Student Name: 王研博
Student ID: 2022141460122

高级语言程序设计
Assignment 2

Section 1 Makefile Cmake

Question: 在 stuinfo.h 中声明一个名为 stuinfo 的结构和下面的四个函数原型。在 stufun.cpp 中实现这四个函数。编写一个包含 main () 的 main.cpp，并演示原型函数的所有特性。
Answer:

CMakeLists.txt:

```
cmake_minimum_required(VERSION 3.24)
project(Stuinfo)

set(CMAKE_CXX_STANDARD 20)

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

stuinfo.h:

```
#ifndef STUINFO_H
#define STUINFO_H

#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
```

stufun.cpp:

```cpp
#include "stuinfo.h"
#include <algorithm>
#include <cstring>
using namespace std;

void inputstu(stuinfo stu[] , int n)
{
    for (int i = 0; i < n; i++)
    {
        cout << "Please input the name of the student: ";
        cin >> stu[i].name;
        cout << "Please input the score of the student: ";
        cin >> 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++)
    {
        cout << "The name of the student is: " << stu[i].name << endl;
        cout << "The score of the student is: " << stu[i].score[0] << " " << stu[i].score[1] << " " <<
            stu[i].score[2] << endl;
        cout << "The average score of the student is: " << stu[i].ave << endl;
    }
}
bool cmp_ave(stuinfo a, stuinfo b)
{
    return a.ave > b.ave;
}
void sortstu(stuinfo stu[] , int n)
{
    sort(stu, stu + n, cmp_ave);
}
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 <iostream>
#include "stuinfo.h"
using namespace std;
int main()
{
    int n;
    cout<<"Please input the number of students: ";
    cin>>n;
    struct stuinfo stu[n];
    inputstu(stu, n);
    showstu(stu, n);
    sortstu(stu, n);
    cout<<"The sorted students are: "<<endl;
    showstu(stu, n);
    char ch[20];
    cout<<"Please input the name of the student you want to find: ";
    cin>>ch;
    if (findstu(stu, n, ch))
    {
        cout<<"The student is found!"<<endl;
    }
    else
    {
        cout<<"The student is not found!"<<endl;
    }
    return 0;
}
```

Demo

```
Please input the number of students: 3
Please input the name of the student: Jack
Please input the score of the student: 89 79 96
Please input the name of the student: Amy
Please input the score of the student: 97 68 94
Please input the name of the student: Mike
Please input the score of the student: 84 86 99
The name of the student is: Jack
The score of the student is: 89 79 96
The average score of the student is: 88
The name of the student is: Amy
The score of the student is: 97 68 94
The average score of the student is: 86.3333
The name of the student is: Mike
The score of the student is: 84 86 99
The average score of the student is: 89.6667
The sorted students are:
The name of the student is: Mike
The score of the student is: 84 86 99
The average score of the student is: 89.6667
The name of the student is: Jack
The score of the student is: 89 79 96
The average score of the student is: 88
The name of the student is: Amy
The score of the student is: 97 68 94
The average score of the student is: 86.3333
Please input the name of the student you want to find: Jack
The student is found!
```

Section 2 Types

2.1: 问答题

2.1.1:static 的用法和作用

Answer:

```
1:Static variables: When "static" is used in front of a variable inside a function, it declares the variable as a static variable. A static variable retains its value between function calls, and its lifetime is the entire duration of the program. Static variables are initialized only once, and the value is retained across function calls.

2.Static functions: When "static" is used in front of a function, it declares the function as a static function. Static functions are only visible within the file in which they are defined, and they cannot be called by functions in other files. This makes static functions useful for encapsulation and information hiding.

3.Static class members: When "static" is used in front of a member variable or function inside a class, it declares the member as a static class member. Static class members are shared by all instances of the class and are not tied to any specific instance of the class. They can be accessed without creating an object of the class.

4.Static libraries: Static libraries are used to link together object files to create a single executable file. Static libraries contain compiled code and are linked into the final executable at compile time. Static libraries are useful for distributing a single executable file that does not require external dependencies.
```

2.1.2: 什么是隐式转换，如何消除隐式转换？

Answer:

```
Definition:
Implicit conversion is the automatic conversion of one data type to another data type by the programming language or compiler. This conversion is done automatically by the compiler, without any explicit instruction from the programmer.

