A. 选择题

考点：链表、栈、队列、字符串。

1. 所谓“先进先出（FIFO）”、“先进后出（FILO）”是对于数据结构中的一个特定元素而言，其进入容器和从容器中弹出的顺序。
   “先进先出”是指先进入容器的元素优先被弹出，可以类比排队的过程，越先排队的人就越早排到。这与队列的性质相契合。
   “先进后出”是指先进入容器的元素最后被弹出，可以类比叠放积木的过程，最先放置的积木处于最底部，想要将其取出必须先将其上方所有叠放的积木取出。这与栈的性质相契合。
2. 如果需要将单链表中的第k个结点删去（假设不需要释放内存），只需要将第$k-1$个结点的后继指针指向第$k+1$个结点（即第$k-1$个结点的后继结点的后继结点）。用代码可以表示为 `p->next = p->next->next` 。

首先对这段代码简要分析。

```c
#include <stdio.h>

int main() {
    int n;
    scanf("%d", &n); // 读入一个整数n
    int arr[n];
    for (int i = 0; i < n; ++i) {
        scanf("%d", &arr[i]); // 读入n个整数
    }
    int stack[n], idx = 0; // 创建一个大小为n的栈
    for (int i = 0; i < n; i++) { // 遍历读入的n个整数
        while (idx > 0 && stack[idx - 1] > arr[i]) { // 当栈不为空且栈顶元素大于该整数时
            idx--; // 栈顶元素出栈
        }
        stack[idx] = arr[i]; // 将该整数入栈
        idx++;
    }
    while (idx > 0) {
        printf("%d ", stack[idx - 1]); // 将栈中所有元素出栈并输出
        idx--;
    }
    return 0;
}
```

3. 主要对中间for循环遍历n个整数的部分进行分析。注意到其中嵌套的$while$循环始终执行出栈操作，而由于每个元素一定进栈一次，所以也至多出栈一次（也就是说每个元素被出栈后一定不会再进栈），因此$while$循环执行的次数一定不超过n次。整体时间复杂度为$O(n)$。
4. 模拟程序运行。
   遍历到3，此时栈空，跳过$while$循环，将3入栈。
   遍历到2，此时栈顶元素3大于2，将3出栈。此时栈空，跳出$while$循环，将2入栈。
   遍历到1，此时栈顶元素2大于1，将2出栈。此时栈空，跳出$while$循环，将1入栈。

遍历到4，此时栈顶元素2小于4，跳过$while$循环，将4入栈。
遍历到8，此时栈顶元素4小于8，跳过$while$循环，将8入栈。
遍历到7，此时栈顶元素8大于7，将8出栈。此时栈顶元素4小于7，跳出$while$循环，将7入栈。
遍历到6，此时栈顶元素7大于6，将7出栈。此时栈顶元素4小于6，跳出$while$循环，将6入栈。
遍历到2，此时栈顶元素6大于2，将6出栈。此时栈顶元素4仍然大于2，将2出栈。此时栈顶元素1小于2，跳出$while$循环，将2入栈。
遍历到9，此时栈顶元素2小于9，跳过$while$循环，将9入栈。
遍历到5，此时栈顶元素9大于5，将9出栈。此时栈顶元素2小于5，跳出$while$循环，将5入栈。
遍历结束，此时栈中元素从栈顶至栈底依次为$5, 2, 1$，将其依次出栈并输出，因此输出结果为 `5 2 1` 。
更一般化地，我们发现遍历到a_i的时候，会将栈顶所有大于a_i的元素弹出，然后将a_i入栈。因此栈中的元素（从栈底到栈顶）一定以a_i结尾且是**单调不减**的，具有这种特性的栈叫做**单调栈**。利用其单调性质，我们可以在线性时间复杂度内解决一些较为复杂的算法问题（如寻找一个序列中每个元素左侧第一个比当前元素大的元素）。

