Student Name: 戈天齐
Student ID: 2022141460202

C++: 面向对象程序设计
Assignment 1

1 Makefile&Cmake

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.18)

project(stuinfo)

add_executable(stuinfo stuinfo.h main.cpp  stuinfo.cpp)
```

stuinfo.h

```
//
// Created by 77 on 2023/3/9.
//
#ifndef STUINFO_STUINFO_H
#define STUINFO_STUINFO_H
#include <cstring>
#include <iostream>

struct stuinfo
{
    char name[20];
    double score[3];
    double ave;
};
void inputstu(stuinfo stu[] , int n);

void showstu(stuinfo stu[] , int n);

void sortstu(stuinfo stu[] , int n);

bool findstu(stuinfo stu[] , int n, char ch[]) ;

#endif //STUINFO_STUINFO_H
```

stuinfo.cpp

```cpp
//
// Created by 77 on 2023/3/10.
//

#include "stuinfo.h"

void inputstu(stuinfo stu[] , int n)
{
    std::cout << "Please input the name and score of the student:" << std::endl;
    for(int i = 0; i < n; i++){
        std::cin >> stu[i].name >> stu[i].score[0] >> stu[i].score[1] >> stu[i].score[2];
        stu[i].ave = (stu[i].score[0] + stu[i].score[1] + stu[i].score[2]) / 3;
    }
}

void showstu(stuinfo stu[] , int n)
{
    for(int i = 0 ; i < n ; i++){
        std::cout << stu[i].name << " " << stu[i].score[0] << " " << stu[i].score[1] << " " << stu[i].
            score[2] << " " << stu[i].ave << std::endl;
    }
}

void sortstu(stuinfo stu[] , int n)
{
    for(int i = 0 ; i < n ; i++){
        for(int j = i + 1 ; j < n ; j++){
            if(stu[i].ave < stu[j].ave){
                stuinfo temp = stu[i];
                stu[i] = stu[j];
                stu[j] = temp;
            }
        }
    }
}
bool findstu(stuinfo stu[] , int n, char ch[])
{
    for(int i = 0 ; i < n ; i++){
        if(strcmp(stu[i].name , ch) == 0){
            return true;
        }
    }
    return false;
}
```

main.cpp

```cpp
#include "stuinfo.h"
#include <iostream>

int main() {
    int n;
    std::cout << "Please input the number of students:" << std::endl;
    std::cin >> n;
    stuinfo stu[n];
    inputstu( stu , n);
    sortstu( stu , n);
    std::cout << "The sorted student information is:" << std::endl;
    showstu( stu , n);
    std::cout << "Please input the name of the student you want to find:" << std::endl;
    char ch[20];
    std::cin >> ch;
    if(findstu( stu , n , ch)){
        std::cout << "Yes" << std::endl;
    } else{
        std::cout << "No" << std::endl;
    }
    return 0;
}
```

运行结结果如下：

```
Please input the number of students:
3
Please input the name and score of the student:
a 1.1 1.2 1.5
b 3.1 5.2 4.1
c 2.8 0 0.9
The sorted student information is:
b 3.1 5.2 4.1 4.13333
a 1.1 1.2 1.5 1.26667
c 2.8 0 0.9 1.23333
Please input the name of the student you want to find:
c
Yes

进程已结束,退出代码0
```

2 Types

2.1 问答题

2.1.1 static 用法和作用

- 定义静态变量或静态函数
- 定义静态局部变量函数或静态局部函数，使变量或者函数局限在文件中

2.1.2 什么是隐式转换，如何消除隐式转换？

- 什么是隐式转换？

  隐式转换是指编译器自动把一种数据类型转换成另一种数据类型的过程，不需要程序员显式地进行类型转换操作，例如我们把一个浮点数赋值给一个整形，编译器会自动帮我们去除小数部分，保留下整数部分

- 如何消除隐式转换？
- 直接显式类型转换：我们可以利用 c++ 中的强制转换操作符进行显式转换
- 在类的构造函数钱加上关键字 explicit，来禁止编译器进行隐形转换
- 利用类型转换函数：在类中定义类型转换函数，可以将类的对象转换为其他类型的值

2.2 程序解释提

2.2.1

```
sum = -1825831516
fsum = 1.23457e+09
sum10x = 1
mul10x = 1
(sum10x == 1) is 0
(mul10x == 1) is 1