Solution:
1.Use explicit type conversions: Explicit type conversions require the programmer to specify the conversion explicitly, which helps to avoid any unintentional conversions.

2. Instead of using the +/- operator for concatenation of different data types may help.
```

2.2: 程序解释题

2.2.1:

Code and explanation

```cpp
#include <iostream>
using std::cout;
using std::endl;

int main() {
    int num1 = 1234567890;
    int num2 = 1234567890;
    int sum = num1 + num2;
    cout << "sum = " << sum << endl;
    //The largest number that can be represented by the int type is 2^31-1 = 2147483647
    //So the value of sum is overflowed and the result is wrong.
    float f1 = 1234567890.0f;
    float f2 = 1.0f;
    float fsum = f1 + f2;
    cout << "fsum = " << fsum << endl;
    cout << "(fsum == f1) is " << (fsum == f1) << endl;
    //The float type has a precision of 6-7 decimal places.
    //However, but fsum has a ninth place precision, can't be represented by the float type.
    //So the value of fsum is equal to f1.
    float f = 0.1f;
    float sum10x = f + f + f + f + f + f + f + f + f + f;
    float mul10x = f * 10;
    //When added,  there is a rounding error that accumulates and affects
    //the final result.1 When multiplied, there is only one rounding error
    //and you get exactly the result you want
    cout<<"sum10x = "<< sum10x << endl;
    cout<<"mul10x = "<< mul10x << endl;
    cout<<"(sum10x == 1) is "<< (sum10x == 1.0) << endl;
    cout<<"(mul10x == 1) is "<< (mul10x == 1.0) << endl;
     return 0;
}
```

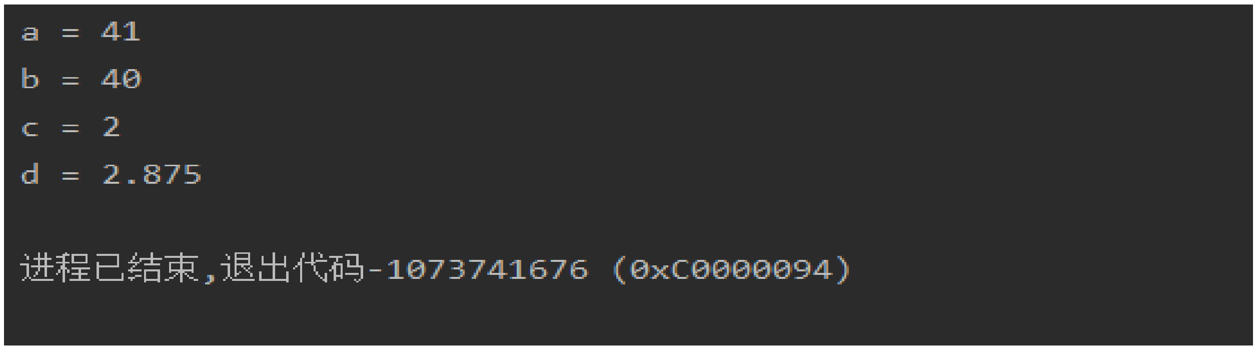
2.2.2:

Code and explanation

```
#include iostream
using namespace std;

int main()
{
    cout  fixed;
    float f1 = 1.0f;
    coutf1 = f1endl;

    float a = 0.1f;
    float f2 = a+a+a+a+a+a+a+a+a+a;
    coutf2 = f2endl;

    if(f1 == f2)
    cout  f1 = f2  endl;
    else
    cout  f1 != f2  endl;
    \\When adding, each addition produces one rounding error, which accumulates to 10 errors.
    \\When multiplying, there is only one rounding error, so f1 is not equal to f2.
}
```

2.2.3:

The Output:



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

进程已结束,退出代码-1073741676 (0xC0000094)
```