5. 根据字典序的定义比较即可。
   记A、B、C、D四个选项的字符串分别为S_A, S_B, S_C, S_D。由于S_A为S_B, S_C, S_D的前缀，根据定义可知$S_A < S_B, S_A < S_C, S_A < S_D$。
   S_B和S_C, S_D前五个字符相同，从第六个字符开始出现差异，由于$S_{B6} < S_{C6} = S_{D6}$，根据定义可知$S_B < S_C, S_B < S_D$。
   S_C和S_D前六个字符相同，从第七个字符开始出现差异，由于$S_{C7} > S_{D7}$，根据定义可知$S_C > S_D$。
   因此，字典序最大的字符串为S_C。

B. 汇编验证

考点：模拟，栈。

签到题。

题外话，本题实际上是一个高度简化的Java虚拟机，本题中唯一要求实现的指令$call$实际上是Java虚拟机中的$invokestatic$指令。

正如题干所说的，你需要维护一个栈，记录虚拟机栈上各个元素的类型，遇到$call$指令时，将元素类型出栈，并与函数的参数列表中的类型逆序匹配，如果匹配成功，则将函数的返回值类型压栈，否则回答 `NO` 。最后如果声明的返回值类型不是0，确认返回时栈上只有一个元素，且类型与返回值类型匹配，否则回答 `NO` ，如果返回值类型是0，则确认栈为空，否则回答 `NO` 。都检查通过则回答 `YES` 。

标准程序（Python）：

```python
class Func:
    def __init__(self, rval, args):
        self.rval = rval
        self.args = list(reversed(args)) # Turn the push-order into pop-order

class Stack:
    def __init__(self, init):
        self.items = init

    def call(self, func):
        for j in func.args:
            if len(self.items) == 0 or self.items.pop() != j:
                return False
        if func.rval != 0:
            self.items.append(func.rval)
        return True

n = int(input())
funcs = [None]
for i in range(n):
    rval, _, *args = map(int, input().split())
    funcs.append(Func(rval, args))
s, r, _, *p = map(int, input().split())
stack = Stack(p)
for i in range(s):
    j = int(input())
    if not stack.call(funcs[j]):
        print("NO")
        exit(0)
if (r == 0 and len(stack.items) == 0) or (len(stack.items) == 1 and stack.items[0] == r):
    print("YES")
else:
    print("NO")
```

标准程序（C）：

```c
#include <stdio.h>

#define MAXN 10005
int stack[MAXN], top;
int rett[MAXN], argc[MAXN], argt[MAXN][50];

int main() {
    int n;
    scanf("%d", &n);
    for (int i = 1; i <= n; ++i) {
        scanf("%d %d", &rett[i], &argc[i]);
        for (int j = 0; j < argc[i]; ++j) {
            scanf("%d", &argt[i][j]);
        }
    }
    int s, r, c;
    scanf("%d %d %d", &s, &r, &c);
    for (int i = 0; i < c; ++i) {
        int t;
        scanf("%d", &t);
        stack[top++] = t;
    }
    for (int t = 0; t < s; ++t) {
        int i;
        scanf("%d", &i);
        for (int j = argc[i] - 1; j >= 0; --j) {
            if (top == 0 || argt[i][j] != stack[top - 1]) {
                printf("NO\n");
                return 0;
            }
            --top;
        }
        if (rett[i]) stack[top++] = rett[i];
    }
    if ((r == 0 && top == 0) || (r && top == 1 && stack[top - 1] == r)) {
        printf("YES\n");
    } else {
        printf("NO\n");
    }
    return 0;
}
```

评分标准：

1. 通过所有样例：100分

C. 打牌

考点：队列

本题要求实现一个至多q个元素的队列，并且在达到q个元素时，将最后一个元素出队，并维护队列中元素中每种牌的数量加一的积。也就是实现$push(n)$和$product()$两个操作。

考虑一个元素入队时，积如何变化。显然，如果队列中已经有k张相同的牌，积将变为$\frac{k+1}{k}$倍。我们开一个数组统计每种牌有多少个，每次入队时，将积乘上$\frac{k+1}{k}$，然后将该牌的数量加一。出队时，将积乘上$\frac{k-1}{k}$，然后将该牌的数量减一。

