

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

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int n, sum = 0;
6
7      scanf("%d", &n);
8      while (n != 0)
9      {
10         sum += n % 10;
11         n /= 10;
12     }
13     printf("%d", sum);
14
15     return 0;
16 }
```

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 $1, 2, \dots, maxk$ 的节点的比例。

示例程序

```
1  #include <stdio.h>
2
3  int sum[101];
4
5  int main()
6  {
7      int n, i, Max = 0;
8      int x;
9      double p;
10
11     scanf("%d", &n);
12     for (i = 1; i <= n; i++)
13     {
14         scanf("%d", &x);
15         sum[x]++;
16         if (x > Max) Max = x;
17     }
18     for (i = 1; i <= Max; i++)
19     {
20         p = sum[i] * 1.0 / n;
21         printf("%.2f ", p);
22     }
23
24     return 0;
25 }
```

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

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int a[80];
5
6  int main()
7  {
8      int T, n, sum = 0, i;
9
10     scanf("%d", &T);
11     while (T--)
12     {
13         scanf("%d", &n);
14         a[n]++;
15     }
16     for (i = 1; i <= 77; i++)
17     {
18         if (a[i] > 3) a[i] = 3;
19         sum += a[i];
20     }
21     printf("%d", sum);
22
23     return 0;
24 }
```

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

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

因此可据此判断是否自然溢出。

示例代码一

```
1  #include <stdio.h>
2
3  #define ull unsigned long long
4  ull Max = 0xFFFFFFFFFFFFFFFF; // 0xFFFFFFFFFFFFFFFF 在 unsigned long long 下为
   2^64 - 1, 即 unsigned long long 的最大值
5
6  int main()
7  {
8      ull T, a, b, ovf;
9      scanf("%llu", &T);
10     while (T--)
11     {
12         scanf("%llu%llu", &a, &b);
13         if (!a || !b)
14             ovf = 0;
15         else
16         {
17             ull L = Max / b;
18             if (a <= L)
19                 ovf = 0;
20             else
21                 ovf = 1;
22         }
23         if (ovf)
24             puts("But it is the same for modulo 2^64.");
25         else
26             puts("Here's where the parade begins!");
27     }
28
29     return 0;
30 }
```

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

```
1  #include <stdio.h>
2
3  #define ull unsigned long long
4  const ull mask = ((1ull << 32) - 1);
5
6  int main()
7  {
8      ull T, a, b;
9      ull ahi, alo, bhi, blo, chihi, chilo, clohi, clolo;
10
11     scanf("%llu", &T);
12     while (T--)
13     {
14         scanf("%llu%llu", &a, &b);
15         ahi = (a >> 32);
16         alo = a & mask;
17         bhi = (b >> 32);
18         blo = b & mask;
19         chihi = (ahi * bhi) >> 32;
20         chilo = ((ahi * bhi) & mask) + ((alo * bhi) >> 32) + ((ahi * blo) >>
21         32);
22         clohi = ((ahi * blo) & mask) + ((alo * bhi) & mask) + ((alo * blo)
23         >> 32);
24         clolo = ((alo * blo) & mask);
25         clohi += clolo >> 32;
26         clolo &= mask;
27         chilo += clohi >> 32;
28         clohi &= mask;
29         chihi += chilo >> 32;
30         chilo &= mask;
31         if (chihi || chilo)
32             puts("But it is the same for modulo 2^64.");
33         else
34             puts("Here's where the parade begins!");
35     }
36     return 0;
37 }
```

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

```
1  #include <assert.h>
2  #include <stdio.h>
3
4  #define N 500 + 5
5
6  int a[N][N];
7
8  int main()
9  {
10     int n, m, p, q;
11
12     scanf("%d%d%d%d", &n, &m, &p, &q);
13
14     int i, j;
15     for (i = 1; i <= n; ++i)
16     {
17         for (j = 1; j <= m; ++j)
18         {
19             scanf("%d", &a[i][j]);
20         }
21     }
22     for (i = 1; i <= p; ++i)
23     {
24         for (j = 1; j <= q; ++j)
25         {
26             int c = i * n / p, d = j * m / q;
27             printf("%d ", (a[c][d] + a[c + 1][d] + a[c][d + 1] + a[c + 1][d
+ 1]) / 4);
28         }
29         printf("\n");
30     }
31
32     return 0;
33 }
```

