# 中级软件设计师下午试题模拟62

## 试题一

1、 【说明】

所谓货郎担问题，是指给定一个无向图，并已知各边的权，在这样的图中，要找一个闭合回路，使回路经过图中的每一个点，而且回路各边的权之和最小。

应用贪婪法求解该问题。程序先计算由各点构成的所有边的长度(作为边的权值)，按长度大小对各边进行排序后，按贪婪准则从排序后的各边中选择边组成回路的边，贪婪准则使得边的选择按各边长度从小到大选择。

函数中使用的预定义符号如下：

```
#define M 100
typedef struct{/*x为两端点p1、p2之间的距离，p1、p2所组成边的长度*/
    float x;
    int p1, p2;
}tdr;
typedef struct{/*p1、p2为和端点相联系的两个端点，n为端点的度*/
    int n, P1, p2;
}tr;
typedef struct{/*给出两点坐标*/
    float x,y;
}tpd;
typedef int tl[M];
int n=10;
```

【函数】

```
float distance(tpd a,tpd b);/*计算端点a、b之间的距离*/
    void sortArr(tdr a[M], int m);
    /*将已经计算好的距离关系表按距离大小从小到大排序形成排序表，m为边的条数*/
    int isCircuit(tr[M], int i, int j);
    /*判断边(i, j)选入端点关系表r[M]后，是否形成回路，若形成回路返回0*/
    void selected(tr r[M], int i, int j);/*边(i,j)选入端点关系表r*/
    void course(tr r[M], tl 1[M]);/*从端点关系表r中得出回路轨迹表*/
    void exchange(tdr a[M], int m, int b);
    /*调整表排序表，b表示是否可调，即是否有边长度相同的边存在*/
    void travling(tpd pd[M], int n, float dist, t1 locus[M])
    /*dist记录总路程*/
    {
        tdr dr[M];/*距离关系表*/
        tr r[M];;/*端点关系表*/
        int i, j, k, h, m;/*h表示选入端点关系表中的边数*/
        int b;/*标识是否有长度相等的边*/
        k=0;
        /*计算距离关系表中各边的长度*/
        for(i=1;i<n;i++){
            for(j=i+1;j<=n;j++){
                k++;
                dr[k].x= (1) ;
                dr[k].p1=i;
                dr[k].p2=j;
            }
        }
    m=k;
```

```
    sortArr(dr,m);/*按距离大小从小到大排序形成排序表*/
    do{
        b=1;
        dist=0;
        k=h=0;
        do{
            k++;
            i=dr[k].p1;
            j=dr[k].p2;
            if((r[i].n<=1)&&(r[j].n<=1)){/*度数不能大于2*/
                if( (2) ){
                    /*若边(i，j)加入r后形成回路，则不能加入*/
                    (3) ;
                    h++;
                    dist+=dr[k].x;
                }else if( (4) ){
                    /*最后一边选入r成回路，则该边必须加入且得到解*/
                    selected(r,i,j);
                    h++;
                    dist+=dr[k].x;
                }
            }
        }while((k!=n)&&(h!=n));
        if(h==n){/*最后一边选入构成回路，完成输出结果*/
            course(r,locus);
        }else{/*找不到解，调整dr，交换表中边长相同的边在表中的顺序，并将b置0*/
            (5) ;
        }
    }while(!b);
}
```

## 试题二

2、 [说明]

下面的流程图(如图所示)用N－S盒图形式描述了数组A中的元素被划分的过程。其划分方法是：以数组中的第一个元素作为基准数，将小于基准数的元素向低下标端移动，而大于基准数的元素向高下标端移动。当划分结束时，基准数定位于A[i]，并且数组中下标小于i的元素的值均小于基准数，下标大于i的元素的值均大于基准数。设数组A的下界为 low，上界为high，数组中的元素互不相同。例如，对数组(4，2，8，3，6)，以4为基准数的划分过程如下：

