嵌入式工作室一轮 C 语言基础

出题人：LRL，QQ：525687841

C 语言是嵌入式开发的基础，也是 CS 学习的根基，通过学习 C 语言将帮助我们更好地深入理解计算机系统。嵌入式工作室一轮 C 语言基础将从零基础开始，通过开放探究式的题目，带领大家快速入门 C 语言，并具备一定独立编写 C 程序的能力。

在开始完成题目之前，**请先阅读附录了解做题要求**。点此获得更好的阅读体验和最新题目更新。

工欲善其事，必先利其器

编程学习不能纸上谈兵，颅内编程。在学习 C 语言的过程中，我们希望你能勤于思考，动手实践，复现教程代码，并尝试修改再观察运行结果。

在本题中，你需要搭建 C 语言开发环境（最好在 Linux 环境下），并掌握基本的单步调试方法。为了减少不必要的麻烦，推荐**统一使用 GCC（GNU Compiler Collection） 跨平台编译器套件**，代码编辑器或 IDE 没有限制。推荐选择 `VsCode/Sublime/Neovim` 或 `CLion` 的任意其一即可。

请注意，在学习 C 语言的过程中不要碰瓷 C++，C 语言文件的通常扩展名是 `.c, .h`，而 C++ 文件的扩展名通常是 `.cpp, .cc, .hpp` 等。`C style` 与 `C++ style` 也有很大的不同。因此在做题的过程**请勿使用 C++ 语法**。

加分项：

✅ Linux 开发环境

✅ 支持最新 `c2x` 标准（`-std=c2x`）

✅ 拥有美观舒适的代码编辑环境（语法高亮、LSP 自动补全等）

> Note：如果无法满足上述条件，可以暂时跳过。以上要求不是必需，不会影响后面部分题目的完成。

Q1：请使用命令行编译并执行下面这段代码？（需要编译器支持最新 `c2x` 标准）

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

[[noreturn]] void welcome() {
    typeof(const char*) str = "Welcome to join Embedded Studio!";
    puts(false ? : str);
    exit(EXIT_SUCCESS);
}

int main () {
    welcome();
}
```

上述代码主要用于测试开发环境，如果你的编译器不支持最新 `c2x` 标准，请替换为下面的代码：

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

void welcome() {
    const char* str = "Welcome to join Embedded Studio!";
    puts(str);
    exit(EXIT_SUCCESS);
}

int main () {
    welcome();
}
```

Q2：使用 GDB （不要求一定使用 GDB 命令行）调试上面的代码，观察变量 `str` 在内存中的地址和值。

> Tips：为了更好的调试体验，你可能需要添加编译选项让生成的程序中有 *debugging symbols*

Q3：写出几条你了解的其他 GCC 编译指令，并演示他们的用途。

请在解题报告中记录你配置开发环境的过程，并回答上述问题。

分支与循环

分支：程序的执行也不是一成不变的，往往会要求程序能够在不同的场合下有不同的动作。这时就需要在代码中使用条件语句来做出不同的选择。

循环： 虽然计算机可以在短时间批量处理成千上万条指令，但是不少问题中有许多规律性的重复操作，比如说计算几百个学生的平均分，或者对上万人的名单进行排序。仅使用顺序或者分支结构，对每一步操作都写出对应的语句是不可能的；但可以使用循环语句让计算机反复执行类似的任务。

请自行学习 C 语言的相关知识点：`if-else`，`switch-case`，`for`，`while` 与 `do-while` 结构。并完成下面两道题目：

重拾小学数学

给出三条线段 a, b, c 的长度，均是不大于 10000 的正整数。打算把这三条线段拼成一个三角形，它可以是什么三角形呢？

- 如果三条线段不能组成一个三角形，输出 `Not triangle`；
- 如果是直角三角形，输出 `Right triangle`；
- 如果是锐角三角形，输出 `Acute triangle`；
- 如果是钝角三角形，输出 `Obtuse triangle`；
- 如果是等腰三角形，输出 `Isosceles triangle`；
- 如果是等边三角形，输出 `Equilateral triangle`。

