A - 程设基础知识选择题 1

答案为 CADDD。

第一题

- A. 一元运算符正号（+）、负号（-）的运算优先级都很高，但是 `->` 和 `.` 的优先级更高。
- B. `a[i] = i++;` 这一语句的行为是未定义的，结果取决于编译器自身规则，`a[3]` 可能等于 2、3、4。
- D. `10` 和 `11` 都是十进制数，并且 `==` 运算优先级更高，先算前半部分。

第二题

- B. 函数调用任何外部变量，除了全局变量外都需要使用 `extern` 进行声明，全局变量可以直接调用。
- C. 函数可以通过调用自身实现递归，递归的求解方式代码更简洁，但占用的时间资源更多。
- D. precision 的值必须是一个非负整数，可以是 0。

第三题

- A. `int *a[50]` 定义 `a` 是一个指针数组，`a` 是 `50` 个指向一个 `int` 型变量的指针
- B. C 语言可以定义多维数组，数组维数没有上限。
- C. 我们总是可以使用 `sizeof(a) / sizeof(a[0])` 来获得数组 `a` 的长度，`a[1]` 可能不存在。

第四题

- A. 除 `void *` 指针外，需要不同类型指针互相赋值时，必须通过强制类型转换改变对指针类型的解释
- B. 指针也是变量，指针在储存时占用的储存空间大小和它指向的变量无关，所有指针变量占用空间相同。
- C. 指针和数组之间的关系十分密切，所有通过数组下标完成的操作都可以通过指针实现

第五题

- A. `unsigned int` 所能储存的最大正整数是 `int` 所能储存的最大正整数的两倍再加一。
- B. 声明为 `static` 的对象不会随着退出程序块而消失，只在其作用域内有效，不能被其他代码用 `extern` 访问。
- C. 隐式类型转换和显式类型转换（强制类型转换）可以同时存在。

B - 求个和吧

难度	考点
1	简单循环、简单计算

题目分析

本题根据题目要求计算各位数之和即可，可以使用循环计算，也可以使用表达式（即手动把 4 个位都提取出来后求和）计算。

示例代码

```c
#include <stdio.h>

int main()
{
    int n, sum = 0;

    scanf("%d", &n);
    while (n != 0)
    {
        sum += n % 10;
        n /= 10;
    }
    printf("%d", sum);

    return 0;
}
```

C - 度分布

难度	考点
1	简单循环、类型转换，输入输出

题目分析

通过题目描述，我们可以抽象出这样一个模型：给定 n 个节点，每个节点有一个权值，找出其中最大的权值 $maxk$，分别输出权值为 $1, 2, \ldots, maxk$ 的节点的比例。

示例程序

```c
#include <stdio.h>

int sum[101];

int main()
{
    int n, i, Max = 0;
    int x;
    double p;

    scanf("%d", &n);
    for (i = 1; i <= n; i++)
    {
        scanf("%d", &x);
        sum[x]++;
        if (x > Max) Max = x;
    }
    for (i = 1; i <= Max; i++)
    {
        p = sum[i] * 1.0 / n;
        printf("%.2f ", p);
    }

    return 0;
}
```

D - 大家一起刷 TD 3

考点	难度
数组	2

题目分析

可以利用一个数组 $a[i]$，来记录第 i 天刷了多少次 TD，最后遍历整个数组，统计总共刷了多少次，如果一天之内刷了超过 3 次的 TD，则认为是 3 次。

示例代码

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

int a[80];

int main()
{
    int T, n, sum = 0, i;

    scanf("%d", &T);
    while (T--)
    {
        scanf("%d", &n);
        a[n]++;
    }
    for (i = 1; i <= 77; i++)
    {
        if (a[i] > 3) a[i] = 3;
        sum += a[i];
    }
    printf("%d", sum);

    return 0;
}
```

E - Detect the Overflow!!!

考点	难度
判断	4

思路一

从数学的角度考虑，当 a, b 均不为零时，有：

$$ab < 2^{64} \iff a < \frac{2^{64}}{b}$$

记 $L = \left\lfloor \frac{2^{64} - 1}{b} \right\rfloor$，则：

当 $a \leqslant L$ 时，$a \times b \leqslant 2^{64} - 1$；

当 $a > L$ 时，$a \times b \geqslant (L + 1) \times b > 2^{64} - 1$。

因此可据此判断是否自然溢出。

示例代码一