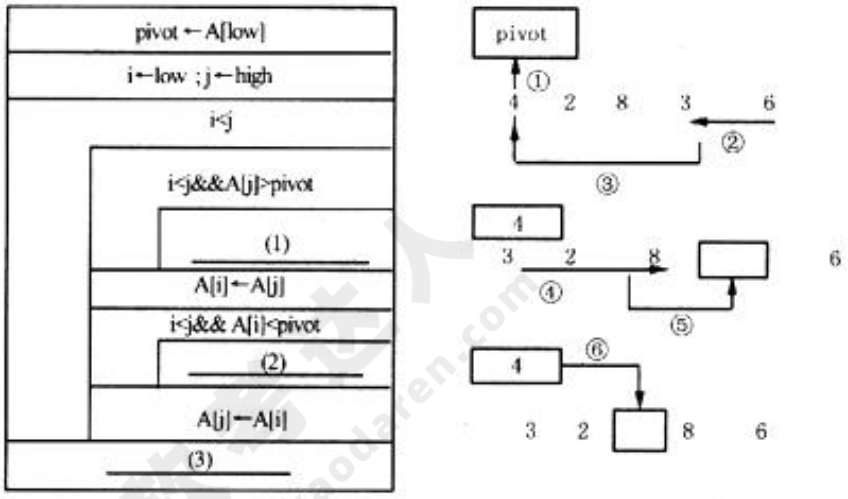



图 3 流程图

[流程图]

[算法说明]

将上述划分的思想进一步用于被划分出的数组的两部分，就可以对整个数组实现递增排序。设函数int p(int A[], int low, int hieh)实现了上述流程图的划分过程并返回基准数在数组A中的下标。递归函数void sort(int A[], int L, int H)的功能是实现数组A中元素的递增排序。

[算法]

```
void sort(int A[], int L, int H)  {
if (L<H)    {
k=p(A, L, R);      //p()返回基准数在数组A中的下标
 sort(__(4)__);      //小于基准敷的元素排序
sort(__(5)__);      //大于基准数的元素排序
}
}
```

## 试题三

3、 【说明】

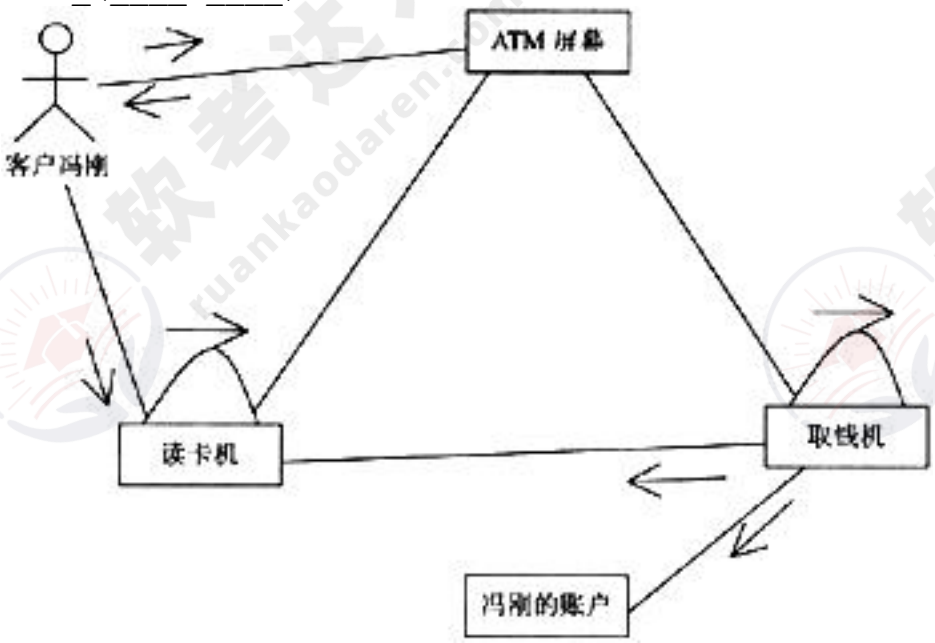
银行客户需要从ATM取100元，他向ATM的读卡机插卡，读卡机读取他的卡号，然后ATM屏幕初始化，ATM提示输入密码，客户输入密码(123456)，ATM打开他的账户，密码有效，因此ATM提示选择事务，客户选择取钱，ATM提示输入金额，客户输入100元，ATM验证账户上有足够的钱，就从账上减去100元，ATM吐出100元，并退出的卡。

【问题】

根据上面的描述，在下面填写，完成未完成的协作图。

1. 插卡(客户一读卡机)
2. _(_____→_____)
3. _(_____→_____)
4. 提示输入PIN (123456) (ATM 显示屏→客户)
5. _(_____→_____)
6. _(_____→_____)
7. 验证PIN(__→__)
8. 提示选择事务(__→__)
9. _(客户→ATM屏幕)
10. 提示金额(ATM屏幕→客户)