进程已结束,退出代码0
```

- sum = -1825831516

  1234567890 + 1234567890 = 2469135780，但是超过了 int 型变量的范围，所以导致程序输出的结果是 sum = -1825831516

- fsum = 1.23457e+09

  foat 大概能精确到小数点后 6 7 位，所以无论是 1234567890.0 还是 1234567891.0，float 都只能精确到 1.234567e+9

- (fsum == f1) is 1

  由于浮点数的特殊性，用” == “去判断两个浮点数是否完全相等，是存在误差的，所以才会造成 1234567890.0f = 1234567891.0f。通常情况下，我们如果要去判断两个浮点数是否相等，通常采用（f1 - f2）< 1e-9 的办法

- sum10x = 1 mul10x = 1 (sum10x == 1) is 0 (mul10x == 1) is 1

  虽然我们给 f 赋值为 0.1f，但是计算机并不能精确的表示 0.1，而是用一个比较接近的数来代替 0.1，同时由于浮点数的尾数位数有限，所以浮点数的表示和计算过程中存在误差。

  又因为 sum10x 的计算过程在 10 的累加过程中误差累加一直且变大，而 mul10x 的计算过程只有一步，误差总的来说是比 sum10x 小的，最终导致，sum10x 结果偏大，而 mul10x 在 1.0 的误差范围内

2.2.2

```
f1 = 1.000000
f2 = 1.000000
f1 != f2
f2 - f1 = 1.19209e-07

进程已结束,退出代码0
```

- 我在源代码的基础上增加了 f 2 - f 1 的输出，结果是如上图

- f 2 是由 10 个 0.1f 相加得到的，虽然输出显示六位是 1.000000，但是实际计算过程中存在误差，第七位开始出现误差，所以 f 2 != f 1

2.2.3

```
a = 41
b = 40
c = 2
d = 2.875

进程已结束,退出代码136
```

- a = 19.99 + 21.99

  该计算过程会先实现 19.99 + 21.99 = 41.98 然后将 41.98 赋值给 int 型变量时发生了截断（隐式转换），所以 a = 41

- b = (int)19.99 + (int)21.99

  该过程会先实现强制类型转换，b = 19 + 21 = 40

- c = 23 / 8

  int 型 / int 型 = int 型，23 / 8 = 2，所以 c = 2（该过程存在隐式转换）

- d = 23 / 8.0

  23 / 8.0 = 23.0 / 8.0 该过程存在隐式转换，双目运算中有一个变量是浮点型时，编译器会自动把另一个也变成浮点型

3 Structs

3.1 结构体对齐问题

- 在给出的两段代码中，alignas 生效和失效取决于 alignas 指定的对齐方式的大小，当指定对齐值大于结构体成员的最大长度时，alignas 成立，反之，则不成立。
- 当然除此之外还有几种情况会导致 alignas 失效，情况如下：
  - alignas 的指定对齐值不是 2 的幂
  - 指定对齐值大于类型的本身的大小
  - 指定对齐值大于了编译器的最大对齐方式

3.2 Exercise

2DGeo.h

```
//
// Created by 77 on 2023/3/14.
//

#ifndef ASMGNT_4_2_2DGEO_H
#define ASMGNT_4_2_2DGEO_H
#include <cmath>
struct Point { double x, y; };//点
using Vec = Point;//向量
Vec getLine(Point A,Point B );
struct Line { Point P; Vec v; }; // 直线（点向式）
struct Seg { Point A, B; }; // 线段（存两个端点）
struct Circle { Point O; double r; }; // 圆（存圆心和半径）
const Point O = {0, 0}; // 原点
const Line Ox = {O, {1, 0}}, Oy = {O, {0, 1}}; // 坐标轴
const double PI = acos(-1), EPS = 1e-9; //PI 与偏差值
std::pair<bool, Point> barycenter(const Point &A, const Point &B, const Point &C);// 三角形重心
std::pair<bool, Point> circumcenter(const Point &A, const Point &B, const Point &C);// 三角形外心
std::pair<bool, Point> incenter(Point A, Point B, Point C);
std::pair<bool, Point> orthocenter(const Point &A, const Point &B, const Point &C);// 三角形垂心
Line getPerp_bisector(Point A, Point B);
Line getPlumbline(Point p1,Point p2,Point p3);
Point getIntersection(Line A ,Line B);
double length(Point p1,Point p2);
#endif //ASMGNT_4_2_2DGEO_H
```

2DGeo.h

