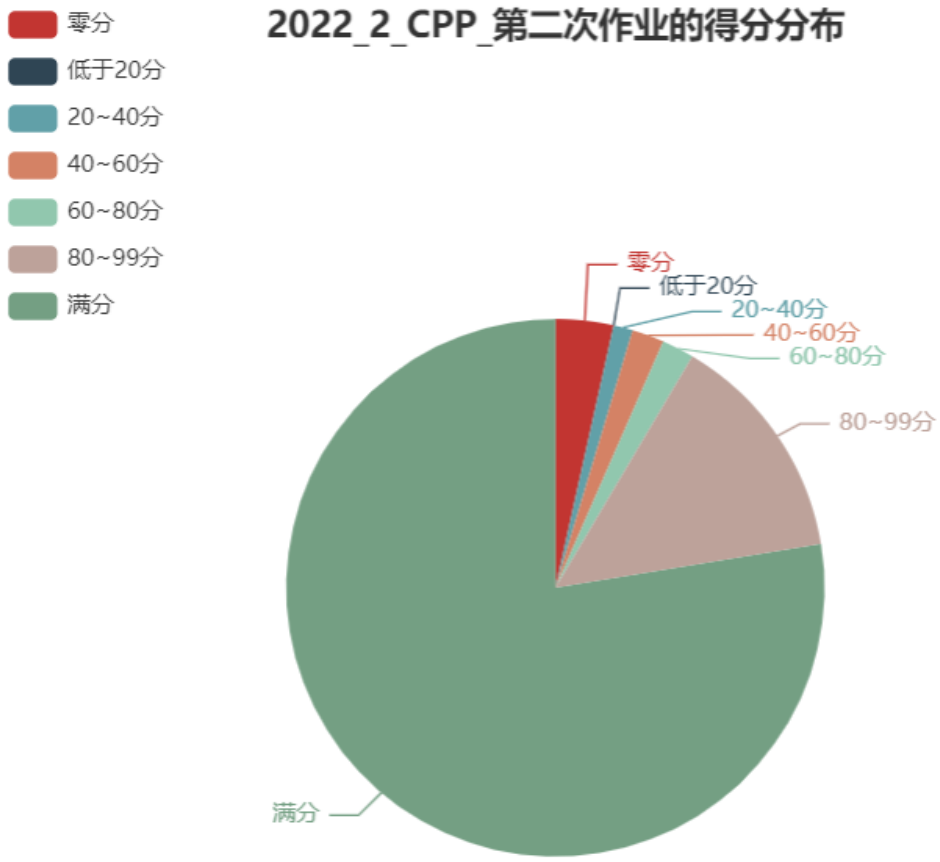
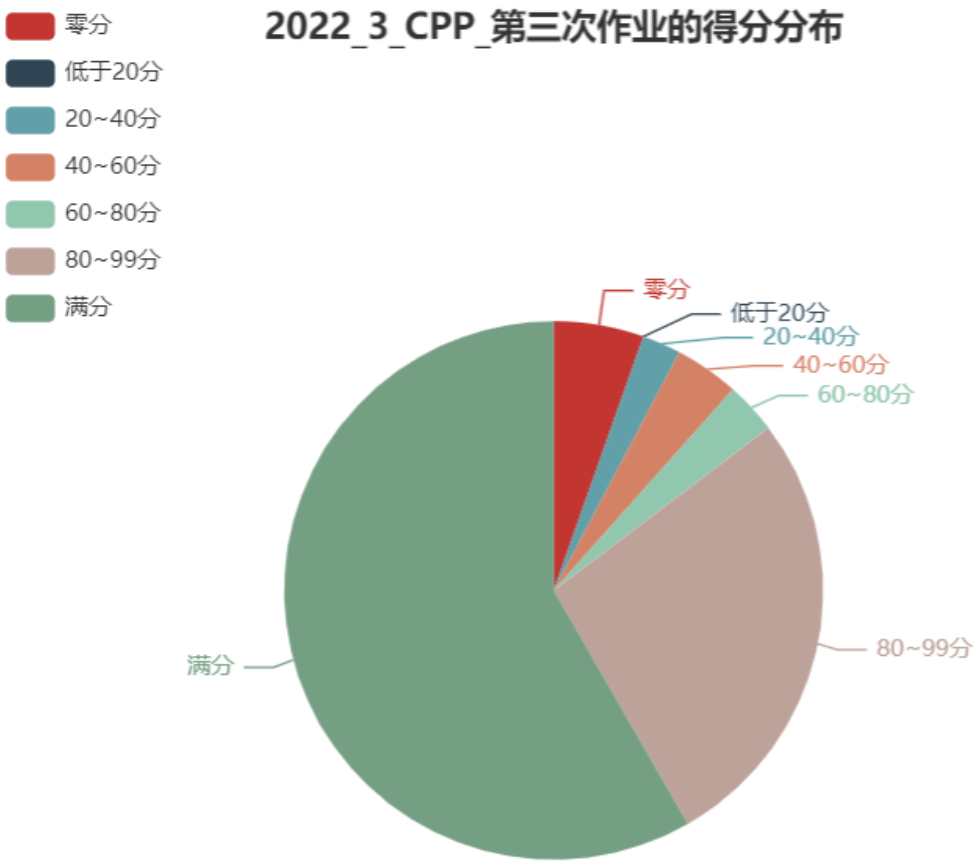