如果这个三角形符合以上多个条件，请按以上顺序分别输出，并用换行符隔开。

输入示例#1:

```
3 3 3
```

输出示例#1:

```
Acute triangle
Isosceles triangle
Equilateral triangle
```

输入示例#2:

```
6 10 6
```

输出示例#2:

```
Obtuse triangle
Isosceles triangle
```

方阵？内卷！

读入一个正整数 n $(1 \leq n \leq 50)$，输出 $n \times n$ 的“内卷方阵”。

从左上角填上 1 开始，顺时针方向依次填入数字，如同样例所示。注意每个数字有都会占用 3 个字符，前面使用空格补齐。

输入示例：

```
4
```

输出示例：

```
  1  2  3  4
 12 13 14  5
 11 16 15  6
 10  9  8  7
```

请在完成一道题目后自行验证代码的正确性，至少构造 5 组测试数据，并在解题报告中附上相应的运行截图。同时完成一道题目后请及时 `git commit`

> Note：虽然正确性的验证过程是不够严谨的，但有助于体验完整编码测试的过程，从初学者学习语法的目标角度已经足够了。

函数

有时程序中会使用多次相同的语句，而且无法通过循环来减少重复编程。对于这样的功能，如果能像使用 `sqrt()`、`max()` 这样变成一个函数，那就可以实现代码复用了！其实每个程序都用到了主函数——`main()`。除此之外，还可以自己定义其他函数，并将参数喂给这些函数，使其能够根据这些参数完成要求的任务。不过这方面还有更复杂的一些知识点，比如参数传递与变量的作用域，接下来也需要学习。函数内还能调用自己，也就是递归函数，这是程序设计新手入门公认的第一道坎，但却是非常重要的一部分。

请自行学习函数相关知识点：函数的声明，获取函数返回值，形参与实参，递归，函数指针。

> Note：函数指针可以与下一节指针一起学习

你也是函数大师

`strlen()`, `memcpy()`, `memset()` 等都是 C 语言最常用的函数，我们无时无刻不享受着函数带来的便利。知其然更需知其所以然，请你也用 C 语言简单实现一下 `strlen()`, `memcpy()`, `memset()` 吧。

```c
unsigned long strlen(const char *str) {
    // Your code
}

void *memcpy(void *restrict dest, const void *restrict src, size_t n)
{
    // Your code
}

void *memset(void *__s, int __c, size_t __n) {
    // Your code
}
```

在你写代码实现之前，为了帮助你更好的理解 `strlen()` 函数和字符串的存储机制，请先阅读以下代码并解释输出结果的由来。

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

int main() {
    char str[] = "Embedded Studio";
    int n = strlen(str);
    for (int i = 0; i < n; ++i)
        if (str[i] == ' ') {
            str[i] = 0;
            break;
        }
    int n1 = strlen(str), n2 = strlen(str + n1 + 1);
    printf("n1 = %d, n2 = %d\n", n1, n2);
    printf("str1 = %s\nstr2 = %s\n", str, str + n1 + 1);
    return 0;
}
```

输出结果：