```
//
// Created by 77 on 2023/3/14.
//
#include <iostream>
#include <cmath>
#include "2DGeo.h"

double length(Point p1,Point p2)
{
    return sqrt((p1.x - p2.x)*(p1.x - p2.x)+(p1.y - p2.y)*(p1.y - p2.y));
}

```

```
Vec getLine(Point A,Point B)
{
    Vec AB;
    AB.x = A.x - B.x;
    AB.y = A.y - B.y;
    return AB;
}
Line getPerp_bisector(Point A, Point C)
{
    Point D; //ac线段上
    Line  PD;
    Vec AC = getLine(A,C);
    D.x = (A.x + C.x)/2;
    D.y = (A.y + C.y)/2;

    PD.P = D;
    PD.v.x = -(AC.y);
    PD.v.y = AC.x;
    return PD;
}
Line getPlumbline(Point p1,Point p2,Point p3)
{
    Line line;
    line.P = p1;
    line.v.x = -(p2.y - p3.y);
    line.v.y = p2.x - p3.x;
    return line;
}
Point getIntersection(Line A ,Line B)
{
    if(A.v.x * B.v.y == A.v.y * B.v.x){
        return 0;
    }
    Point P;
    P.x = (A.P.x * A.v.y * B.v.x - A.P.y * A.v.x * B.v.x - B.P.x * B.v.y* A.v.x + B.P.y * B.v.x * A.v.x)
         / (A.v.y * B.v.x - A.v.x * B.v.y);
    P.y = -(A.P.x * A.v.y * B.v.y - A.P.y * A.v.x * B.v.y - B.P.x * B.v.y * A.v.y + B.P.y * B.v.x * A.v.
        y) / (A.v.x * B.v.y - A.v.y * B.v.x);
    return P;
}
// 三角形重心
std::pair<bool, Point> barycenter(const Point &A, const Point &B, const Point &C)
{
    Vec AB = getLine(A,B);
    Vec AC = getLine(A,C);
    if(AB.x * AC.y == AB.y * AC.x){
```

```
        return std::make_pair(false,O);
    }
    else{
        Point G;
        G.x = (A.x + B.x + C.x) / 3;
        G.y = (A.y + B.y + C.y) / 3;
        return std::make_pair(true,G);
    }
}
// 三角形外心
std::pair<bool, Point> circumcenter(const Point &A, const Point &B, const Point &C)
{
    Vec AB = getLine(A,B);
    Vec AC = getLine(A,C);
    if(AB.x * AC.y == AB.y * AC.x){
        return std::make_pair(false,O);
    }
    else
        return std::make_pair(true,getIntersection(getPerp_bisector(A,B),getPerp_bisector(A,C)));
}
// 三角形内心
std::pair<bool, Point> incenter(Point A, Point B, Point C)
{
    Vec AB = getLine(A,B);
    Vec AC = getLine(A,C);
    if(AB.x * AC.y == AB.y * AC.x){
        return std::make_pair(false,O);
    }
    else {
        double a = length(B,C);
        double b = length(A,C);
        double c = length(A,B);
        Point P{(a*A.x + b*B.x + c*C.x)/(a+b+c),(a*A.y + b*B.y + c*C.y)/(a+b+c)};
        return std::make_pair(true,P);
    }
}
// 三角形垂心
std::pair<bool, Point> orthocenter(const Point &A, const Point &B, const Point &C)
{
    Vec AB = getLine(A,B);
    Vec AC = getLine(A,C);
    if(AB.x * AC.y == AB.y * AC.x){
        return std::make_pair(false,O);
    }
    else {
        Line AP = getPlumbline(A,B,C);
```

```
        Line BP = getPlumbline(B,A,C);
        return std::make_pair(true, getIntersection(AP, BP));
    }
}
```

2DGeo.h