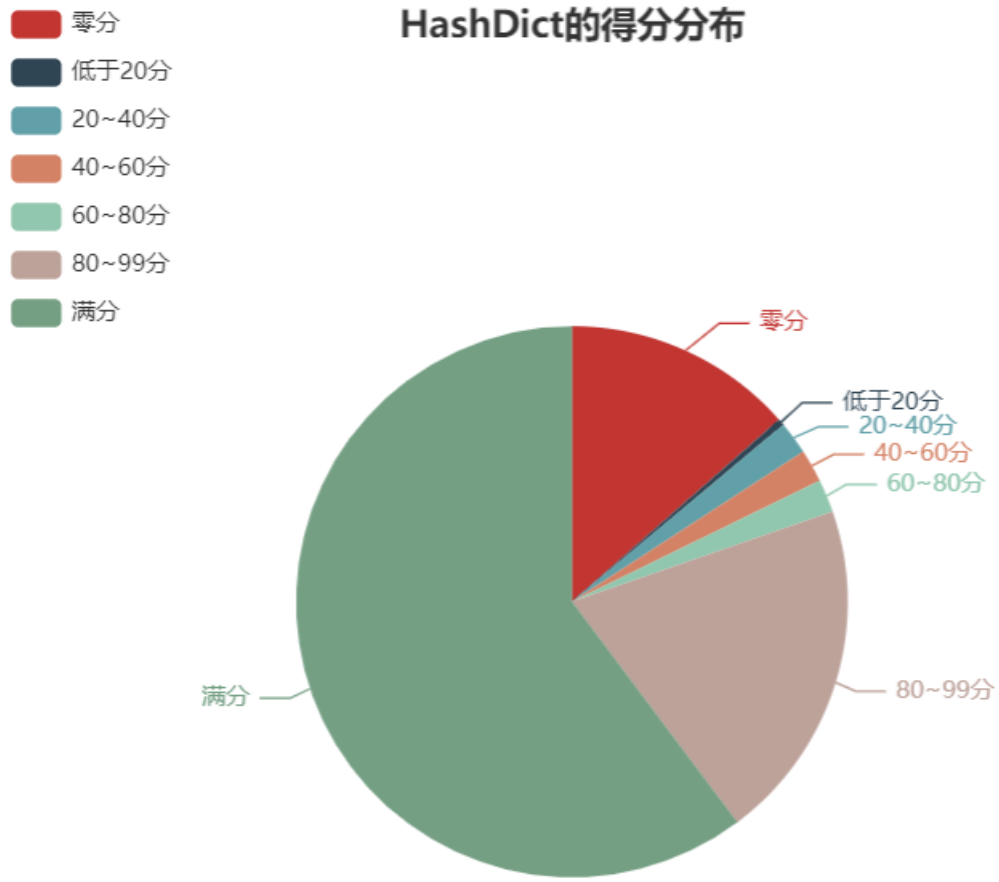
第二次习题课