```
n1 = 8, n2 = 6
str1 = Embedded
str2 = Studio
```

Q: `restrict` 关键字有什么作用？除了 `restrict` 关键字，请你顺带学习一下 `extern, static, const, volatile` 关键字的作用与用法，在后面的题目中可能会涉及。

函数参数也能是函数？

相信大家或许都使用或了解过 C++ 的 `std::sort()` 或 C 的 `qsort()`，在调用的时候他们传入的第 3 个参数是自定义比较函数，很神奇对吧？原来函数也能作为函数参数传递。在 C 语言里被称作**函数指针**，或者 C++ 里叫做回调函数。

请你补全一下代码，统计长度为 n 的数组 a 中大于/小于 k 的元素个数。

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

// 如果 x > y，则 *res ← *res + 1
void greater(int *res, int x, int y) {
    if (x > y)
        *res += 1;
}

// 如果 x < y，则 *res ← *res + 1
void less(int *res, int x, int y) {
    if (x < y)
        *res += 1;
}

// 统计长度为 n 的数组 a 中大于/小于 k 的元素个数，比较谓词通过函数参数传入
int count(/* 这里请设计函数参数，使之满足主函数中的调用格式 */) {
    //Your code
}

int main() {
    srand(time(NULL));
    int n, k;
    scanf("%d%d", &n, &k); // 输入数组的长度 n 和比较值 k
    int *a = malloc(n * sizeof(int));
    for (int i = 0; i < n; ++i)
        a[i] = rand() % 100; // 生成 100 以内的随机数
    printf("Array: ");
    for (int i = 0; i < n; ++i)
        printf("%d ", a[i]);
    puts("");
    // 大于 k 的数的个数
    printf("Numbers of greater than %d: %d\n", k, count(a, n, k,
greater));
    // 小于 k 的数的个数
```

```
    printf("Numbers of greater than %d: %d\n", k, count(a, n, k,
less));
    free(a);
    return 0;
}
```

运行示例如下：

```
$ ./a.out
20 52
Array: 60 25 82 63 27 1 84 53 96 83 11 82 13 80 81 0 47 48 43 75
Numbers of greater than 52: 11
Numbers of greater than 52: 9
```

请回答问题，并完成相应代码编写，并自行验证代码的正确性，至少构造 3 组测试数据，并在解题报告中附上相应的运行截图。同时完成一道题目后请及时 `git commit`

指针与结构体

指针是 C 语言最重要的概念之一，也是比较难理解的概念之一。通过指针在某种程度上它能把程序员想要传达的指令以更接近机器的方式表达，帮助我们更好的理解程序的运行原理。

请自行学习指针相关知识点：声明指针，`*` 与 `&` 运算符，动态内存分配，结构体与结构体指针。

动态数组

在 C99 标准以后，允许声明数组的大小可以是变量，而不必是常量。LRL 也想尝鲜 C99 的动态数组，遂写出了如下代码：

```
#include <stdio.h>

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

    int a[n][n];

    printf("sizeof(a) = %lu\n", sizeof(a));

    // Just do something to test
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < i; ++j)
            a[i][j] = i * j;
```

```
    long long sum = 0;
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            sum += a[i][j];

    printf("Sum of array a is %lld\n", sum);

    return 0;
}
```

在 `Ubuntu-23.10` 的**默认环境**下，LRL 运行上述程序，

```
$ ./dynamic-array
1000
sizeof(a) = 4000000
Sum of array a is 124583731214
```

没有达到 LRL 期望的结果，增大输入的数字 n 再来一次，

```
$ ./dynamic-array
5000
[1]    10199 segmentation fault  ./dynamic-array
```

OhOhOhOhOhOh，喜闻乐见的 `segmentation fault`，达到了 LRL 的期望结果。 请据此回答下面的问题。

Q1：为什么会出现 `segmentation fault`？

Q2：请修改上述代码，使之在 Linux/Windows 默认环境下， $n = 5000$ 的情况下，能够正常运行，并简单解释原因。

Q3：在完成 Q2 的基础上，`sizeof(a)` 的输出是多少？为什么？如果我想让 `sizeof(a)` 得到数组 a 的大小，该如何修改？

负数下标

众所周知，Python 列表支持负数下标，C 语言的数组下标只能是非负整数。LRL 不爽了，遂发明了负数下标（注：这里的负数下标和 Python 的负数下标意义不同）。

请补全下面的代码，构造一个支持负数下标的字符串数组 `negative_idx_str`，并在 `negative_idx_str[-52] ~ negative_idx_str[-20]` 内存储字符串 `Welcome to join Embedded Studio!`。