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

```
1 // 需要保存为 .c 后缀文件
2 #include <stdio.h>
3
4 #define maxn 200010
5 int data[maxn][3];
6
7 int cmp_rank(const int *p1, const int *p2)
8 {
9     return p1[0] - p2[0];
10 }
11
12 int cmp_order(const int *p1, const int *p2)
13 {
14     return p1[1] - p2[1];
15 }
16
17 void get_rank(int n)
18 {
19     int i;
20     data[1][2] = 1;
21     for (i = 2; i <= n; i++)
22     {
23         if (data[i][0] == data[i - 1][0])
24             data[i][2] = data[i - 1][2];
25         else
26             data[i][2] = data[i - 1][2] + 1;
27     }
28 }
29
30 int main()
31 {
32     int i, n;
33     scanf("%d", &n);
```



```

34 // 读入数据，在data[i][0]位置处保存数值
35 // 在data[i][1]位置处保存原位置大小
36 for (i = 1; i <= n; i++)
37 {
38     scanf("%d", &data[i][0]);
39     data[i][1] = i;
40 }
41 // 按照数值大小排序，为编号做准备
42 qsort(data + 1, n, sizeof(data[0]), cmp_rank);
43 // 进行编号操作
44 get_rank(n);
45 // 按照原位置大小排序，恢复为输入时的顺序
46 qsort(data + 1, n, sizeof(data[0]), cmp_order);
47 // 按照格式进行输出
48 for (i = 1; i <= n; i++)
49 {
50     printf("%d: [%d]", data[i][0], data[i][2]);
51     if (i < n) puts("");
52 }
53
54 return 0;
55 }

```

示例程序二

```

1 // 两种均可，不易出错，更加推荐
2 #include <stdio.h>
3 #include <stdlib.h>
4
5 #define maxn 200010
6 int data[maxn][3];
7
8 int cmp_rank(const void *p1, const void *p2)
9 {
10     int *a = (int *)p1;
11     int *b = (int *)p2;
12     return a[0] - b[0];
13 }
14
15 int cmp_order(const void *p1, const void *p2)
16 {
17     int *a = (int *)p1;
18     int *b = (int *)p2;
19     return a[1] - b[1];
20 }
21
22 void get_rank(int n)
23 {
24     int i;
25     data[1][2] = 1;
26     for (i = 2; i <= n; i++)
27     {
28         if (data[i][0] == data[i - 1][0])
29             data[i][2] = data[i - 1][2];
30         else
31             data[i][2] = data[i - 1][2] + 1;

```

```
32     }
33 }
34
35 int main()
36 {
37     int i, n;
38
39     scanf("%d", &n);
40     for (i = 1; i <= n; i++)
41     {
42         scanf("%d", &data[i][0]);
43         data[i][1] = i;
44     }
45     qsort(data + 1, n, sizeof(data[0]), cmp_rank);
46     get_rank(n);
47     qsort(data + 1, n, sizeof(data[0]), cmp_order);
48     for (i = 1; i <= n; i++)
49     {
50         printf("%d: [%d]", data[i][0], data[i][2]);
51         if (i < n) puts("");
52     }
53
54     return 0;
55 }
```

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

```
1  #include <stdio.h>
2
3  char s[1100];
4  int a[1100], ans[1100];
5
6  int main()
7  {
8      int n;
9      int f = 0;
10     int i, j;
11     int top;
12
13     scanf("%d", &n);
14     scanf("%s", s);
15     for (i = 0; i < n; i++)
16     {
17         a[i] = s[i] - '0';
18     }
19     if (a[0] == 1)
20     {
21         f = 1;
```

```
22     i = n;
23     while (a[--i] == 0)
24         ; // 找到最低位的1
25     if (i)
26     {
27         for (j = 0; j < i; j++)
28             a[j] = a[j] ^ 1; // 高位取反
29     }
30 }
31
32 top = 1;
33 for (i = 0; i < n; i++)
34 {
35     int carry = 0;
36     for (j = 0; j < top; j++)
37     {
38         ans[j] *= 2;           // 本位乘2
39         ans[j] += carry;       // 加上进位
40         carry = ans[j] / 10;    // 算出进位
41         ans[j] %= 10;          // 本位的结果
42     }
43     if (carry) // 还有进位
44         ans[top++] = carry;
45     ans[0] += a[i];
46 }
47 if (f == 1)
48     putchar('-');
49 for (i = top - 1; i >= 0; i--)
50 {
51     printf("%d", ans[i]);
52 }
53
54 return 0;
55 }
```