做题情况


零分
低于20分
20~40分
40~60分
60~80分
80~99分
满分
2022_2_CPP_第二次作业的得分分布
零分
低于20分
20~40分
40~60分
60~80分
80~99分
满分


零分
低于20分
20~40分
40~60分
60~80分
80~99分
满分
2022_3_CPP_第三次作业的得分分布
零分
低于20分
20~40分
40~60分
60~80分
80~99分
满分

做题情况 (cont'd)


零分
低于20分
20~40分
40~60分
60~80分
80~99分
满分
HashDict的得分分布
零分
低于20分
20~40分
40~60分
60~80分
80~99分
满分

大纲

- 题解
- I/O
- 其他问题

大纲

- 题解
- I/O
- 其他问题

参考代码 Git 仓库

- https://git.nju.edu.cn/rhanqtl/2022-cpp-hw-3

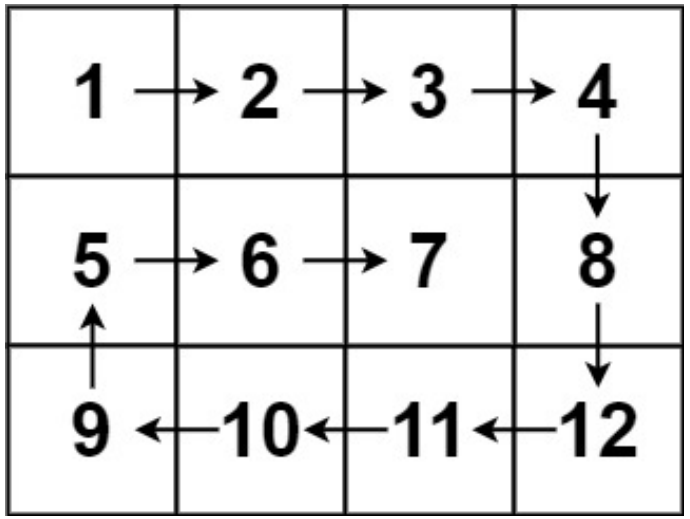
HashDict

- hash_code 是根据 key 计算出来的，而 key 是不会变的，所以 hash_code 也不会变，因此只需要在创建时计算一次
- HashDict 的结构是数组 + 有序链表
  - 链表可以是双向也可以是单向
- 注意点
  - 可能存在连续多次 rehash 的情况
  - 扩容的时候可以直接转移链表节点，不需要重新分配

BF 解释器

- 对 [和] 的处理，三种方式
  - 运行时使用栈找到匹配的另一半
  - 运行时使用计数的方式找到匹配的另一半
  - **运行前使用栈构建好跳转表**
- 读取 BF 程序的输入

螺旋矩阵



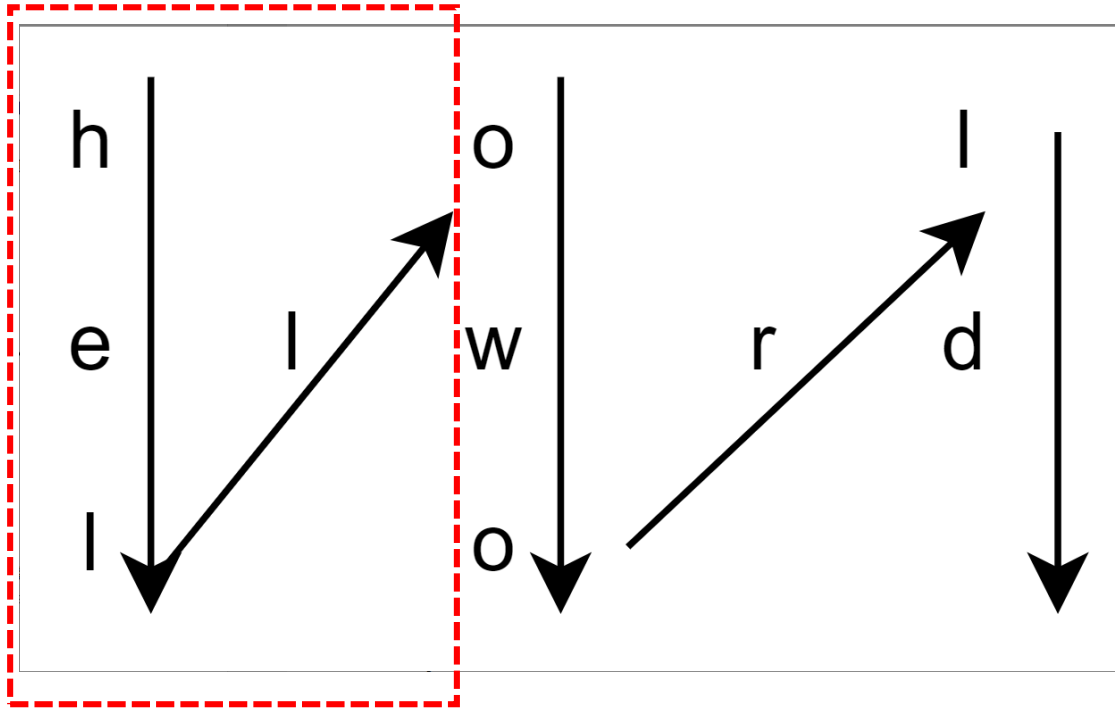
- 从外到里一圈一圈地处理，用四个变量划定一个范围
- 注意边界情况
  - 只有一行或一列
  - 只有两行或两列