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

int main() {
    const char *str = "Welcome to join Embedded Studio!";

    //Your code

    for (int i = -52; negative_idx_str[i]; ++i)
        putchar(negative_idx_str[i]);
    puts("");
}
```

输出如下：

```
$ ./negative-idx

Welcome to join Embedded Studio!
```

膨胀的结构体

```c
struct {
    char *a;
    short b;
    double c;
    char d;
    float e;
    char f;
    long g;
    int h;
} rec;
```

在 x86-64 下，对于上面这个结构体的声明，回答下面 3 个问题：

Q1：这个结构中的所有的字段的字节偏移量是多少？

Q2：这个结构体的总大小是多少？为什么？

Q3：重新排列这个结构中的字段，以最小化浪费的空间，然后再给出重排过的结构字节偏移量和总的大小。

⭐多线程求和

CUDA 中提供了一类函数 `atomicCAS(dst, cmp, src)`，它的功能是：原子地判断是否相等，相等则写入，并读取旧值。`old = atomicCAS(dst, cmp, src)` 相当于：

```
old = *dst;
if (old == cmp)
  *dst = src;
return old;
```

为什么需要这么复杂的一个原子指令？因为 `atomicCAS` 的作用在于他可以用来实现任意 CUDA 没有提供的原子读-修改-写回指令。比如我们可以通过 `atomicCAS` 实现了整数原子加法 `atomicAdd` 同样的效果。

```
int atomicAdd(int *dst, int src) {
    int expect, old = *dst;
    do {
        expect = old; // 先保存旧值
        // 如果返回值 old 与旧值 expect 相同，则说明 *dst 没有被修改过，接受 expect + src 的值
        // 如果返回值 old 与旧值 expect 不相同，则说明在这之间有其它指令修改了 *dst 的值，expect + src 的值将没有意义了
        old = atomicCAS(dst, expect, expect + src);
    } while (expect != old);
    return old;
}
```

LRL 在学习 C 语言的时候，也仿照 CUDA 动手实现了一个丐版的 `atomicCAS`。很遗憾的是 C 语言不同于 C++，它不支持函数重载（function overloading），意味着 `int` 版的 `atomicCAS` 将无法支持 `float`，难道我真的得重写一遍 `float` 版的函数？。LRL 观察了 `atomicCAS` 代码后灵机一动，修改了 `atomicAdd`，成功实现了多线程浮点数求和。

说了这么多好像和指针没有关系？No，**你并不需要学习或了解 CUDA、多线程编程**（当然了解更好），你的任务就是**仅**修改 `atomicAdd`，**实现多线程浮点数求和**。下面是多线程整数求和的示例代码，仅供参考：

```
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
#include <string.h>
#include <threads.h>
#include <unistd.h>

#define N 100000000
#define THREAD_NUM 20
```

```c

static pthread_mutex_t mutex;
int n = N, sum, array[N];

int atomicCAS(int *dst, int cmp, int src) {
    pthread_mutex_lock(&mutex);
    int old = *dst;
    if (old == cmp) {
        *dst = src;
    }
    pthread_mutex_unlock(&mutex);
    return old;
}

int atomicAdd(int *dst, int src) {
    int expect, old = *dst;
    do {
        expect = old; // 先保存旧值
        // 如果返回值 old 与旧值 expect 相同，则说明 *dst 没有被修改过，接受 expect + src 的值
        // 如果返回值 old 与旧值 expect 不相同，则说明在这之间有其它指令修改了 *dst 的值，expect + src 的值将没有意义了
        old = atomicCAS(dst, expect, expect + src);
    } while (expect != old);
    return old;
}

void *calcSum(void *argv) {
    int id = atoi(argv), local_sum = 0;
    for (int i = id; i < n; i += THREAD_NUM) {
        local_sum += array[i];
    }
    atomicAdd(&sum, local_sum);
    pthread_exit(NULL);
}