Code and explanation

```cpp
#include <iostream>
using namespace std;
int main()
{
    int a, b;
    double c, d;
    a = 19.99 + 21.99;
    b = (int)19.99 + (int)21.99;
    c = 23 / 8;
    d = 23 / 8.0;
    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
    //When calculating a, first add the floating point numbers and then force a type conversion to int.
    //When calculating b, the floating point number is first converted to an int type and then summed.
    //So the result is different
    cout << "c = " << c << endl;
    cout << "d = " << d << endl;
    //When calculating c, the result is an integer, so the decimal part is discarded.
    //When calculating d, the result is a floating point number, so the decimal part is not discarded.
    cout << "0/0= " << 0/0 << endl;
    //Dividing an integer by zero in c++ is undefined behavior and has no standard result or exception
        handling.
    // Different compilers or platforms may behave differently, such as b crashing, returning infinity,
        returning 0, etc.
    return 0;
}
```

Setcion 3 Structs

3.1: 结构体对齐问题

Explanation:

```
In the first code, alignas specifies that the alignment of the Info2 structure is 4 bytes, because the
    largest variable type is 2 bytes, so it can be aligned correctly.
```

```
In the second code, alignas specifies the alignment of the Info2 structure as 2 bytes, but the largest
    variable type is 4 bytes and cannot be aligned in 2-byte format, so the compiler automatically s
    set to 4-byte alignment.
```

3.2:Exercise

CMakeList.txt

```
cmake_minimum_required(VERSION 3.22)
project(4_3)
# 指定要构建的动态库名称
add_library(mylib SHARED Implement.cpp)

# 将mylib库添加到链接列表中

set(CMAKE_CXX_STANDARD 17)

add_executable(4_3  main.cpp struct.h)
target_link_libraries(4_3 mylib)
```

struct.h

```
#ifndef STRUCT_H
#define STRUCT_H
#include <cmath>
struct Point { double x, y; }; // 点
using Vec = Point; // 向量
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
//判断三个点构成的是否为三角形
bool isTriangle(const Point &A, const Point &B, const Point &C);
// 三角形的重心
std::pair<bool, Point> barycenter(const Point &A, const Point &B, const Point &C);
// 三角形的外心
std::pair<bool, Point> circumcenter(const Point &A, const Point &B, const Point &C);
// 三角形的内心
std::pair<bool, Point> incenter(const Point &A, const Point &B, const Point &C);
// 三角形的垂心
std::pair<bool, Point> orthocenter(const Point &A, const Point &B, const Point &C);
#endif
```