旋转矩阵

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

```
int temp = mat.data[i][j];
mat.data[i][j] = mat.data[n - j - 1][i];
mat.data[n - j - 1][i] = mat.data[n - i - 1][n - j - 1];
mat.data[n - i - 1][n - j - 1] = mat.data[j][n - i - 1];
mat.data[j][n - i - 1] = temp;
```

Z 形变换

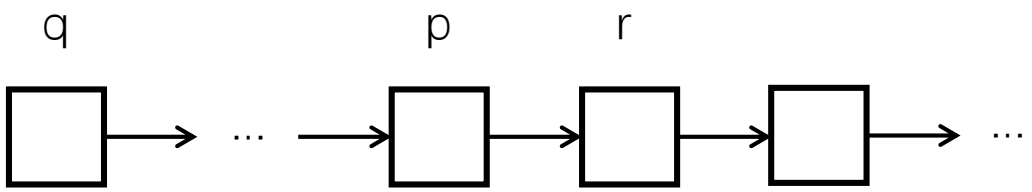


- 如果行数为 N，那么一组有 2N – 2 个字母
- 第一行是每组第一个
- 第二行是每组第二个 + 每组倒数第一个
- 第三行是每组第三个 + 每组倒数第二个（如果有）
- ...
- 依此类推

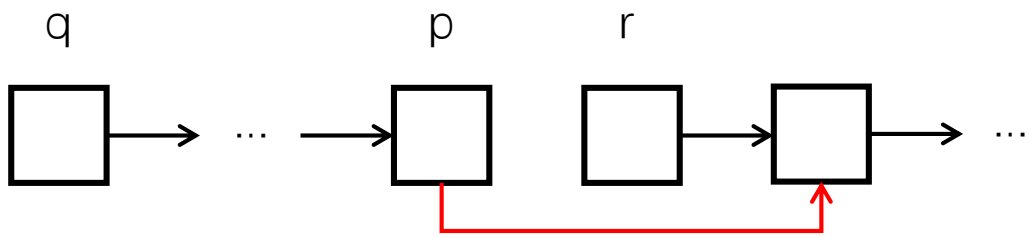
翻转链表

- 三个指针
  - 一个始终指向最初的头节点，记为 p
  - 一个始终指向翻转后的头节点，记为 q（初始值为原头节点）
  - 一个始终指向下一个待移动的节点，记为 r（初始值为 p->next）

```
while (r != nullptr) {
    p->next = r->next;
    r->next = q;
    q = r;
    r = p->next;
}
```