int main() {
    srand(time(NULL));
    pthread_t thread[THREAD_NUM];

    pthread_mutex_init(&mutex, NULL);

    char argv[THREAD_NUM][5];
    memset(argv, 0, sizeof(argv));
```

```
54
55      int ans = 0;
56
57      for (int i = 0; i < n; ++i) {
58          array[i] = rand() % 10;
59          ans += array[i];
60      }
61
62      for (int i = 0; i < THREAD_NUM; ++i) {
63          sprintf(argv[i], "%d", i);
64          pthread_create(&thread[i], NULL, (void*)calcSum,
(void*)argv[i]);
65      }
66
67      for (int i = 0; i < THREAD_NUM; ++i) {
68          pthread_join(thread[i], NULL);
69      }
70
71      if (ans == sum) printf("The sum of array is %d\n", sum);
72      else printf("Wrong answer! The sum of array should be %d, but your
result is %d\n", ans, sum);
73
74      pthread_mutex_destroy(&mutex);
75
76      return 0;
77  }
```

请完善下面多线程浮点数求和版本的代码，以支持浮点数的多线程加法：

```
1  #include <math.h>
2  #include <stdio.h>
3  #include <pthread.h>
4  #include <stdlib.h>
5  #include <string.h>
6  #include <threads.h>
7  #include <unistd.h>
8
9  #define N 1000000
10 #define THREAD_NUM 50
11
12 #define min(x, y) ((x) < (y) ? (x) : (y))
13 #define max(x, y) ((x) > (y) ? (x) : (y))
14
15 static pthread_mutex_t mutex;
16 int n = N;
17 float sum = 0.0f, array[N];
```

```
18
19  int atomicCAS(int *dst, int cmp, int src) {
20      pthread_mutex_lock(&mutex);
21      int old = *dst;
22      if (old == cmp) {
23          *dst = src;
24      }
25      pthread_mutex_unlock(&mutex);
26      return old;
27  }
28
29  // 原子加法, *dst ← *dst + src, 返回相加前的旧值 *dst
30  float atomicAdd(float *dst, float src) {
31      // Your code
32  }
33
34  void *calcSum(void *argv) {
35      int id = atoi(argv);
36      float local_sum = 0.0f;
37      for (int i = id; i < n; i += THREAD_NUM) {
38          local_sum += array[i];
39      }
40      atomicAdd(&sum, local_sum);
41      pthread_exit(NULL);
42  }
43
44
45
46  int main() {
47      srand(time(NULL));
48      pthread_t thread[THREAD_NUM];
49
50      pthread_mutex_init(&mutex, NULL);
51
52      char argv[THREAD_NUM][5];
53      memset(argv, 0, sizeof(argv));
54
55      long double ans = 0.0;
56
57      for (int i = 0; i < n; ++i) {
58          array[i] = (float)rand() / RAND_MAX;
59          ans += array[i];
60      }
61
62      for (int i = 0; i < THREAD_NUM; ++i) {
63          sprintf(argv[i], "%d", i);
```