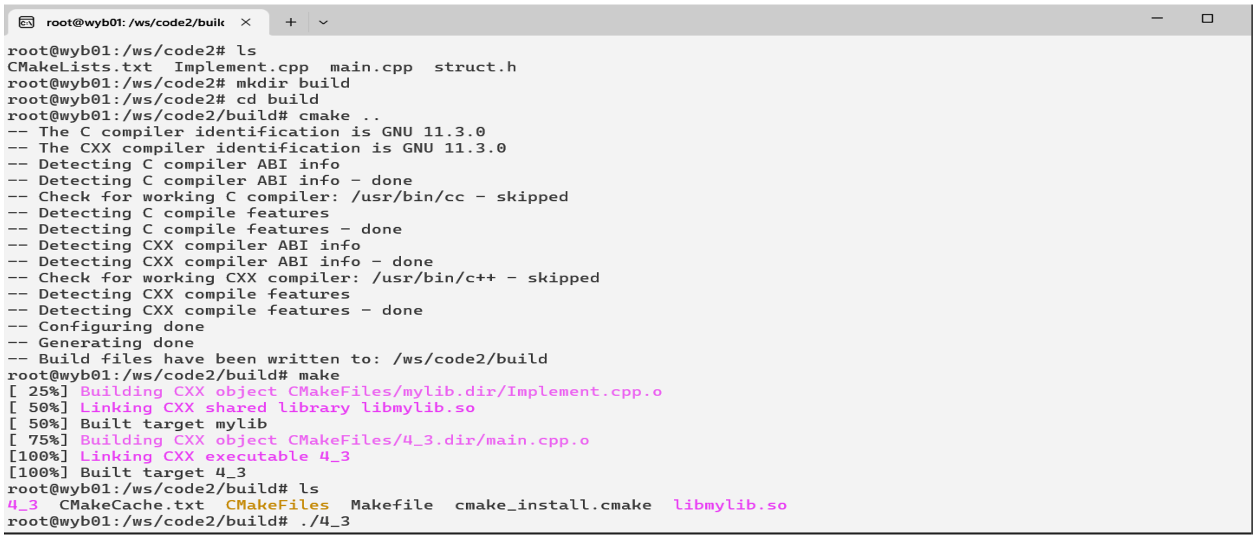
Implement.cpp

```cpp
#include "struct.h"
#include <cmath>

//判断三个点构成的是否为三角形
bool isTriangle(const Point &A, const Point &B, const Point &C)
{
    return (A.x - B.x) * (A.y - C.y) != (A.x - C.x) * (A.y - B.y);
}
// 三角形的重心
std::pair<bool, Point> barycenter(const Point &A, const Point &B, const Point &C)
{
    if (!isTriangle(A, B, C)) return {false, 0};
    return {true, {(A.x + B.x + C.x) / 3, (A.y + B.y + C.y) / 3}};
}
// 三角形的外心
std::pair<bool, Point> circumcenter(const Point &A, const Point &B, const Point &C)
{
    if (!isTriangle(A, B, C)) return {false, 0};
    Point p1 = {(A.x + B.x) / 2, (A.y + B.y) / 2};
    Point p2 = {(A.x + C.x) / 2, (A.y + C.y) / 2};
    Vec v1 = {B.y - A.y, A.x - B.x};
    Vec v2 = {C.y - A.y, A.x - C.x};
    double t = ( v2.x * (p1.y - p2.y) - v2.y * (p1.x - p2.x)) / (v1.x * v2.y - v1.y * v2.x);
    return {true, {p1.x + t * v1.x, p1.y + t * v1.y}};

}
// 三角形的内心
std::pair<bool, Point> incenter(const Point &A, const Point &B, const Point &C)
{
    if(!isTriangle(A, B, C)) return {false, 0};
    double a = sqrt((B.x - C.x) * (B.x - C.x) + (B.y - C.y) * (B.y - C.y));
    double b = sqrt((A.x - C.x) * (A.x - C.x) + (A.y - C.y) * (A.y - C.y));
    double c = sqrt((A.x - B.x) * (A.x - B.x) + (A.y - B.y) * (A.y - B.y));
    return {true, {(a * A.x + b * B.x + c * C.x) / (a + b + c), (a * A.y + b * B.y + c * C.y) / (a + b +
         c)}};

}
// 三角形的垂心
std::pair<bool, Point> orthocenter(const Point &A, const Point &B, const Point &C) {
    if (!isTriangle(A, B, C)) return {false, 0};
    double vC = (B.x - A.x) / (A.y - B.y);
    double vB = (C.x - A.x) / (A.y - C.y);
    Point p1;
    p1.x = (B.y - C.y - vB * B.x + vC * C.x) / (vC - vB);
    p1.y = vC * (p1.x - C.x) + C.y;
    return {true, p1};
}
```

main.cpp

```cpp
#include <iostream>
#include "struct.h"
int main()
{
    Point A, B, C;
    std::cin>> A.x>> A.y>> B.x>> B.y>> C.x>> C.y;
    if(!isTriangle(A, B, C))
    {
        std::cout<< "Not a triangle!"<<std::endl;
        return 0;
    }
    std::cout<< barycenter(A, B, C).first<<' '<<barycenter(A, B, C).second.x<<" "<<barycenter(A, B, C).
        second.y<<std::endl;
    std::cout<< circumcenter(A, B, C).first<<' '<<circumcenter(A, B, C).second.x<<" "<<circumcenter(A, B
        , C).second.y<<std::endl;
    std::cout<< incenter(A, B, C).first<<' '<<incenter(A, B, C).second.x<<" "<<incenter(A, B, C).second.
        y<<std::endl;
    std::cout<< orthocenter(A, B, C).first<<' '<<orthocenter(A, B, C).second.x<<" "<<orthocenter(A, B, C
        ).second.y<<std::endl;
    return 0;
}
```