队列本身可以使用$q+1$个元素的数组实现，使用两个指针分别指向队列的头和尾，每次入队时，将尾指针向后移动一位，出队时，将头指针向后移动一位。当指针指向数组的末尾时，将其置为0。队列满时，头指针和尾指针相邻，队列空时，头指针和尾指针相等。每次入队时，如果队是满的，就先出队一个，如题目所说。

标准程序（Python）：

```
class Queue:
    def __init__(self, sz):
        self.items = [None] * (sz + 1)
        self.head = 0
        self.tail = 0
        self.stat = [1] * 6
        self.prod = 1

    def push(self, n):
        if (self.head + 1) % len(self.items) == self.tail:
            self._pop() # Maintain fixed size queue
        self.items[self.head] = n
        self.head = (self.head + 1) % len(self.items)
        self.prod = self.prod // self.stat[n] * (self.stat[n] + 1)
        self.stat[n] += 1

    def _pop(self):
        n = self.items[self.tail]
        self.prod = self.prod // self.stat[n] * (self.stat[n] - 1)
        self.stat[n] -= 1
        self.tail = (self.tail + 1) % len(self.items)

d = 0
n, q = map(int, input().split())
qu = Queue(q)
for _ in range(n):
    z = int(input())
    x = (d + z) % 6
    qu.push(x)
    d = qu.prod
    print(d)
```

标准程序（C）：

```c
#include <stdio.h>

int queue[100004], l, r;
int cnt[6];

int main() {
    int n, q;
    long long prev_d = 0;
    scanf("%d %d", &n, &q);
    for (int i = 0; i < n; ++i) {
        int x;
        scanf("%d", &x);
        x = (x + prev_d) % 6;
        queue[r++] = x;
        ++cnt[x];
        if (i >= q) {
            --cnt[queue[l++]];
        }
        long long d = 1;
        for (int i = 0; i < 6; ++i) {
            d *= cnt[i] + 1;
        }
        printf("%lld\n", d);
        prev_d = d;
    }
    return 0;
}
```

评分标准：

1. print打印给出的样例输出：5分
   ~~2. 每次暴力遍历队列：60分~~
2. 通过所有样例：100分

D. 傅立叶-勒让德展开

考点：逼近，分部积分，勒让德多项式，列表

先来考虑怎么算勒让德多项式的问题，勒让德多项式的递推式为

$$P_0(x) = 1$$
$$P_1(x) = x$$
$$P_n(x) = \frac{2n-1}{n} x P_{n-1}(x) - \frac{n-1}{n} P_{n-2}(x)$$

我们用一个浮点数的列表来保存勒让德多项式，列表的第i个（从0开始编号）元素表示x^i的系数，于是三项递推式可以表达为

$$P_n[i] = \frac{2n-1}{n} P_{n-1}[i-1] - \frac{n-1}{n} P_{n-2}[i]$$

n次勒让德多项式，我们数组开$n+1$就可以了。用代码实现没啥难度，就是写的多了点。

然后我们知道系数

$$a_k = \frac{(f, P_k)}{(P_k, P_k)} = \frac{2k+1}{2} \int_{-1}^{1} P_k(x) \cos(ax+b) dx$$