```
#include <iostream>
#include "2DGeo.h"
using namespace std;
int main() {
    Point A,B,C;
    Vec AB = getLine(A,B);
    Vec AC = getLine(A,C);
    Vec BC = getLine(B,C);
    cout << "Please enter the coordinates of the three points of the triangle" << endl;
    cin >> A.x >> A.y >> B.x >> B.y >> C.x >> C.y;
    //重心
    std::pair<bool, Point> G = barycenter(A,B,C);
    if(G.first){
        cout << "The barycenter of the triangle is (" << G.second.x << "," << G.second.y << ")" << endl;
    }
    else{
        cout << "The three points are on the same line" << endl;
    }
    //外心
    std::pair<bool, Point> H = circumcenter(A,B,C);
    if(H.first){
        cout << "The circumcenter of the triangle is (" << H.second.x << "," << H.second.y << ")" <<
            endl;
    }
    else{
        cout << "The three points are on the same line" << endl;
    }
    //内心
    std::pair<bool, Point> I = incenter(A,B,C);
    if(I.first){
        cout << "The incenter of the triangle is (" << I.second.x << "," << I.second.y << ")" << endl;
    }
    else{
        cout << "The three points are on the same line" << endl;
    }
    //垂心
    std::pair<bool, Point> J = orthocenter(A,B,C);
    if(J.first){
```

```
        cout << "The orthocenter of the triangle is (" << J.second.x << "," << J.second.y << ")" << endl
            ;
    }
    else{
        cout << "The three points are on the same line" << endl;
    }
    return 0;
}
```

上面给出了该项目的所有基础文件，下面的操作则是完成将函数编译到共享库“lib.so”中

1. 将包含函数定义的文件 2DGeo.cpp 编译成一个名为 lib.so 的共享库文件
2. 将 main.cpp 与共享库 lib.so 链接，并生成一个可执行文件
3. 把 lib.so 复制到目录/usr/lib 下，并检查复制是否成功，运行可执行文件时，系统会在该目录寻找共享库
4. 运行可执行文件

```
root@fa571cbced78:/ws/Assighment_4.2# g++ -shared -fPIC 2DGeo.cpp -o lib.so
root@fa571cbced78:/ws/Assighment_4.2# g++ -L. -l:lib.so -o libtest.out
/usr/bin/ld: /usr/lib/x86_64-linux-gnu/crt1.o: in function `_start':
(.text+0x20): undefined reference to `main'
collect2: error: ld returned 1 exit status
root@fa571cbced78:/ws/Assighment_4.2# g++ main.cpp -L. -l:lib.so -o libtest.out
root@fa571cbced78:/ws/Assighment_4.2# cp lib.so /usr/lib
root@fa571cbced78:/ws/Assighment_4.2# cd /usr/lib
root@fa571cbced78:/usr/lib# ls
X11          dpkg            git-core  lib.so   os-release            python3.9    sudo        udev
apt          emacsen-common  gnupg     locale   pkg-config.multiarch  sasl2        sysctl.d    valgrind
bfd-plugins  file            gnupg2    mime     pkgconfig             sftp-server  systemd     x86_64-linux-gnu
compat-ld    gcc             gold-ld   openssh  python3               ssl          tmpfiles.d
root@fa571cbced78:/usr/lib# cd /ws/Assighment_4.2
root@fa571cbced78:/ws/Assighment_4.2# ./libtest.out
Please enter the coordinates of the three points of the triangle
0 0 3 0 0 4
The barycenter of the triangle is (1,1.33333)
The circumcenter of the triangle is (1.5,2)
The incenter of the triangle is (1,1)
The orthocenter of the triangle is (0,0)
root@fa571cbced78:/ws/Assighment_4.2#
```

4 C++ 动态内存申请

4.1 C++ 的内存分区

- static int a 全局/静态存储区 a 是一个静态变量，存储在全局/静态存储区
- auto b = 0 全局/静态存储区 b 在函数外声明，属于全局变量，存储在全局/静态存储区
- char s[]= ”abc” s 在栈区
- char *p = “123456” p 在栈区
- p1 = (char *) malloc(10) 堆区

- A *a = new A() 自由存储区

4.2 问答题

1.new 和 malloc 的区别

- new 和 delete 是操作符，而 malloc 和 free 是函数。这意味着 new 和 delete 可以被重载，而 malloc 和 free 不行
- 用 new 来分配内存，可以自动调用构造函数初始化对象而且可以自动推断类型，避免了类型转换的麻烦
- new 和 delete 可以处理对象，而 malloc 和 free 只能处理内存块。在使用 new 时，会自动调用构造函数初始化对象，在使用 delete 时，会自动调用析构函数释放对象。而使用 malloc 和 free 时，需要手动调用构造函数和析构函数
- new 和 delete 可以重载，而 malloc 和 free 不行。在 C++ 中，可以通过重载 new 和 delete 操作符来实现自定义的内存分配和释放方式。这在某些特殊情况下非常有用，例如实现内存池等
- new 和 delete 是类型安全的。在使用 new 时，编译器会自动检查类型，并在编译时发现类型错误。而使用 malloc 时，由于它返回的是 void* 类型的指针，需要手动进行类型转换，容易出现类型错误

2.delete p、delete[ ] p、allocator 都有什么作用?

- delete p 用于释放单个对象的内存
- delete[] p 用于释放数组的内存
- allocator 可以动态地分配内存并管理该内存，它提供了一种比 new 和 delete 更灵活的方式来管理内存，通过使用 allocator，我们可以在不直接使用 new 和 delete 的情况下分配和释放内存

3. malloc 申请的存储空间能用 delete 释放吗?

在 C++ 中，使用 malloc 申请的存储空间应该使用 free 来释放。而在 C++ 中，使用 new 来申请存储空间，应该使用 delete 来释放。因为 malloc 和 new 是不同的内存分配方式，malloc 分配的内存需要使用 free 来释放，而 new 分配的内存需要使用 delete 来释放。如果使用 delete 来释放 malloc 分配的内存，可能会导致程序出现未定义的行为。

4. malloc 与 free 的实现原理

malloc 和 free 是 C 语言中用于动态内存分配和释放的函数。malloc 函数用于在堆上动态分配指定大小的内存空间，返回指向该内存区域的指针。free 函数用于释放之前分配的内存空间，使其可以被重新使用。

malloc 函数的实现原理是，当程序调用 malloc 函数时，系统会在堆中寻找一块足够大的空闲内存来满足申请的大小。如果找到的空闲内存大小正好等于申请的大小，则直接返回该内存地址。如果找到

的空闲内存大小大于申请的大小，则将多余的部分放回堆中，以备后续使用。如果找不到足够大的空闲内存，则返回空指针。

free 函数的实现原理是，当程序调用 free 函数时，系统会将之前申请的内存空间标记为可用状态，以便后续使用。如果之前申请的内存空间与其他内存块相邻且空闲，则会将它们合并成一个更大的内存块，以减少内存碎片的产生。

4.3 Exercise

RandomGenerator.h