Linux 下创建动态库进行编译



```
root@wyb01:/ws/code2# ls
CMakeLists.txt  Implement.cpp  main.cpp  struct.h
root@wyb01:/ws/code2# mkdir build
root@wyb01:/ws/code2# cd build
root@wyb01:/ws/code2/build# cmake ..
-- The C compiler identification is GNU 11.3.0
-- The CXX compiler identification is GNU 11.3.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /ws/code2/build
root@wyb01:/ws/code2/build# make
[ 25%] Building CXX object CMakeFiles/mylib.dir/Implement.cpp.o
[ 50%] Linking CXX shared library libmylib.so
[ 50%] Built target mylib
[ 75%] Building CXX object CMakeFiles/4_3.dir/main.cpp.o
[100%] Linking CXX executable 4_3
[100%] Built target 4_3
root@wyb01:/ws/code2/build# ls
4_3  CMakeCache.txt  CMakeFiles  Makefile  cmake_install.cmake  libmylib.so
root@wyb01:/ws/code2/build# ./4_3
```

Output

```
root@wyb01:/ws/code2/build# ./4_3
5 9 8 5 6 3
1 6.33333 5.66667
1 5.07143 5.92857
1 6.67119 5.09444
1 8.85714 5.14286
root@wyb01:/ws/code2/build# ./4_3
9 6 2 5 8 7
1 6.33333 6
1 5.75 3.75
1 7.8541 6.38197
1 7.5 10.5
root@wyb01:/ws/code2/build#
```

Section 4 C++ 动态内存申请

4.1: C++ 的内存分区

Codes and explanation

```cpp
#include <iostream>
#include <cstdlib>
using namespace std;
class A1{
    int num;
};

static int a; //(1) Static variable exists in the global area
auto b=0; //(2) Global variable exists in the global area
int main()
{
    char s[]="abc"; //(3) The s array exists on the stack and the content is in the global area

    char *p="123456"; //(4) The p pointer itself exists on the stack and points to content in the global area

    char* p1= (char *)malloc(10); // (5) The p1 pointer itself exists on the stack and points to content in the heap area

    A1 *f = new A1(); // (6) The f pointer itself exists on the stack and points to content in the heap area
    cout<<"&a: "<<&a<<endl;
    cout<<"&b: "<<&b<<endl;
    cout<<"&s: "<<&s<<endl;
    cout<<"&p: "<<&p<<endl;
    cout<<"&p1: "<<&p1<<endl;
    cout<<"&f: "<<&f<<endl;
}
```

Evidence

```
&a: 0x557514cdf15c
&b: 0x557514cdf154
&s: 0x7fffd4b75fc4
&p: 0x7fffd4b75fa8
&p1: 0x7fffd4b75fb0
&f: 0x7fffd4b75fb8
```

4.2: 问答题

4.2.1: new 和 malloc 的区别

```
1. new is an operator, while malloc is a function.
2. new automatically calculates the size of the memory space to be allocated, while malloc needs to
    manually specify the size of the allocated memory space.
3. new calls the constructor of the object, while malloc just allocates the memory space and does not
    call the constructor.
4. new will return a pointer to the object type, while malloc returns a pointer of type void*, which
    requires type conversion.
```

4.2.2: delete p、delete[ ] p、allocator 都有什么作用?

```
1. delete p is used to free the memory of a single object allocated by new. If p is a null pointer,
    delete p does not perform any operation. If p is not a pointer allocated by new, the behavior is
    undefined.

2. delete[] p is used to free the memory of an array of objects allocated by new[]. If p is a null
    pointer, delete[] p does not perform any operation. If p is not a pointer allocated by new[], the
    behavior is undefined.

3. allocator is a C++ standard library class for allocating and freeing memory. It provides a generic
    interface that makes it possible to switch between different memory allocators without modifying
    the code. a common use of the allocator class is to allocate memory in container classes (such as
    vector, list, map, etc.). By using allocator, you can ensure that memory allocation and freeing of
    container classes is consistent with the rest of the application.
```