积分的话，很容易想到，把多项式的每一项拿出来和$\cos(ax+b)$积分，然后再加起来就行了。问题转化为求一系列$\int_{-1}^{1} x^i \cos(ax+b) dx$的值。考虑分部积分，我们有

$$\int_{-1}^{1} x^i \cos(ax+b) dx$$
$$= \frac{1}{a} \int_{-1}^{1} x^i d \sin(ax+b)$$
$$= \frac{1}{a} \left[x^i \sin(ax+b) \right]_{-1}^{1} - \frac{1}{a} \int_{-1}^{1} \sin(ax+b) dx^i$$
$$= \frac{1}{a} \left[sin(a+b) - (-1)^i sin(-a+b) \right] - \frac{i}{a} \int_{-1}^{1} x^{i-1} \sin(ax+b) dx$$
$$= \frac{1}{a} \left[sin(a+b) + (-1)^i sin(a-b) \right] + \frac{i}{a^2} \int_{-1}^{1} x^{i-1} d \cos(ax+b)$$
$$= \frac{1}{a} \left[sin(a+b) + (-1)^i sin(a-b) \right] + \frac{i}{a^2} \left[x^{i-1} \cos(ax+b) \right]_{-1}^{1} - \frac{i}{a^2} \int_{-1}^{1} \cos(ax+b) dx^{i-1}$$
$$= \frac{1}{a} \left[sin(a+b) + (-1)^i sin(a-b) \right] + \frac{i}{a^2} \left[cos(a+b) + (-1)^i cos(a-b) \right] - \frac{i(i-1)}{a^2} \int_{-1}^{1} x^{i-2} \cos(ax+b) dx$$

我们把$\frac{1}{a} \left[sin(a+b) + (-1)^i sin(a-b) \right] + \frac{i}{a^2} \left[cos(a+b) + (-1)^i cos(a-b) \right]$写进程序里加到积分结果上，然后把$\frac{i(i-1)}{a^2}$加到低次积分的系数上，然后从高次向低次计算，就可以求出这一堆积分的和了。需要注意的是，i为0或1时，就不需要继续操作其他项的系数了，因为$\frac{i(i-1)}{a^2}$肯定为0，并且$i-2$数组越界就不好玩了。

最后把积分结果代回去，就可以得到系数了。

你也可以直接进行数值积分，但本题没有这个必要。

标准程序（Python）：

```python
from math import sin, cos

def legendre(n):
    ppp, pp, p = [1], [0, 1], None
    if n == 0:
        return ppp
    elif n == 1:
        return pp
    for i in range(2, n + 1):
        p = [0] * (len(pp) + 1)
        for j in range(len(pp)):
            p[j + 1] += (2 * i - 1) / i * pp[j]
        for j  in range(len(ppp)):
            p[j] -= (i - 1) / i * ppp[j]
        ppp, pp = pp, p
    return p

def fit(a, b, order):
    if a == 0:
        return cos(b) if order == 1 else 0
    p = legendre(order)
    r1, r2 = 0, 0
    ans = 0
    for i in range(len(p) - 1, -1, -1):
        f = r1 + p[i]
        n = (-1) ** i
        ans += f * (\
            1 / a * (sin(a + b) + n * sin(a - b)) + \
            i / a ** 2 * (cos(a + b) + n * cos(a - b))\
        )
        r1, r2 = r2, -f * i * (i - 1) / a ** 2
    return ans * (2 * order + 1) / 2

print(fit(*map(int, input().split(" "))))
```

标准程序（C）

```c
#include <stdio.h>
#include <math.h>

double P[55][55];
int n, a, b;

double f(double x) {
    double ans = 0, tx = 1;
    for (int i = 0; i <= n; ++i) {
        ans += P[n][i] * tx;
        tx *= x;
    }
    ans *= cos(a * x + b);
    return ans;
}

double simpson(double l, double r) {
    double mid = l + (r - l) / 2;
    return (r - l) * (f(l) + 4 * f(mid) + f(r)) / 6;
}

double integral(double l, double r, int n) {
    double res = 0, h = (r - l) / n;
    for (int k = 0; k < n; ++k) {
        res += simpson(l + k * h, l + (k + 1) * h);
    }
    return res;
}

int main() {
    scanf("%d %d %d", &a, &b, &n);
    P[0][0] = 1;
    P[1][1] = 1;
    for (int i = 2; i <= n; ++i) {
        for (int j = 0; j <= i; ++j) {
            if (j) P[i][j] = P[i - 1][j - 1] * (2 * i - 1) / i;
            P[i][j] -= P[i - 2][j] * (i - 1) / i;
        }
    }
    printf("%lf\n", integral(-1, 1, 5000) * (2 * n + 1) / 2);
    return 0;
}
```