```h
//
// Created by 77 on 2023/3/23.
//
#include <iostream>
#include <cstdlib>
#ifndef EXERSICE_RANDOMGENERATOR_H
#define EXERSICE_RANDOMGENERATOR_H

int* getrandom(int n);
void output(int* p,int n);
#endif //EXERSICE_RANDOMGENERATOR_H
```

RandomGenerator.cpp

```cpp
//
// Created by 77 on 2023/3/23.
//
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "RandomGenerator.h"
using namespace std;
int* getrandom(int n) {
    srand(time(nullptr));
    int* p = new int[n];
    if(p != nullptr) {
        for (int i = 0; i < n; i++) {
            p[i] = rand() % n;
        }
        return p;
    }
    else {
        return NULL;
    }
```

```
}
void output(int* p,int n) {
    int max = p[0],min = p[0];
    for(int i = 1; i < n; i++) {
        if(max < p[i])
        {
            max = p[i];
        }
        if(min > p[i])
        {
            min = p[i];
        }
    }
//    for(int i = 0; i < n; i++) {
//        cout << p[i] << " ";
//    }
    cout << "The max value is: " << max << endl;
    cout << "The min value is: " << min << endl;

}
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.18)
project(Exersice)

set(CMAKE_CXX_STANDARD 17)

add_executable(Exersice main.cpp RandomGenerator.h RandomGenerator.cpp)
```

main.txt

```
#include <iostream>
#include <cstdlib>
#include "RandomGenerator.h"
using namespace std;
int main() {
    int n;
    cout << "Please input a value: \n";
    cin >> n;
    int* p = getrandom(n);
    if(p == nullptr)
    {
        cout << "Error! Memory allocation failed!\n";
```