11. 输入金额(客户→ATM屏幕)
12. 取钱(ATM屏幕→的账户)
13. _(____→____)
14. _(____→____)
15. _(____→____)
16. 提供收据(客户的账户→取钱机)
17. _(____→____)




## 试题四

【说明】

快速排序是一种典型的分治算法。采用快速排序对数组A[p..r]排序的3个步骤如下。

1. 分解：选择一个枢轴(pivot)元素划分数组。将数组A[p..r]划分为两个子数组（可能为空)A[p..q-1]和A[q+1..r]，使得A[q]大于等于A[p..q-1)中的每个元素，小于 A[q+1..r]中的每个元素。q的值在划分过程中计算。

2. 递归求解：通过递归的调用快速排序，对子数组A[p..q-1]和A[q+1..r]分别排序。

3. 合并：快速排序在原地排序，故不需合并操作。

4、【问题1】

下面是快速排序的伪代码，请填补其中的空缺；伪代码中的主要变量说明如下。

A：待排序数组

p，r：数组元素下标，从p到r

q：划分的位置

x：枢轴元素

i：整型变量，用于描述数组下标。下标小于或等于i的元素的值小于或等于枢轴元素的值

j：循环控制变量，表示数组元素下标

```
QUICKSORT (A, p, r){
    if  (p <r){
        q=PARTITION(A, p, r) ;
        QUICKSORT(A, p, q-1);
        QUICKSORT(A, q+1, r);
    }
}
PARTITION(A, p, r){
```

```
        x=A[r]; i=p-1;
        for(j=p; j≤r-1; j++){
            if  (A[j]≤x){
                i=i+1;
                交换A[i]和A[j]
        }
        }
```

交__(1)__和__(2)__//注：空(1)和空(2)答案可互换，但两空全部答对方可得分

```
return __(3)__
    }
```

5、【问题2】

(1)假设要排序包含n个元素的数组，请给出在各种不同的划分情况下，快速排序的时间复杂度，用O记号。最佳情况为__(4)__，平均情况为__(5)__，最坏情况为__(6)__。

(2)假设要排序的n个元素都具有相同值时，快速排序的运行时间复杂度属于哪种情况？__(7)__。(最佳，平均、最坏)

6、【问题3】

(1)待排序数组是否能被较均匀地划分对快速排序的性能有重要影响，因此枢轴元素的选取非常重要。有人提出从待排序的数组元素中随机地取出一个元素作为枢轴元素，下面是随机化快速排序划分的伪代码—利用原有的快速排序的划分操作，请填充其中的空缺处。其中，RANDOM(i，j)表示随机取i到j之间的一个数，包括i和j。

```
RANDOMIZED- PARTITION(A，p，r){
   i=RANDOM(p，rl);
```

交换__(8)__和__(9)__；//注：空(8)和空(9)答案可互换，但两空全部答对方可得分

```
return PARTITION (A，p，r);
}
```

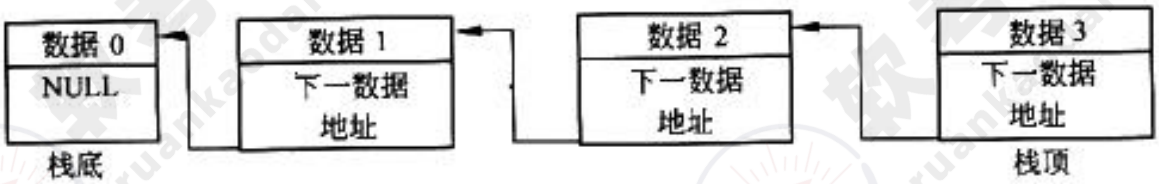
(2)随机化快速排序是否能够消除最坏情况的发生？__(10)__。(是或否)

## 试题五

7、【说明】

栈(Stack)结构是计算机语言实现中的一种重要数据结构。对于任意栈，进行插入和删除操作的一端称为栈顶(Stock Top)，而另一端称为栈底(Stock Bottom)。栈的基本操作包括：创建栈(NewStack)、判断栈是否为空(IsEmpty)、判断栈是否已满(IsFull)、获取栈顶数据(Top)、压栈/入栈(Push)、弹栈/出栈(Pop)。