```c
#include <stdio.h>

#define ull unsigned long long
ull Max = 0xFFFFFFFFFFFFFFFF; // 0xFFFFFFFFFFFFFFFF 在 unsiged long long 下为 2^64 - 1，即 unsigned long long 的最大值

int main()
{
    ull T, a, b, ovf;
    scanf("%llu", &T);
    while (T--)
    {
        scanf("%llu%llu", &a, &b);
        if (!a || !b)
            ovf = 0;
        else
        {
            ull L = Max / b;
            if (a <= L)
                ovf = 0;
            else
                ovf = 1;
        }
        if (ovf)
            puts("But it is the same for mudulo 2^64.");
        else
            puts("Here's where the parade begins!");
    }

    return 0;
}
```

思路二

我们将 64 位的数据均分割成 32 位宽的数据，模拟竖式乘法相乘合并即可。

注意进位的处理。

示例代码二

```
#include <stdio.h>

#define ull unsigned long long
const ull mask = ((1ull << 32) - 1);

int main()
{
    ull T, a, b;
    ull ahi, alo, bhi, blo, chihi, chilo, clohi, clolo;

    scanf("%llu", &T);
    while (T--)
    {
        scanf("%llu%llu", &a, &b);
        ahi = (a >> 32);
        alo = a & mask;
        bhi = (b >> 32);
        blo = b & mask;
        chihi = (ahi * bhi) >> 32;
        chilo = ((ahi * bhi) & mask) + ((alo * bhi) >> 32) + ((ahi * blo) >> 32);
        clohi = ((ahi * blo) & mask) + ((alo * bhi) & mask) + ((alo * blo) >> 32);
        clolo = ((alo * blo) & mask);
        clohi += clolo >> 32;
        clolo &= mask;
        chilo += clohi >> 32;
        clohi &= mask;
        chihi += chilo >> 32;
        chilo &= mask;
        if (chihi || chilo)
            puts("But it is the same for mudulo 2^64.");
        else
            puts("Here's where the parade begins!");
    }

    return 0;
}
```

F - 均值插值法

考点	难度
二维数组	3

题目分析

本题考查二维数组的使用，以及数组的初始化。

需要注意的是，大数组最好定义为全局变量。如果数组被定义为局部变量，那么其元素不会被自动初始化为 0，需要进行手动初始化。

示例代码

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

#define N 500 + 5

int a[N][N];

int main()
{
    int n, m, p, q;

    scanf("%d%d%d%d", &n, &m, &p, &q);

    int i, j;
    for (i = 1; i <= n; ++i)
    {
        for (j = 1; j <= m; ++j)
        {
            scanf("%d", &a[i][j]);
        }
    }
    for (i = 1; i <= p; ++i)
    {
        for (j = 1; j <= q; ++j)
        {
            int c = i * n / p, d = j * m / q;
            printf("%d ", (a[c][d] + a[c + 1][d] + a[c][d + 1] + a[c + 1][d + 1]) / 4);
        }
        printf("\n");
    }

    return 0;
}
```

G - 顺序统计量

难度	考点
4	`qsort()` 的使用

题目分析

本题由 PPT 中的例题改编而来，考察了 `qsort` 排序和二维数组的应用。

易错点提示

- 提供两种 `qsort` 排序的写法，示例程序一必须保存为 `.c` 后缀文件，在保存为 `.cpp` 后缀时会报错，这与 `qsort` 函数对指针类型的要求有关。
- 示例程序二保存为 `.c` 或 `.cpp` 后缀文件均可，但必须引用 `stdlib.h` 头文件，同时需要进行指针的强制类型转换，这种写法不易出错适用范围更广，**推荐这种写法。**

示例程序一

```c
// 需要保存为 .c 后缀文件
#include <stdio.h>

#define maxn 200010
int data[maxn][3];

int cmp_rank(const int *p1, const int *p2)
{
    return p1[0] - p2[0];
}

int cmp_order(const int *p1, const int *p2)
{
    return p1[1] - p2[1];
}

void get_rank(int n)
{
    int i;
    data[1][2] = 1;
    for (i = 2; i <= n; i++)
    {
        if (data[i][0] == data[i - 1][0])
            data[i][2] = data[i - 1][2];
        else
            data[i][2] = data[i - 1][2] + 1;
    }
}

int main()
{
    int i, n;
    scanf("%d", &n);
```

```
    // 读入数据，在data[i][0]位置处保存数值
    // 在data[i][1]位置处保存原位置大小
    for (i = 1; i <= n; i++)
    {
        scanf("%d", &data[i][0]);
        data[i][1] = i;
    }
    // 按照数值大小排序，为编号做准备
    qsort(data + 1, n, sizeof(data[0]), cmp_rank);
    // 进行编号操作
    get_rank(n);
    // 按照原位置大小排序，恢复为输入时的顺序
    qsort(data + 1, n, sizeof(data[0]), cmp_order);
    // 按照格式进行输出
    for (i = 1; i <= n; i++)
    {
        printf("%d: [%d]", data[i][0], data[i][2]);
        if (i < n) puts("");
    }

    return 0;
}
```