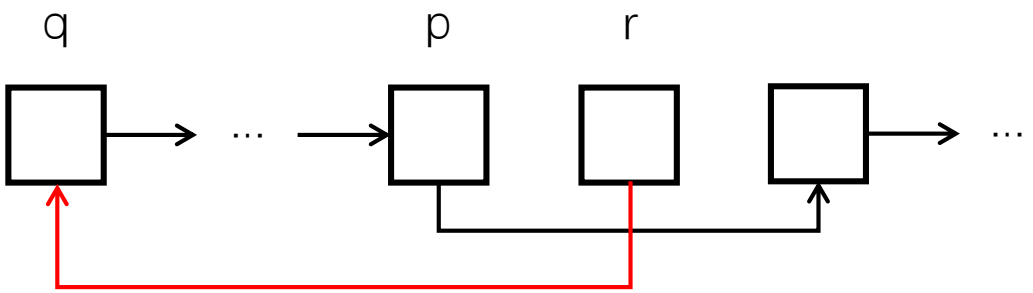


1

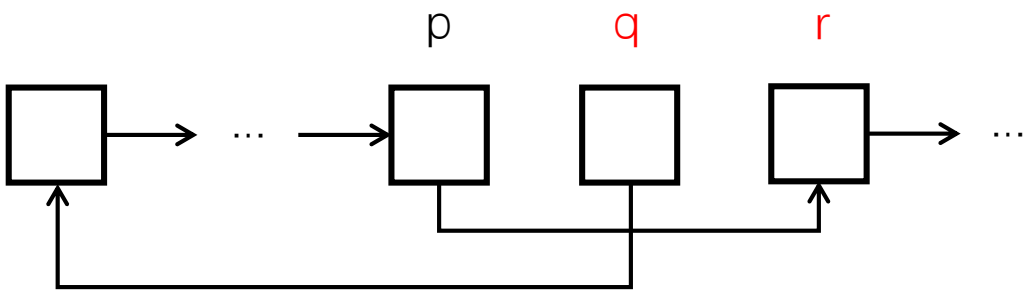


2

3



4



sort

- 使用 C++ 标准库的 sort 函数

```
bool compare(const char *lhs, const char *rhs) {
  return std::strcmp(lhs, rhs) < 0;
}
```

```
char *ss[] = {“1”, “2”, “10”};
// 第一个参数为数组起始位置
// 第二个参数为数组最后一个元素的下一个位置
// 第三个元素为比较函数
std::sort(ss, ss + 3, compare);
// 现在 ss 内容为 {“1”, “10”, “2”}
// std::sort 默认使用 < 比较
```

sort (cont'd)

- 忽略大小写

```
bool compare(const std::string &lhs,
             const std::string &rhs) {
  std::string lhs_lower, rhs_lower;
  // xxx.begin() 和 xxx.end() 划定一个左闭右开的区间
  transform(lhs.begin(), lhs.end(),
      std::back_inserter(lhs_lower.begin()),
      std::tolower);
  transform(rhs.begin(), rhs.end(),
      std::back_inserter(rhs_lower.begin()),
      std::tolower);
  return (lhs_lower < rhs_lower)
         || (lhs_lower == rhs_lower && lhs < rhs);
}
```

sort (cont'd)

- 只考虑字母、数字和空格

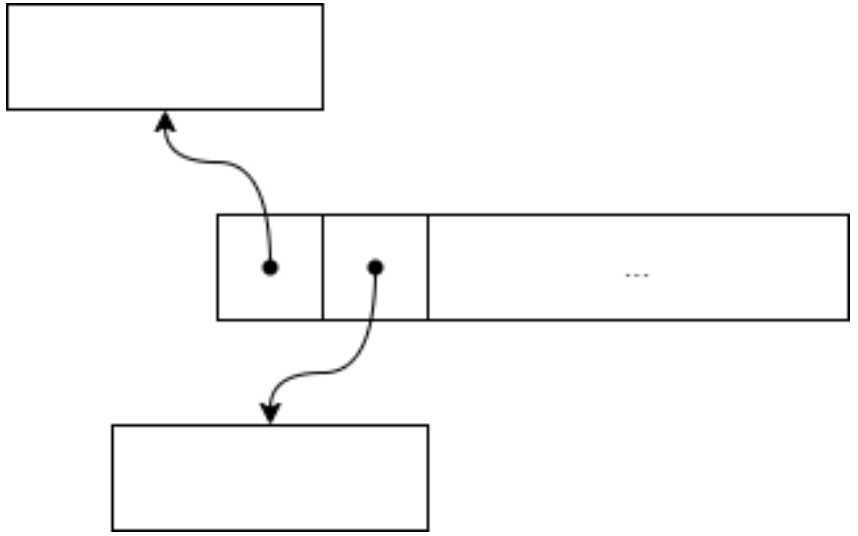
```
bool is_alpha_num_space(char c) {
  return isalpha(c) || isdigit(c) || c == ' ';
}

bool compare(const std::string &lhs,
             const std::string &rhs) {
  std::string lhs_filtered, rhs_filtered;
  // std::back_inserter 将赋值替换成 push_back 的调用
  std::copy_if(lhs.begin(), lhs.end(), std::back_inserter(lhs_filtered),
      is_alpha_num_space);
  std::copy_if(rhs.begin(), rhs.end(), std::back_inserter(rhs_filtered),
      is_alpha_num_space);
  return lhs_filtered < rhs_filtered;
}
```