```
        return 0;
    }
    else
    {
        output(p,n);
        free(p);
    }
}
```

5 Debug 和 Release

5.1 如何判断动态申请越界（C 方式，注意源程序后缀为.c）

VS2022 Debug

```
addr:00B4C0B8
00B4C0B4:fffffffd
00B4C0B5:fffffffd
00B4C0B6:fffffffd
00B4C0B7:fffffffd
00B4C0B8:31
00B4C0B9:32
00B4C0BA:33
00B4C0BB:34
00B4C0BC:35
00B4C0BD:36
00B4C0BE:37
00B4C0BF:38
00B4C0C0:39
00B4C0C1:00
00B4C0C2:fffffffd
00B4C0C3:fffffffd
00B4C0C4:fffffffd
00B4C0C5:fffffffd
00B4C0C6:ffffffdd
00B4C0C7:ffffffdd
请按任意键继续. . .
```

```
addr:00DC1598
00DC1594:fffffffd
00DC1595:fffffffd
00DC1596:fffffffd
00DC1597:fffffffd
00DC1598:31
00DC1599:32
```

```
00DC159A:33
00DC159B:34
00DC159C:35
00DC159D:36
00DC159E:37
00DC159F:38
00DC15A0:39
00DC15A1:00
00DC15A2:61
00DC15A3:fffffffd
00DC15A4:fffffffd
00DC15A5:fffffffd
00DC15A6:41
00DC15A7:42
```

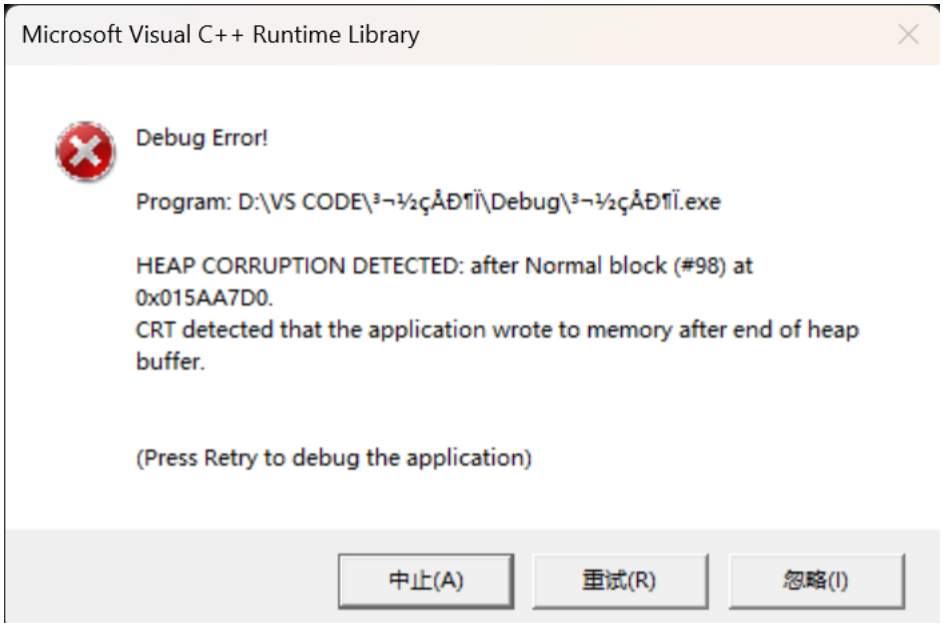
```
addr:00761400
007613FC:fffffffd
007613FD:fffffffd
007613FE:fffffffd
007613FF:fffffffd
00761400:31
00761401:32
00761402:33
00761403:34
00761404:35
00761405:36
00761406:37
00761407:38
00761408:39
00761409:00
0076140A:61
0076140B:fffffffd
0076140C:fffffffd
0076140D:fffffffd
0076140E:41
0076140F:42
请按任意键继续. . .
```

```
addr:010715F8
010715F4:fffffffd
010715F5:fffffffd
010715F6:fffffffd
010715F7:fffffffd
010715F8:31
010715F9:32
010715FA:33
```

