，栈非空，则左括号多余
            return false;
}
```

```
const int N = 100;
struct Stack
{    int top;
     int buffer[N];
};
```

```
void init(Stack &s)
{    s.top = -1;
}     //初始化栈
```

```
bool pop(Stack &s, int &i)
{
     if(s.top == -1)
      return false;
     else
     {i = s.buffer[s.top];
      s.top--;
      return true;
     }
}     //出栈
```

```
bool push(Stack &s, int i)
{
     if(s.top == N-1)
      return false;
     else
     {s.top++;
      s.buffer[s.top] = i;
      return true;
     }
}     //入栈
```

8．假设网络节点A和网络节点B之间的通信协议涉及四种格式的报文内容，通信时，先传送报文内容的格式种类，再传送相应格式的报文内容，每次只能发送一种格式的报文内容，四种报文内容的数据类型是结构类型StructType1~ StructType4，请用统一的数据类型描述整个报文（含格式种类和报文内容）。

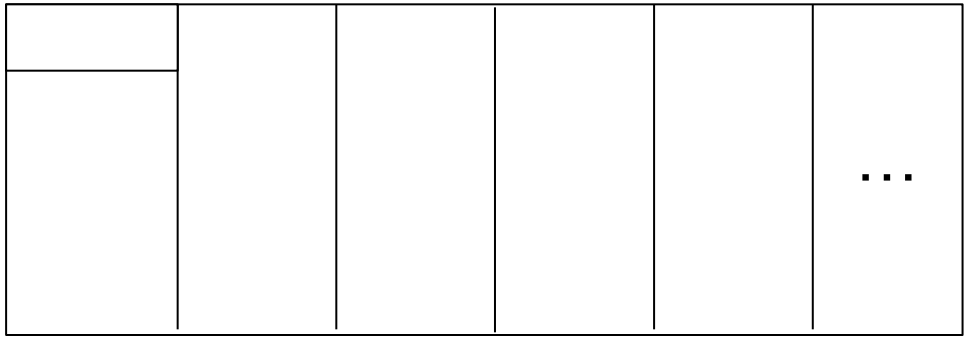
```
//报文内容
typedef union
{
      StructType1 pkt1;
      StructType2 pkt2;
      StructType3 pkt3;
      StructType4 pkt4;
}PacketContent;
```

```
//整个报文
typedef struct
{
      char pktType; //格式种类
      PacketContent pktContent;
}Packet;
```

9． 输入一组图形数据，然后输出相应的图形。其中的图形可以是：线段、矩形和圆。（提示：一组图形数据可以用一个数组来表示和存储，数组中每个元素可以是结构类型，包括表示是何种图形的枚举类型成员，以及存储图形数据的联合类型成员，联合类型自身可以包括线段、矩形和圆三种结构类型成员，线段结构类型的成员对应端点坐标，矩形结构类型的成员对应左上角和右下角端点坐标，圆结构类型的成员对应半径和圆点坐标，每种图形的具体数据也用结构类型表示，是为了可以灵活设置其中成员的类型，例如，圆的半径可以是整数，也可以是小数。）

```
struct Line
{
     double x1, y1, x2, y2;
} ;
struct Rectangle
{
     double left, top, right, bottom;
};
struct Circle
{
     double x, y, r;
};
```

```
const int N = 100;                    //图形的个数

union Figure
{ Line lin;
  Rectangle rect;
  Circle circ;
};

enum Shape { LIN, RECT, CIRC}; //表明figs[i]是何种图形

struct TaggedFigure
{ Shape shp;
  Figure fig;
};
```



```
void input(TaggedFigure figs[ ], int size)
{ int i, s;
  for(i = 0; i < size; ++i)
  {   cout << "输入0、1或2，代表LIN、RECT、CIRC \n";
      cin >> s;
      switch(s)
      {     case 0: ……
            case 1: figs[i].shp = RECT;
                    cout << "依次输入矩形左上和右下顶点的横纵坐标：");
                    cin >> figs[i].fig.rect.left
                        >> figs[i]. fig.rect.top
                        >> figs[i]. fig.rect.right
                        >> figs[i]. fig.rect.bottom;
                    break;
            case 2:   figs[i].shp = CIRC;
```

……

```
int main( )
{
  TaggedFigure figs[N];
  input(figs, N);
  for(int i = 0; i < N; ++i)
   draw(figs[i]);
  return 0;
}
```

```
void input(TaggedFigure figs[ ], int size)
{ int i, s;
  for(i = 0; i < size; ++i)
  {   cout << "输入0、1或2，代表LIN、RECT、CIRC \n";
      cin >> s;
      switch(s)
      {     case 0: ……
            case 1: figs[i].shp = RECT;
                    cout << "依次输入矩形左上和右下顶点的横纵坐标：");
                    cin >> figs[i].fig.rect.left
                        >> figs[i]. fig.rect.top
                        >> figs[i]. fig.rect.right
                        >> figs[i]. fig.rect.bottom;
                    break;
            case 2:   figs[i].shp = CIRC;
```

……

```
void input(TaggedFigure figs[ ], int size)
{ int i, s;
  for(i = 0; i < size; ++i)
  {   cout << "输入0、1或2，代表LIN、RECT、CIRC \n";
      cin >> s;
      switch(s)
      {     case 0: ……
            case 1: ……
            case 2:    figs[i].shp = CIRC;
                       cout << "请依次输入圆心的横纵坐标和半径：";
                       cin >> figs[i].fig.circ.x
                           >> figs[i]. fig.circ.y
                           >> figs[i]. fig.circ.r;
                       break;
      }}}
```

```
void draw(TaggedFigure f)
{
  switch(f.shp)
     //通过成员shp的值就可知道结构变量f存储的是什么图形
  {
     case LIN: draw_line(f. fig. lin);          break;
     case RECT:draw_rectangle(f. fig. rect); break;
     case CIRC:draw_circle(f. fig. circ);     break;
  }
}
```

Thanks !