```
64          pthread_create(&thread[i], NULL, (void*)calcSum,
(void*)argv[i]);
65      }
66
67      for (int i = 0; i < THREAD_NUM; ++i) {
68          pthread_join(thread[i], NULL);
69      }
70
71      if (fabsl(sum - ans) < 1.0) printf("The sum of array is about
%.2Lf to %.2Lf\n", min(sum, ans), max(sum, ans));
72      else printf("Wrong answer! The sum of array should be %.2Lf, but
your result is %.2f\n", ans, sum);
73
74      pthread_mutex_destroy(&mutex);
75
76      return 0;
77  }
```

> Note: 你可能会问为什么要这么麻烦，我不能直接修改 `atomicCAS` 吗？确实不能，你可以认为 `atomicCAS` 类似于 C 提供的“库函数”，无法被直接修改。

> Hint: 本题有一定难度，如果没有思路可以私戳出题人，出题人会视情况给予提示

Q: 如果把 `ans` 变量的类型改为 `float`，你会发现求和结果与 `ans` 的偏差会增大（偏差会超出 1.0），那么哪个结果更接近实际答案呢？请说出你的理由。

多文件程序

一个软件项目往往包含多个源码文件，编译时需要将这些文件一起编译，生成一个可执行文件。每个源文件都包含程序的一部分功能或模块。这种分割代码的方式有助于提高代码的可维护性、可读性和可重用性，同时也使多人协作更加容易。

请自行学习多文件程序相关知识点：C 语言编译流程，预处理与宏，多文件源码组织，简单的 Makefile 编写。

线性代数与空间解析几何

相信各位大一同学正在学习 ~~美妙的~~ 线性代数与空间解析几何，在这道题目中，你需要实现简单的矩阵运算，构建一个简单的多文件项目，满足以下要求：

1. 在 `matrix.h` 中声明与矩阵相关的结构体和函数接口：

```
1  #define get(A, i, j) (A).a[i * (A).c + j]
```

```c

typedef struct {
    // r 行 c 列的矩阵
    int r, c;
    // 为了避免二级指针，a 指向一个 r * c 的一维数组，使用一维数组来模拟二维数组
    int *a;
} Mat;

void matrixPrint(const Mat *A, const char *str);

Mat* matrixAdd(const Mat *A, const Mat *B);

Mat* matrixMul(const Mat *A, const Mat *B);

Mat* matrixTranspose(const Mat *A);
```

2. 在 `matrix.c` 中实现上述函数接口
3. 在 `test.c` 中至少包含如下测试代码，`test()` 中的 `A, B` 来源于 `main.c` 中的定义：

```c
void test() {
    matrixPrint(&A, "A");
    matrixPrint(&B, "B");
    Mat *C = matrixAdd(&A, &A);
    matrixPrint(C, "C");
    Mat *D = matrixMul(&A, &B);
    matrixPrint(D, "D");
    Mat *E = matrixTranspose(&A);
    matrixPrint(E, "E");
    free(C->a);
    free(C);
    free(D->a);
    free(D);
    free(E->a);
    free(E);
}
```

4. 在 `main.c` 中至少包含如下代码，`main.c` 中会调用 `test()` 来测试验证，

```c
Mat A, B;

int main() {
    A.r = 2, A.c = 3;
    A.a = malloc(A.c * A.r * sizeof(int));
```

```
    int cnt = 0;
    for (int i = 0; i < A.r; ++i)
        for (int j = 0; j < A.c; ++j)
            get(A, i, j) = ++cnt;

    B.r = 3, B.c = 2;
    B.a = malloc(B.c * B.r * sizeof(int));
    cnt = 0;
    for (int i = 0; i < B.r; ++i)
        for (int j = 0; j < B.c; ++j)
            get(B, i, j) = ++cnt;

    test();

    free(A.a);
    free(B.a);