评分标准：

1. print打印给出的样例输出：5分
2. 只能求解$n=0$的情况：20分
3. 只能算出课本上给出的前5项勒让德多项式的系数: 40分
4. 通过所有样例：100分

E. 谁活到最后

考点：递归

本题是课本Chapter 2.4 Excercise 67的变形，如果你做了作业，这题应该有一个大致的思路。

这题一看典中典约瑟夫问题。递归三大经典题：汉诺塔问题，直线分割平面，约瑟夫问题。

这题就是约瑟夫问题的一个经典变形，应该能在网上找到很多解法。至于为什么叫这个名字，是模仿数据结构老师的命名。

什么是递归？

递归（归纳性定义）和数学归纳法是高度一致的，它们都由**基础步**、**归纳步/递归步**两部分组成。区别在于，数学归纳法希望证明一个结论，而递归希望定义某种东西（比如函数等）。

> 递归举例：
> 基础步：$0! = 1$
> 递归步：$n! = n(n-1)!$，其中$n \geq 1$

好，我们现在来看这题怎么解，首先，我们令$J(x)$表示x个人约瑟夫环，最后活下来的人的编号。为了方便，我们把$J(x)$向下移动一个单位，即所有人从0开始编号，记这个函数为$J_0(x)$。那么我们有$J(x) = J_0(x) + 1$。这种平移的目的是为了$0 \leq J_0(x) < x$，与取模运算结果的范围一致。

我们需要希望递归定义出J_0。为了确定基础步需要几步，我们先来看递归步。

显然，当$x \geq 2$时，我们需要杀死一个人，这个人肯定不是存活的，存活的人将通过剩下的人约瑟夫环选出，只不过，剩下的$x-1$个人每个人的编号不是0到$x-2$，所以我们没办法直接复用$J(x-1)$，而是需要把剩下的人的编号从0到$x-2$映射回x个人时每个人的编号，除了编号以外这个约瑟夫过程和原来的一样。假设这个从$x-1$个人时每个人的编号映射回x个人时每个人的编号的函数叫$R(m,x)$，我们有：

$$J_0(x) = R(J_0(x-1), x)$$

这个$R(m,x)$函数并不好想，我们把它的对应关系列出来，以试图找到一个解析式。为了方便，我们将被杀死的人前面的人弄到后面去，以打括号标记原来的位置。因为编号是从0开始的，所以被杀死的人是$n-1$号，我们从他的下一位开始标记，即0号。我们有：

$R(m,x)$	(0)	(1)	(...)	$n-1$	n	$n+1$	$n+2$	...	$x-1$	0	1	...	$n-2$
m				killed	0	1	2	...	$x-n-1$	$x-n$	$x-n+1$	...	$x-2$

这里有些不太严谨，因为如果$x \leq n$，上述n应该是$n \mod x$，但是不影响结果，为了空间写得下，我们这张表中的n都是$n \mod x$。

很明显可以找到一个符合条件的函数解析式，即

$$R(m,x) = (m+n) \mod x$$

因此我们得到了J_0的递归步：

$$J_0(x) = (J_0(x-1) + n) \mod x$$

注意到$J_0(x)$只用到了$J_0(x-1)$，因此基础步只需要一步，即一个人约瑟夫环时，肯定这唯一一个人存活，他标号为0，即

$$J_0(1) = 0$$

因此我们得到了J_0的递归定义：

$$\begin{gathered} J_0(1) = 0 \\ J_0(x) = (J_0(x-1) + n) \mod x \end{gathered}$$

至此，你已经可以编写程序求解

```
def josephus0(x, n):
    return 0 if x == 1 else (josephus0(x - 1, n) + n) % x
def josephus(x, n):
    return josephus0(x, n) + 1
```