当设计栈的存储结构时，可以采取多种方式。其中，采用链式存储结构实现的栈中各数据项不必连续存储(如下图所示)。




以下C代码采用链式存储结构实现一个整数栈操作。

【C代码】

```
typedef struct List {
int data;                    //栈数据
struct List* next;           //上次入栈的数据地址
}List;
typedef struct Stack{
List* pTop;                  //当前栈顶指针
```

```
}Stack;
 Stack* NewStack()  {return (Stack*) calloc(1/sizeof(Stack)); }
int IsEmpty(Stack* S){//判断栈S是否为空栈
    if( (1) )return 1;
    return 0;
}
int Top(Stack* s){//获取栈顶数据。若栈为空，则返回机器可表示的最小整数
if(IsEmpty(S))return INT_ MIN;
    return (2) ;
}
void Push(Stack* S, int theData) {//将数据theData压栈
    List* newNode;
    newNode=(List*)calloc(1/sizeof (List));
    newNode->data=theData;
    newNode->next=S->pTop;
    S->pTop= (3) ;
}
void Pop(Stack* S)  {//弹栈
    List* lastTop;
    if(IsEmpty(S)  )  return;
    lastTop=S->pTop;
    S->pTop= (4) ;
    free(lastTop);
}
#define MD(a)  a<<2
int main(){
    int i;
    Stack* myStack;
    myStack= NewStack();
    Push(myStack, MD(1));
    Push(myStack, MD(2));
    Pop(myStack);
    Push(myStack, MD(3)+1);
    while( !IsEmpty(myStack)  ){
         printf("%d", Top(myStack));
        Pop(myStack);
    }
    return 0;
}
```

以上程序运行时的输出结果为： (5)

答案：

**试题一**

1、(1) distance(pd[i],pd[j])
(2) !isCircuit(r,i,j)
(3) selected(r,i,j)
(4) h==n-1
(5) exchange(dr,m,b)

[解析] 本题主要是函数调用的问题。

空(1)是计算各边的长度，根据函数的声明及说明，应填distance(pd[i]，pd[j])。

由注释可见空(2)是判断边(i，j)加入r后是否形成回路，若形成了回路，不加入。由语句"dist+=dr[k].x;"可知此处是将边加入，故此处应该是不形成回路条件。参照isCircuit函数声明及说明可知，若形成回路返回0，故空(2)填"!isCircuit(r,i,j)"。

空(3)是将边(i,j)加入到r中，参照selected函数声明及说明，可得空(3)填selected(r,i,j)。

由注释可见空(4)是最后一条边条件，变量h表示的是"选入端点关系表中的边数"，而 n各节点回路应该包含n条边，这点也可从后面h==n输出解看出，故空(4)填h==n-1。

空(5)是进行调整，调用exchange函数，正确调用形式为exchange(dr,m,b)。

## 试题二

2、(1)j-- (2)i++ (3)A[i]←pivot或[j]←pivot (4) A，L，k-1或A，L，k
(5)A，k+1，H或A，k，H

[解析] 题目考查快速排序算法。快速排序采用了一种分治的策略，通常称为分治法。其基本思想是：将原问题分解为若干个规模更小，但结构与原问题相似的子问题。递归地解这些子问题，然后将这些子问题的解组合为原问题的解。

快速排序的具体过程为：第一步，在待排序的n个记录中任取一个记录，以该记录的排序码为基准，将所有记录分成2组，第一组各记录的排序码都小于等于该排序码，第二组各记录的排序码都大于该排序码，并把该记录排在这2组中间，这个过程称为一次划分。第二步，采用同样的方法，对划分出来的2组元素分别进行快速排序，直到所有记录都排到相应的位置为止。

在进行一次划分时，若选定以第一个元素为基准，就可将第一个元素备份在变量pivot中，如图中的第①步所示。这样基准元素在数组中占据的位置就空闲出来了，因此下一步就从后向前扫描。如图中的第②步所示，找到一个比基准元素小的元素时为止，将其前移，如图中的第③步所示。然后再从前向后扫描，如图中的第④步所示，找到一个比基准元素大的元素时为止，将其后移，如图中的第⑤步所示。这样，从后向前扫描和从前向后扫描交替进行，直到扫描到同一个位置为止，如图中的第⑥步所示。

由题目中给出的流程图可知，以第一个元素作为基准数，并将A[low]备份至pivot，i用于从前向后扫描的位置指示器，其初值为low，j用于从后往前扫描的位置指示器，其初值为high。当i 1)从后向前扫描数组A，在ipivot，就继续向前扫描(j--)；如果被扫描的元素A[i] 2)这时，再从前向后扫描，在ipivot就停止扫描，并将此元素的值赋给目前空闲着的A[j]。