ArrayList

- 扩容操作
  1. 计算新的大小 new_cap = old_cap + (old_cap >> 1)
  2. 创建新的数组
  3. 复制元素
  4. 回收旧数组
  5. 更新字段
- 命名
  - 推荐使用 len 表示元素个数，cap 表示数组大小
  - 如果只有一个“大小”，可以使用 size

Deque

- 双向链表
- 分块



矩阵连乘

- 两个思路：
  - 一次性读入所有矩阵，按顺序运算
  - 先读入一个，然后循环中每次读入一个进行运算

大纲

- 题解
- I/O
- 其他问题

读入未知数量的数据

- int、char 和 std::string
  - 注意对于 char 和 std::string，cin >> x 会跳过空白符

```
int i;
while (cin >> i) { /* … 处理 */ }

char c;
while (cin >> c) { /* … 处理 */ }

std::string s;
while (cin >> s) { /* … 处理 */ }
```

读入单个 char，包含空白符

- cin.get

```
char c;
// 返回值为 std::istream 对象
cin.get(c);
```

读入命令 + 可选参数

- 使用 scanf（如果参数为字符串则失效）

```
char cmd[16] = {0};
int operand;
scanf("%s %d", cmd, &operand);
```

- 先读命令，再判断是否要读入参数

```
std::string cmd;
cin >> cmd;
if      (cmd == "xxx") { int operand; cin >> operand; }
else if (cmd == "yyy") { /* … */ }
```

读入一整行

- getline

```
std::string line;
getline(std::cin, line);
```

  - 默认使用 \n 作为结束标志
  - 这两次作业中存在 \r\n 作为换行的情况
  - Windows 上使用 \r\n 作为换行符，类 UNIX 上使用 \n 作为换行符

读入矩阵

```
std::size_t n_rows, n_cols;
std::cin >> n_rows >> n_cols;
int **mat = /* … 分配动态内存 */;
for (int i = 0; i < n_rows; i++) {
  std::copy_n(
      std::istream_iterator<int>(std::cin), n_cols, mat[i]);
}
```

大纲

- 题解
- I/O
- 其他问题

其他问题

- malloc vs. new
- 内存泄露
- 变长数组
- std::string 的比较
- NULL vs. nullptr

动态内存操作

- malloc / calloc / realloc vs. new
  - malloc / calloc / realloc **不会**调用构造函数，除非你想自己控制调用构造函数的时机（不过应该没有这种场景）

```cpp
#include <iostream>
#include <string>
#include <cstdlib>
using namespace std;

struct A {
    int i;
    A(): i(0x3f3f3f3f) {}
};

int main() {
    A *arr = (A*)malloc(3 * sizeof(A));
    for (int i = 0; i < 3; i++)
        printf("%x\n", arr[i].i);
    // 输出: 0 0 0
    free(arr);

    arr = new A[3];
    for (int i = 0; i < 3; i++)
        printf("%#x\n", arr[i].i);
    // 输出: 0x3f3f3f3f 0x3f3f3f3f 0x3f3f3f3f
    delete[] arr;
}
```

内存泄漏

- 记得释放动态分配的内存!

变长数组

```
int n;
cin >> n;
int arr[n];
```

- 属于 C99 标准，C++ 中**不推荐**使用

std::string 的比较

- 可以使用 <、<=、>、>=、==、!=
- 成员函数 compare 的返回值**只规定了符号**，判断时用 <0、>0、==0，不要用特定的值，例如 -1

NULL vs. nullptr

- NULL 可能如下定义，也就是说，可以当作一个整数

```
#define NULL 0
```

```
void f(int) {}
void f(void *) {}

int main() { f(NULL); }
```

```
 In function 'int main()':
10:11: error: call of overloaded 'f(NULL)' is ambiguous
10:11: note: candidates are:
6:6: note: void f(int)
7:6: note: void f(void*)
```

- nullptr 的类型为 std::nullptr_t，只能转换成指针