示例程序二

```
// 两种均可，不易出错，更加推荐
#include <stdio.h>
#include <stdlib.h>

#define maxn 200010
int data[maxn][3];

int cmp_rank(const void *p1, const void *p2)
{
    int *a = (int *)p1;
    int *b = (int *)p2;
    return a[0] - b[0];
}

int cmp_order(const void *p1, const void *p2)
{
    int *a = (int *)p1;
    int *b = (int *)p2;
    return a[1] - b[1];
}

void get_rank(int n)
{
    int i;
    data[1][2] = 1;
    for (i = 2; i <= n; i++)
    {
        if (data[i][0] == data[i - 1][0])
            data[i][2] = data[i - 1][2];
        else
            data[i][2] = data[i - 1][2] + 1;
```

```
    }
}

int main()
{
    int i, n;

    scanf("%d", &n);
    for (i = 1; i <= n; i++)
    {
        scanf("%d", &data[i][0]);
        data[i][1] = i;
    }
    qsort(data + 1, n, sizeof(data[0]), cmp_rank);
    get_rank(n);
    qsort(data + 1, n, sizeof(data[0]), cmp_order);
    for (i = 1; i <= n; i++)
    {
        printf("%d: [%d]", data[i][0], data[i][2]);
        if (i < n) puts("");
    }

    return 0;
}
```

H - intX_t

考点	难度
高精度、补码	5

题目分析

本题是一道高精度运算题目，首先再帮同学们复习一下补码的表示方法：

- 正数的补码就是它的原码（二进制表示）
- 负数的补码是它相反数的原码按位取反再 $+1$

那么在本题中，我们首先可以根据最高位是 1 还是 0 判断出数的符号（如果全是 0 也可以当做正数）。如果是负数的话，我们进行一次减一再按位取反的操作。这个操作其实等价于找到最低位的 1，将高位全部取反即可，这样我们就得到了输入数绝对值的原码。

随后需要通过原码计算该数的绝对值，我们将原码从左向右处理，每处理一位就首先将结果乘 2（相当于做了一次左移运算），然后加上原码这一位的数码。对于中间需要的高精度数乘单精度数运算，这儿我们使用数组模拟竖式乘法，从结果的低位开始，按位依次乘 2，并加上之前的“进位”，每次所得结果的最后一位（模 10 余数）作为该位的计算结果，除 10 的结果作为“进位”参与下位运算，当超过被乘数的最高位后再把剩余“进位”依次向高位写出，即可完成一次乘 2 运算。

最后的输出部分首先根据符号判断是否需要输出负号，再将数组从高位到低位依次输出，注意不要输出前导 0。

本题和 E2-J Long Long Factorial 一题很像，同学们可以对比学习两道题目。

本题错误的同学可以考虑一些经典二进制数的补码表示，比如 0，-1，-2^{31}（1 后面 31 个 0）等。另外注意本题只有部分数据位于 `int` 范围内，想要 AC 的同学还是要按照高精度方式处理。

示例代码

```c
#include <stdio.h>

char s[1100];
int a[1100], ans[1100];

int main()
{
    int n;
    int f = 0;
    int i, j;
    int top;

    scanf("%d", &n);
    scanf("%s", s);
    for (i = 0; i < n; i++)
    {
        a[i] = s[i] - '0';
    }
    if (a[0] == 1)
    {
        f = 1;
```

```
        i = n;
        while (a[--i] == 0)
            ; // 找到最低位的1
        if (i)
        {
            for (j = 0; j < i; j++)
                a[j] = a[j] ^ 1; // 高位取反
        }
    }

    top = 1;
    for (i = 0; i < n; i++)
    {
        int carry = 0;
        for (j = 0; j < top; j++)
        {
            ans[j] *= 2;          // 本位乘2
            ans[j] += carry;      // 加上进位
            carry = ans[j] / 10; // 算出进位
            ans[j] %= 10;         // 本位的结果
        }
        if (carry) // 还有进位
            ans[top++] = carry;
        ans[0] += a[i];
    }
    if (f == 1)
        putchar('-');
    for (i = top - 1; i >= 0; i--)
    {
        printf("%d", ans[i]);
    }

    return 0;
}
```