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

```
1  #include <stdio.h>
2
3  #define N 1010
4  #define ll long long
5
6  char str[N], *p;
7
8  char *normalize(char *x) {
9      if (x[1] == '-') return x + 1;
10     x[0] = '+';
11     return x;
12 }
13
14 int isDigit(char x) {
15     return x >= '0' && x <= '9';
16 }
17
18 ll coef[10];
19
20 int main() {
21     int i, x0;
22
23     scanf("%d", &x0);
24     scanf("%s", str + 1);
25     p = normalize(str);
26     while (*p) {
27         int sign = (*p++) == '+' ? 1 : -1;
28         ll c = 0, a = 0;
29         // coefficient
30         if ((*p) == 'x') c = 1;
31         else {
32             while (isDigit(*p))
33                 c = c * 10 + (*p++) - '0';
34         }
35         // 'x', power
36         if ((*p) == 'x') {
37             p++;
38             if ((*p) == '^') {
39                 p++;
```

```

40         while (isDigit(*p))
41             a = a * 10 + (*p++) - '0';
42     } else {
43         a = 1;
44     }
45 }
46     coef[a] += c * sign;
47 }
48 // f(x) = f(x0) + f'(x0)(x - x0)
49 ll fx = 0, dfx = 0;
50 for (i = 5; i >= 0; i--) fx = fx * x0 + coef[i];
51 for (i = 5; i >= 1; i--) dfx = dfx * x0 + 1LL * i * coef[i];
52 // px + q + R(x)
53 ll p = dfx, q = fx - dfx * x0;
54 int first = 1;
55 if (p) {
56     if (p == -1) printf("-");
57     else if (p != 1) printf("%lld", p);
58     printf("x");
59     first = 0;
60 }
61 if (q) {
62     if (!first && q > 0) printf("+");
63     printf("%lld", q);
64     first = 0;
65 }
66 if (!first) printf("+");
67 printf("R(x)");
68
69     return 0;
70 }

```

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**教科书般的衰读**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**教科书般的衰读**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。

```
1  #include <stdio.h>
2  #define max(a, b) (a > b ? a : b)
3
4  int a[2][10], b[2][10], c[2][10], d[2][10], e[2][10], f[2][10];
5
6  int check()
7  {
8      int c[30] = {0}, mx = 0, i, j, cnt = 0;
9      for (i = 0; i <= 1; i++)
10     {
11         for (j = 1; j <= 7; j++)
12         {
13             if (b[i][j] <= 0) continue;
14             cnt++;
15             c[b[i][j] + d[i][j]] = 1;
16             c[b[i][j] + d[i][j] + f[i][j]] = 1;
17             mx = max(mx, b[i][j] + d[i][j] + f[i][j]);
18         }
19     }
20     if (cnt <= 2) return 0;
21     for (i = 1; i <= mx; i++)
22         if (!c[i]) return 0;
23     return 1;
24 }
25
```

```

26 int main()
27 {
28     int t, i, j, suc, x, y;
29
30     scanf("%d", &t);
31     while (t--)
32     {
33         for (i = 0; i <= 1; i++)
34             for (j = 1; j <= 7; j++) scanf("%d%d%d%d%d", &a[i][j], &b[i]
18 [j], &c[i][j], &d[i][j], &e[i][j], &f[i][j]);
35         suc = check();
36         for (i = 0; !suc; i ^= 1)
37         {
38             x = y = 0;
39             for (j = 1; j <= 7; j++)
40             {
41                 if (b[i][j] > 0 && (!x || a[i][j] > a[i][x]))
42                 {
43                     x = j;
44                 }
45             }
46             for (j = 1; j <= 7; j++)
47             {
48                 if (b[i ^ 1][j] > 0 && (!y || c[i ^ 1][j] > c[i ^ 1][y] ||
18 c[i ^ 1][j] == c[i ^ 1][y] && b[i ^ 1][j] > b[i ^ 1][y]))
49                 {
50                     y = j;
51                 }
52             }
53             if (!x || !y) break;
54             if (d[i][x])
55             {
56                 d[i][x] = 0;
57             }
58             else
59             {
60                 b[i][x] -= a[i ^ 1][y];
61             }
62             if (d[i ^ 1][y])
63             {
64                 d[i ^ 1][y] = 0;
65             }
66             else
67             {
68                 b[i ^ 1][y] -= a[i][x];
69             }
70             if (b[i][x] <= 0 && e[i][x])
71             {
72                 a[i][x] = e[i][x], b[i][x] = f[i][x];
73                 c[i][x] = d[i][x] = e[i][x] = f[i][x] = 0;
74             }
75             if (b[i ^ 1][y] <= 0 && e[i ^ 1][y])
76             {
77                 a[i ^ 1][y] = e[i ^ 1][y], b[i ^ 1][y] = f[i ^ 1][y];
78                 c[i ^ 1][y] = d[i ^ 1][y] = e[i ^ 1][y] = f[i ^ 1][y] = 0;
79             }
80             suc |= check();
81         }