但是，这里有个问题，它只能通过x比较小的测试样例，它的时间复杂度是$O(x)$，而题目给出的数据范围是大x、小n，所以我们需要转向对x友好而不是对n友好的算法。

注意到，因为n特别小，而x又特别大，这就导致了99%的模x操作都在做无用功。我们可以考虑把这些无用功去掉，把$J_0(x-1)$展开。假设从x到$x+s$这一段都没有有意义的模x操作，我们有：

$$J_0(x+s) = J_0(x) + s \cdot n$$

再假设我们只要向前走一步就会遇到有意义的模x操作，即

$$J_0(x+s+1) = (J_0(x) + (s+1) \cdot n) \mod (x+s+1)$$

式子感觉比较丑，令$s_1 = s+1$，我们有

$$J_0(x+s_1) = (J_0(x) + s_1 \cdot n) \mod (x+s_1)$$

我们求解这个s_1，注意到$a \mod b \neq a$等价于$a \geq b$，我们有

$$J_0(x) + n \cdot s_1 \geq x + s_1$$

注意$J_0(x) < x$恒成立，当$n > 1$时

$$s_1 \geq \frac{x - J_0(x)}{n-1}$$

当$n = 1$时，无解，但此时$J_0(x) = x - 1$，这是平凡的，到时候写程序来个特判就行了。下面的讨论都是在$n > 1$的情况下进行的。

上界：

$$\begin{gathered} J_0(x) + n \cdot s < x + s \\ s < \frac{x - J_0(x)}{n-1} \\ s_1 < \frac{x - J_0(x)}{n-1} + 1 \end{gathered}$$

结论:

$$s_1 = \left\lceil \frac{x - J_0(x)}{n-1} \right\rceil$$

唯一与课本不同的是，课本中$n=2$，这意味着$\frac{x-J_0(x)}{n-1}$始终是整数，无需向上取整，从而我们在后面步骤中可以推出一个好看的通项公式，但其实对编程来说时间复杂度不会有任何变化。

通过这个式子，我们可以“快进”到$J_0(x+s_1)$，省去中间的递归步骤。

$$s_1 = \left\lceil \frac{x_0 - J_0(x_0)}{n-1} \right\rceil$$
$$J_0(x_0+s_1) = (J_0(x_0) + s_1 \cdot n) \mod (x_0+s_1)$$

如果要求的x满足$x_0 \le x < x_0 + s_1$，我们有

$$J_0(x) = J_0(x_0) + (x - x_0) \cdot n$$

否则，令$x_0 \leftarrow x_0 + s_1$，继续递推求解。

当然，你可以直接打表找到上述规律，然后再证明规律是正确的，这样的思维强度小很多。

标准程序（Python）：

```python
def josephus(x, n):
    if n == 1:
        return x
    currx, curry = 1, 0
    while True:
        step = (currx - curry + n - 2) // (n - 1)
        nextx = currx + step
        nexty = (curry + n * step) % nextx
        if x < nextx:
            return curry + n * (x - currx) + 1
        currx, curry = nextx, nexty

print(josephus(*map(int, input().split(" "))))
```

标准程序（C）：

```c
#include <stdio.h>

long long solve(long long x, int n) {
    if (n == 1) return x;
    int s = 0;
    long long i = 1;
    while (1) {
        long long k = (i - s + n - 2) / (n - 1);
        if (x < i + k) return s + n * (x - i) + 1;
        i += k;
        s = (s + n * k) % i;
    }
    return 0;
}

int main() {
    long long x;
    int n;
    scanf("%lld %d", &x, &n);
    printf("%lld\n", solve(x, n));
    return 0;
}
```

评分标准：

1. 写对基础步，但是递归步写错：20分
2. 使用数组暴力模拟：40分
3. 使用链表暴力模拟：60分
4. 写对基础步和递归步，但是没化简：80分
5. 通过所有样例：100分

拓展： 如果要求倒数第m个被杀死的，应该怎么做？（提示：基础步）