I - 一阶泰勒展开

考点	难度
字符串处理	6

题目分析

这道题比较类似 `E7-J`（二元多项式求偏导），是该题的简化版本。

同样只需要预处理首项后对于每一项用二元组存储。发现答案必为 $px+q+R(x)$，直接从公式中代入 $f(x_0)$ 和 $f'(x_0)$ 求出 p,q 即可。

示例程序

```c
#include <stdio.h>

#define N 1010
#define ll long long

char str[N], *p;

char *normalize(char *x) {
    if (x[1] == '-') return x + 1;
    x[0] = '+';
    return x;
}

int isDigit(char x) {
    return x >= '0' && x <= '9';
}

ll coef[10];

int main() {
    int i, x0;

    scanf("%d", &x0);
    scanf("%s", str + 1);
    p = normalize(str);
    while (*p) {
        int sign = (*p++) == '+' ? 1 : -1;
        ll c = 0, a = 0;
        // coefficiency
        if ((*p) == 'x') c = 1;
        else {
            while (isDigit(*p))
                c = c * 10 + (*p++) - '0';
        }
        // 'x', power
        if ((*p) == 'x') {
            p++;
            if ((*p) == '^') {
                p++;
```

```
                while (isDigit(*p))
                    a = a * 10 + (*p++) - '0';
            } else {
                a = 1;
            }
        }
        coef[a] += c * sign;
    }
    // f(x) = f(x0) + f'(x0)(x - x0)
    ll fx = 0, dfx = 0;
    for (i = 5; i >= 0; i--) fx = fx * x0 + coef[i];
    for (i = 5; i >= 1; i--) dfx = dfx * x0 + 1LL * i * coef[i];
    // px + q + R(x)
    ll p = dfx, q = fx - dfx * x0;
    int first = 1;
    if (p) {
        if (p == -1) printf("-");
        else if (p != 1) printf("%lld", p);
        printf("x");
        first = 0;
    }
    if (q) {
        if (!first && q > 0) printf("+");
        printf("%lld", q);
        first = 0;
    }
    if (!first) printf("+");
    printf("R(x)");

    return 0;
}
```

J - 69 岁，是异灵术

难度	考点
7	模拟

题目分析

题目数据范围很小，我们可以直接暴力模拟游戏过程的每一步，并判断每一个时刻能否使用**教科书般的亵渎**进行清场（使得场上不存在未死亡随从）。

考虑对于一个游戏时刻的场上状态，可以使用**教科书般的亵渎**进行清场当且仅当：

- 设场上未死亡随从的最大生命值为 x，则场上必须存在生命值分别为 $1, 2, \cdots, x$ 的未死亡随从。
- 特别地：
  - 只具有**圣盾**的生命值为 b 的随从等价于生命值为 $b+1$ 的随从。
  - 只具有**亡语**的生命值为 b 的随从，设其死亡后产生的随从生命值为 f，则其等价于两个随从，生命值分别为 b 和 $b+f$。
  - 既具有**圣盾**又具有**亡语**的生命值为 b 的随从，设其死亡后产生的随从生命值为 f，则其等价于两个随从，生命值分别为 $b+1$ 和 $b+f+1$。

上述判断过程可以通过遍历一遍所有场上未死亡随从得到，注意，不要忘记场上随从必须大于 2 这个条件。

示例程序1

不使用结构体。

```
#include <stdio.h>
#define max(a, b) (a > b ? a : b)

int a[2][10], b[2][10], c[2][10], d[2][10], e[2][10], f[2][10];

int check()
{
    int C[30] = {0}, mx = 0, i, j, cnt = 0;
    for (i = 0; i <= 1; i++)
    {
        for (j = 1; j <= 7; j++)
        {
            if (b[i][j] <= 0) continue;
            cnt++;
            C[b[i][j] + d[i][j]] = 1;
            C[b[i][j] + d[i][j] + f[i][j]] = 1;
            mx = max(mx, b[i][j] + d[i][j] + f[i][j]);
        }
    }
    if (cnt <= 2) return 0;
    for (i = 1; i <= mx; i++)
        if (!C[i]) return 0;
    return 1;
}