3)这时又接第(1)步，直到i>j时退出循环。退出循环时，将pivot赋给当前的A[i](A[i]←pivot)。

递归函数的目的是执行一系列调用，直到到达某一点时递归终止。为了保证递归函数正常执行，应该遵守下面的规则：

1)每当一个递归函数被调用时，程序首先应该检查基本的条件是否满足，例如，某个参数的值等于0，如果是这种情形，函数应停止递归。

2)每次当函数被递归调用时，传递给函数一个或多个参数，应该以某种方式变得"更简单"，即这些参数应该逐渐靠近上述基本条件。例如，一个正整数在每次递归调用时会逐渐变小，以至最终其值到达0。

本题中，递归函数sort(int A[], int L, int H)有3个参数，分别表示数组A及其下界和上界。根据流程图可知，这里的L相当于流程图中的i，这里的H相当于流程图中的j。因为P()返回基准数所在数组A中的下标，也就是流程图中最后的"A[i]←pivot"中的i。根据快速排序算法，在第一趟排序后找出了基准数所在数组A中的下标，然后以该基准数为界(基准数在数组中的下标为k)，把数组A分成2组，分别是A[L,…,k-1)和A[k+1,…,H)，最后对这2组中的元素再使用同样的方法进

行快速排序。

## 试题三

3、1．插卡(客户→读卡机)
2．读卡号(读卡机→读卡机)
3．屏幕初始化(读卡机→ATM屏幕)
4．提示输入PIN(ATM显示屏→客户)
5．输入PIN(123456)(客户→ATM屏幕)
6．打开账户(ATM屏幕→客户的账户)
7．验证PIN(ATM屏幕→客户的账户)
8．提示选择事务(ATM屏幕→客户)
9．选择事务(取钱)(客户→ATM屏幕)
10．提示金额(ATM屏幕→客户)
11．输入金额(100元)(客户→ATM屏幕)
12．取钱(100元)(ATM屏幕→客户的账户)
13．验钱(100元)(客户的账户→客户的账户)
14．扣钱(100元)(客户的账户→客户的账户)
15．提供钱(100元)(客户的账户→取钱机)
16．提供收据(客户的账户→取钱机)
17．退卡(客户的账户→读卡机)

[解析] 这道题和模拟试题4中的试题3是相似的，一个需求描述的时序图和协作图是可以相互转换的，所以，这个取钱过程的时序图的分析方法同样可以用在协作图的分析上。

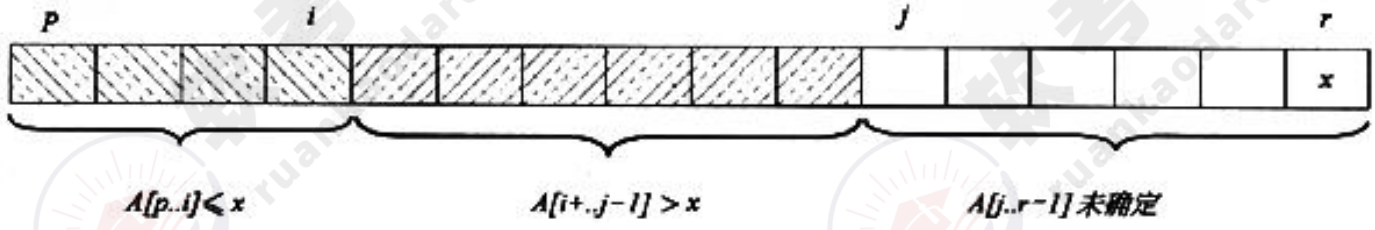
根据上述的分析方法并结合题中已经给出的提示可以得出答案，答案如下。

## 试题四

4、(1)A[i+1]　(2)A[r]　(3)i+1
注：空(1)和空(2)答案可以互换
5、(4)O(nlgn)或$O(\log_2 n)$　(5)O(nlgn)或$O(n\log_2 n)$
(6)$O(n^2)$　(7)最坏
6、(8)A[i]　(9)A[r]　(10)否
注；空(8)和空(9)答案可以互换
试题四[分析]

本题考查算法的设计与分析技术。

问题1考查快速排序算法的伪代码，快速排序最核心的处理是进行划分，即 PARTITION操作，根据枢轴元素的值，把一个较大的数组分成两个较小的子数组，一个子数组的所有元素的值小于等于枢轴元素的值，一个子数组的所有元素的值大于枢轴元素的值，而子数组内的元素不排序。划分时，以最后一个元素为枢轴元素，从左到右依次访问数组的每一个元素，判断其与枢轴元素的大小关系，并进行元素的交换，如图所示：