```



```

82     puts(suc ? "1" : "0");
83 }
84
85 return 0;
86 }

```

示例程序 2

使用结构体。

```

1  #include <stdio.h>
2  #define max(a, b) (a > b ? a : b)
3
4  struct Minion
5  {
6      int a, b, c, d, e, f;
7  } A[2][10];
8
9  int check()
10 {
11     int c[30] = {0}, mx = 0, i, j, cnt = 0;
12     for (i = 0; i <= 1; i++)
13     {
14         for (j = 1; j <= 7; j++)
15         {
16             if (A[i][j].b <= 0) continue;
17             cnt++;
18             c[A[i][j].b + A[i][j].d] = 1;
19             c[A[i][j].b + A[i][j].d + A[i][j].f] = 1;
20             mx = max(mx, A[i][j].b + A[i][j].d + A[i][j].f);
21         }
22     }
23     if (cnt <= 2) return 0;
24     for (i = 1; i <= mx; i++)
25         if (!c[i]) return 0;
26     return 1;
27 }
28
29 int main()
30 {
31     int t, i, j, suc, x, y;
32
33     scanf("%d", &t);
34     while (t--)
35     {
36         for (i = 0; i <= 1; i++)
37             for (j = 1; j <= 7; j++) scanf("%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);
38         suc = check();
39         for (i = 0; !suc; i ^= 1)
40         {
41             x = y = 0;
42             for (j = 1; j <= 7; j++)
43             {
44                 if (A[i][j].b > 0 && (!x || A[i][j].a > A[i][x].a))
45                     {

```

```

46         x = j;
47     }
48 }
49 for (j = 1; j <= 7; j++)
50 {
51     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))
52     {
53         y = j;
54     }
55 }
56 if (!x || !y) break;
57 if (A[i][x].d)
58 {
59     A[i][x].d = 0;
60 }
61 else
62 {
63     A[i][x].b -= A[i ^ 1][y].a;
64 }
65 if (A[i ^ 1][y].d)
66 {
67     A[i ^ 1][y].d = 0;
68 }
69 else
70 {
71     A[i ^ 1][y].b -= A[i][x].a;
72 }
73 if (A[i][x].b <= 0 && A[i][x].e)
74 {
75     A[i][x].a = A[i][x].e, A[i][x].b = A[i][x].f;
76     A[i][x].c = A[i][x].d = A[i][x].e = A[i][x].f = 0;
77 }
78 if (A[i ^ 1][y].b <= 0 && A[i ^ 1][y].e)
79 {
80     A[i ^ 1][y].a = A[i ^ 1][y].e, A[i ^ 1][y].b = A[i ^ 1]
[y].f;
81     A[i ^ 1][y].c = A[i ^ 1][y].d = A[i ^ 1][y].e = A[i ^ 1]
[y].f = 0;
82 }
83 suc |= check();
84 }
85 puts(suc ? "1" : "0");
86 }
87
88 return 0;
89 }

```