```

```
int main()
{
    int t, i, j, suc, x, y;

    scanf("%d", &t);
    while (t--)
    {
        for (i = 0; i <= 1; i++)
            for (j = 1; j <= 7; j++) scanf("%d%d%d%d%d%d", &a[i][j], &b[i][j], &c[i][j], &d[i][j], &e[i][j], &f[i][j]);
        suc = check();
        for (i = 0; !suc; i ^= 1)
        {
            x = y = 0;
            for (j = 1; j <= 7; j++)
            {
                if (b[i][j] > 0 && (!x || a[i][j] > a[i][x]))
                {
                    x = j;
                }
            }
            for (j = 1; j <= 7; j++)
            {
                if (b[i ^ 1][j] > 0 && (!y || c[i ^ 1][j] > c[i ^ 1][y] || c[i ^ 1][j] == c[i ^ 1][y] && b[i ^ 1][j] > b[i ^ 1][y]))
                {
                    y = j;
                }
            }
            if (!x || !y) break;
            if (d[i][x])
            {
                d[i][x] = 0;
            }
            else
            {
                b[i][x] -= a[i ^ 1][y];
            }
            if (d[i ^ 1][y])
            {
                d[i ^ 1][y] = 0;
            }
            else
            {
                b[i ^ 1][y] -= a[i][x];
            }
            if (b[i][x] <= 0 && e[i][x])
            {
                a[i][x] = e[i][x], b[i][x] = f[i][x];
                c[i][x] = d[i][x] = e[i][x] = f[i][x] = 0;
            }
            if (b[i ^ 1][y] <= 0 && e[i ^ 1][y])
            {
                a[i ^ 1][y] = e[i ^ 1][y], b[i ^ 1][y] = f[i ^ 1][y];
                c[i ^ 1][y] = d[i ^ 1][y] = e[i ^ 1][y] = f[i ^ 1][y] = 0;
            }
            suc |= check();
        }
```

```
        puts(suc ? "1" : "0");
    }

    return 0;
}
```

示例程序 2

使用结构体。

```
#include <stdio.h>
#define max(a, b) (a > b ? a : b)

struct Minion
{
    int a, b, c, d, e, f;
} A[2][10];

int check()
{
    int c[30] = {0}, mx = 0, i, j, cnt = 0;
    for (i = 0; i <= 1; i++)
    {
        for (j = 1; j <= 7; j++)
        {
            if (A[i][j].b <= 0) continue;
            cnt++;
            c[A[i][j].b + A[i][j].d] = 1;
            c[A[i][j].b + A[i][j].d + A[i][j].f] = 1;
            mx = max(mx, A[i][j].b + A[i][j].d + A[i][j].f);
        }
    }
    if (cnt <= 2) return 0;
    for (i = 1; i <= mx; i++)
        if (!c[i]) return 0;
    return 1;
}

int main()
{
    int t, i, j, suc, x, y;

    scanf("%d", &t);
    while (t--)
    {
        for (i = 0; i <= 1; i++)
            for (j = 1; j <= 7; j++) scanf("%d%d%d%d%d%d", &A[i][j].a, &A[i][j].b, &A[i][j].c, &A[i][j].d, &A[i][j].e, &A[i][j].f);
        suc = check();
        for (i = 0; !suc; i ^= 1)
        {
            x = y = 0;
            for (j = 1; j <= 7; j++)
            {
                if (A[i][j].b > 0 && (!x || A[i][j].a > A[i][x].a))
                {
```

```
                    x = j;
                }
            }
            for (j = 1; j <= 7; j++)
            {
                if (A[i ^ 1][j].b > 0 && (!y || A[i ^ 1][j].c > A[i ^ 1]
[y].c || A[i ^ 1][j].c == A[i ^ 1][y].c && A[i ^ 1][j].b > A[i ^ 1][y].b))
                {
                    y = j;
                }
            }
            if (!x || !y) break;
            if (A[i][x].d)
            {
                A[i][x].d = 0;
            }
            else
            {
                A[i][x].b -= A[i ^ 1][y].a;
            }
            if (A[i ^ 1][y].d)
            {
                A[i ^ 1][y].d = 0;
            }
            else
            {
                A[i ^ 1][y].b -= A[i][x].a;
            }
            if (A[i][x].b <= 0 && A[i][x].e)
            {
                A[i][x].a = A[i][x].e, A[i][x].b = A[i][x].f;
                A[i][x].c = A[i][x].d = A[i][x].e = A[i][x].f = 0;
            }
            if (A[i ^ 1][y].b <= 0 && A[i ^ 1][y].e)
            {
                A[i ^ 1][y].a = A[i ^ 1][y].e, A[i ^ 1][y].b = A[i ^ 1]
[y].f;
                A[i ^ 1][y].c = A[i ^ 1][y].d = A[i ^ 1][y].e = A[i ^ 1]
[y].f = 0;
            }
            suc |= check();
        }
        puts(suc ? "1" : "0");
    }

    return 0;
}
```