PARTITION 操作

在问题1给出的伪代码中，当循环结束后，A[p..i]中的值应小于等于枢轴元素值x，而A[i+1..r-1]中的值应大于枢轴元素值x。此时A[i+1)是第一个比A[r]大的元素，因此 A闭与A[i+1]交换，得到划分后的两个子数组。PARTITION操作返回枢轴元素的位置，因此返回值为i+1。

问题2考查的是快速排序算法的时间复杂度分析。当每次能作均匀划分时，算法为最佳情况，此时时间复杂度可以通过计算递归式T(n)=2T(n/2)+O(n)，得到时间复杂度为O(nlgn)：当每次为

极端不均匀划分时，即长度为n的数组划分后一个子数组为n-1，一个为0，算法为最坏情况，此时时间复杂度可以通过计算递归式T(n)=T(n-1)+O(n)，得到时间复杂度为$O(n^2)$；平均情况的分析较为复杂，我们可以假设数组每次划分为 9/10：1/10，此时时间复杂度可以通过计算递归式T(n)=T(9/10)+T(1/10)+O(n)，得到时间复杂度为O(nlgn)，因此在平均情况下快速排序仍然有较好的性能，时间复杂度为 O(nlgn)。当所有的n个元素具有相同的值时，可以认为数组已经有序，此时每次都划分为长度为n-1和0的两个子数组，属于最坏情况。

问题3中，由于随机化的快速排序的划分调用了传统的快速排序算法的PARTITION操作，而传统的划分每次以数组的最后一个元素作为枢轴元素，因此，随机化的划分操作中每次先随机获得一个元素，将其与最后一个元素交换。随机化的快速排序消除了输入数据的不同排列对算法性能的影响，降低了极端不均匀划分的概率，但不能保证不会导致最坏情况的发生。

## 试题五

7、(1)S==NULL||S->pTop==NULL　　(2)S->pTop->data　(3)newNode
　　(4)S->pTop->next，或lastTop->next　　(5)244

[解析]

本题考查基本程序设计能力。

堆栈是软件设计中常使用的一种经典数据结构，题目给出的操作都是任何堆栈都具有的基本操作。堆栈的存储结构通常采用数组或链表形式，但无论采用哪种存储结构，整体上呈现的是后进先出的特点，即后进入堆栈的元素先出栈。题目中给出的结构体 Stack仅包含一个指向栈顶元素的指针(栈顶指针)，当且仅当堆栈中没有元素时，该指针应为NULL。当向堆栈中增加元素时，首先需要动态创建该元素的存储区，并且栈顶指针指向该元素。当元素出栈时，栈顶指针则指向出栈元素的紧前一个元素。结构体List表示栈中元素，包含对应的数据和指向紧上次入栈的元素指针next，对于第1个入栈的元素，指针next为NULL,而其他元素中的指针next一定不为NULL。

C语言中，如果用一个整数型表达式表示条件判定语句的话，该表达式的值为。则表示假，非0表示真。从给定程序代码可以看出，对于函数IsEmpty，若其返回值为0则表示堆栈非空，否则表示堆栈为空。因此，对于空(1)，必须填写可表示堆栈为空的判定语句：S=NULL||S->p)Top==NULL，这2个条件中只要有1个条件满足，则表明堆栈S为空。对于空(2)，此时需要返回栈顶元素中的数据，而栈顶元素为S->pTop，所以对应的数据应该为S->pTop->data。

对于压栈操作Push，在为新元素获取存储空间后，必须调整堆栈的栈顶指针S->pTop指向新元素的存储区，即S->pTop=newNode。对于弹栈操作Pop，弹出栈顶元素lastTop后，需要调整栈顶指针，使其指向被弹出元素的下一个元素，即S->pTop=S->pTop->next，或S->pTop=lastTop->next。

对于main函数中宏MD(x)，在程序预编译时会按字符替换为"x<<2"。所以在main函数中，首先入栈的元素为"1<<2"，即整数4，第2个入栈的元素为"2<<2"，即整数8，其次将8弹出，然后再将"3<<2+1"入栈，C语言中"+"优先级高于"<<"，所以此时入栈者为整数24，而此时堆栈中有2个元素，其中栈顶元素为24，下一元素为4。最后，若堆栈非空，则循环完成显示栈顶元素的值、弹出栈顶元素的操作，直至堆栈为空。所以程序执行时的输出内容为"244"。