```
010715FB:34
010715FC:35
010715FD:36
010715FE:37
010715FF:38
01071600:39
01071601:00
01071602:fffffffd
01071603:fffffffd
01071604:fffffffd
01071605:fffffffd
01071606:41
01071607:42
请按任意键继续. . .
```

在 Visual Studio (VS) 中的 Debug 模式下, CRT (C 运行时库) 调试堆会检测内存泄漏和缓冲区溢出。所有堆函数的调用都会被解析为这些函数在调试堆中运行的调试版本, 执行 malloc 将调用 _malloc_dbg, 调试堆管理器会从基堆分配比请求稍大的内存块，并在数据两侧创建小缓冲区，用于捕获对已分配区域的改写。申请的用户数据区域会以 0xCD 填充，在用户数据区域两侧会各有一个大小为 4 字节的缓冲区（称作 no_mans_land 块），并填充 0xFD。最后，在内存块被释放时，用户数据区域与前后的 no_mans_land 块会被统一填充 0xDD。释放内存块时，调试堆会自动检查已分配区域两侧的缓冲区的完整性，并在发生改写时发出错误报告。这样可以帮助开发者在开发过程中及时发现和解决内存泄漏和缓冲区溢出等问题，提高代码质量和稳定性

在所给出的四种情况中，第三种情况即修改了 no_masland 块，并且 free 对此进行了检测，所以系统会判定超界, 报出一下错误



VS2022 Release

```
addr:0099DE70
0099DE6C:53
0099DE6D:04
0099DE6E:00
0099DE6F:ffffff8e
0099DE70:31
0099DE71:32
0099DE72:33
0099DE73:34
0099DE74:35
0099DE75:36
0099DE76:37
0099DE77:38
0099DE78:39
0099DE79:00
0099DE7A:fffffffd
0099DE7B:00
0099DE7C:74
0099DE7D:00
0099DE7E:41
0099DE7F:42
请按任意键继续. . .
```

四种情况运行的过程中并没有报错，也就是说 Release 情况下并没有上面的那种检测机制

但是我们还可以借用一些其他方式来判断动态申请越界，例如 AddressSanitizer (ASan) 工具来判断动态申请越界。ASan 是一种内存错误检测工具，可以帮助开发者在运行时发现内存错误，如越界访问、内存泄漏等。

GBD

在 GBD 调试过程中，同样也没有类似 VS2022 Debug 一样的检测机制，只能借用一些其他工具，可以使用 Valgrind 工具来判断动态申请越界。

5.2 如何判断动态申请越界（c++）

结果与 c 相同，debug 状态下有结合缓冲区的超界检测机制，其他两种没有

5.3 如何判断普通数组的越界访问（C++ 方式，注意源程序后缀为.cpp）

```cpp
#include <iostream>
#include <cstring>
#include <cstdlib>

using namespace std;
```

```
6
7  int main()
8  {
9    int a[10];
10   for (int i = 0; i < 10; i++)
11   {
12     a[i] = i;
13   }
14   a[10] = 10;
15   a[14] = 14;
16
17   for (int i = 0; i < 16; i++)
18     printf("%p:%02x\n", &a[i], a[i]);
19
20   char ch[10] = "abcdefghi";
21   ch[10] = 'j';
22   ch[11] = 'k';
23   for (int i = 0; i < 16; i++)
24     printf("%p:%02x\n", &ch[i], ch[i]);
25   system("pause");
26  }
```

没有报错，在 vs 里面会显示

写入”a” 时缓冲区溢出。

索引“11”超出了“0”至“9”的有效范围 (对于可能在堆栈中分配的缓冲区“a”)。

5.4 总结

操作系统提供了虚拟内存机制，将程序的地址空间与物理内存分离，从而实现了内存的保护。当程序访问一个地址时，操作系统会检查该地址是否合法，如果不合法就会触发内存访问异常，从而终止程序的执行。这种机制可以防止程序越界访问内存，同时也可以保护操作系统不受恶意程序的攻击。

编译器可以在编译时对程序进行静态分析，检查数组、指针等数据结构的访问是否越界。如果发现越界访问，编译器会发出警告或错误信息，提示程序员进行修正。此外，一些编译器还提供了内存检测工具，可以在程序运行时动态地检测越界访问，并在发现问题时终止程序的执行。

虽然操作系统和编译器都会防范越界，但是我们编写的同时，也要去注意内容超界问题，结合上学期的写代码经验，我发现容易出现超界的地方往往是在对数组遍历并修改过程，在这样的地方应该多加一些限制条件，尽可能地考虑更多情况