    return 0;
}
```

5. 完善上述提到的所有代码，使之能够正常通过编译并运行（注：你可以添加新的头文件）。

期望运行结果如下：

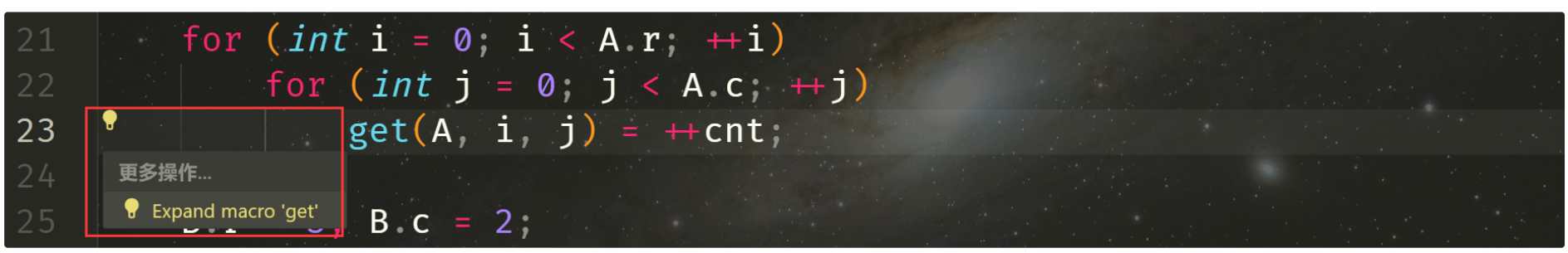
```
$ ./main
A.r = 2, A.c = 3
1 2 3
4 5 6

B.r = 3, B.c = 2
1 2
3 4
5 6

C.r = 2, C.c = 3
2 4 6
8 10 12

D.r = 2, D.c = 2
22 28
49 64

E.r = 3, E.c = 2
1 4
2 5
3 6
```

预处理与宏

这是一道简单的大水题。

Subtestk1

借助代码编辑器或者 IDE，我们可以很容易的展开宏，看到宏替换后的代码。比如上面一道题目 `main.c` 中的代码：



宏替换后的代码如下：

```
for (int i = 0; i < A.r; ++i)
    for (int j = 0; j < A.c; ++j)
      (A).a[i * (A).c + j] = ++cnt;
```

Q：编译器在预处理阶段做了什么？试着不用编辑器或者 IDE，借助 `gcc` 观察一下预处理后的代码。

Subtestk2

请在**不修改**下面的代码情况下，分两次编译并运行下面的代码，尝试输出 `You got this message.` 和 `You got another message.` 。

```c
#include <stdio.h>

int main() {

#if defined TEST && defined LOCAL
    printf("You got this message.\n");
#elif defined TEST || defined LOCAL
    printf("You got another message.\n");
#else
    printf("Don't print me\n");
#endif


    return 0;
}
```

Makefile

在完成了“线性代数与空间解析几何”题目的情况下，请为该多文件程序编写简单的 makefile。要求如下：

- `make` ：构建项目程序（Release 版本，适当优化，不需要添加调试选项）
- `make run` ：运行程序
- `make debug` ：构建调试版本的程序（Debug 版本，需要添加调试选项）
- `make clean` ：清除项目构建产生的所有文件

附录

如何完成题目

请自行借助互联网搜索引擎、B 站或课本等方式学习 C 语言相关知识点。在完成题目的过程中，请使用 markdown 撰写解题报告，记录下你的解题思路和过程，必要时在解题报告中回答题目的问题。对于编码题目，请在完成一道题目后自行验证代码的正确性，并在解题报告中附上相应的运行截图。**你的所有代码和 md 文件都应该放在一个本地 git 仓库里，完成一道题目或撰写更新了 md 文件后，请及时后 `git commit` 以记录下你的学习足迹。**

题目不一定按照难度排序。我们不要求你一定要完成所有的题目，而是更关注你的学习态度。关于题目有任何问题可以 QQ 私戳出题人（525687841）。完成题目后尽早提交，私戳出题人及时获取反馈评价，如有问题可以再完善修改。在 **2023 年 10 月 8 日晚 23:59** 截止日期前，请将你的本地 git 仓库打包（md 导出为 PDF）发送至出题人邮箱： `lrl@lrl52.top` 。

参考学习资源

提问的智慧

Markdown

廖雪峰 Git 教程

C 语言入门教程

浙江大学翁恺教你 C 语言程序设计

《C Primer Plus 第6版》

> Note：Git 主要用于记录你的学习足迹，为了快速上手，关于 Git 你只需要了解 `git init` ， `git add` 和 `git commit` 命令即可。