4.2.3: malloc 申请的存储空间能用 delete 释放吗?

```
No. In C++, malloc() and free() are C functions that are used to dynamically allocate and free memory.
    New and delete are C++ operators that are also used to dynamically allocate and free memory. In C
    ++, new and delete should be used in pairs with malloc() and free(), not in combination. Thus, if
    you allocate memory space using malloc(), you should use free() to free it, not delete, and
    similarly, if you allocate memory space using new, you should use delete to free it, not free().
    Mixing these functions and operators can lead to memory leaks or program crashes.
```

4.2.3：malloc 与 free 的实现原理?

```
Principle of malloc function implementation.

1. The malloc function first checks if the arguments passed in are legal.

2. If the argument is legal, the malloc function requests a contiguous section of memory from the
    operating system. The size of this space is at least the size of the memory to be allocated plus
    some extra bytes to store some management information, such as the size of the allocated memory
    block, whether it has been allocated, etc.

3. If the operating system successfully allocates memory, the malloc function returns the address of
    this memory space to the caller as the starting address of the allocated memory block.

4. If the operating system is unable to allocate the required memory space, the malloc function will
    return a null pointer.

How the free function is implemented.

1. The free function first checks if the incoming pointer is legal, i.e., if it is NULL or if it points
    to a block of memory that has already been freed.

If the pointer is legal, the free function will mark the block pointed to by the pointer as freed.

3. The free function checks if the freed block is adjacent to an adjacent block that has already been
    freed, and if so, merges them into a larger block.

4. Finally, the free function records the address and size of the freed block for subsequent memory
    allocation.
```

4.3：Exercise

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.22)
project(RG)

set(CMAKE_CXX_STANDARD 17)

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

RandomGenerator.h

```cpp
#ifndef RG_RANDOMGENERATOR_H
#define RG_RANDOMGENERATOR_H
#include <iostream>
#include <ctime>
#include <cstdlib>
#include <algorithm>
int* generateRandomArray(int n);
void releaseMemory(int* arr);
#endif //RG_RANDOMGENERATOR_H
```

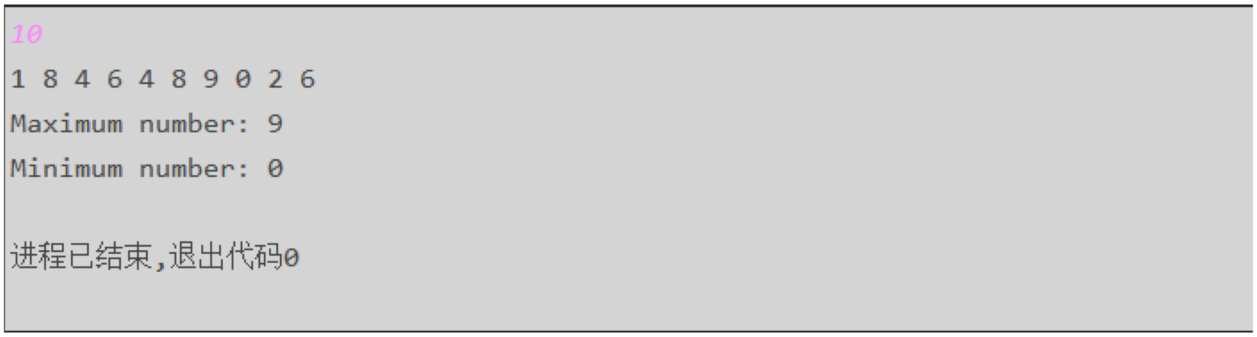
RandomGenerator.cpp

```cpp
#include "RandomGenerator.h"
int* generateRandomArray(int n)
{
    int* arr = new int[n];
    srand(time(NULL));
    for (int i = 0; i < n; i++)
    {
        arr[i] = rand() % n;
    }
    return arr;
}
void releaseMemory(int* arr)
{
    delete[] arr;
}
```

main.cpp

```cpp
#include "RandomGenerator.h"
using namespace std;
int main() {
    int n;
    cin>>n;
    int *arr = generateRandomArray(n);
    sort(arr, arr+n);
    cout << "Maximum number: " << arr[n -1] << endl;
    cout << "Minimum number: " << arr[0] << endl;
    releaseMemory(arr);
    return 0;
}
```

Output



```
10
1 8 4 6 4 8 9 0 2 6
Maximum number: 9
Minimum number: 0

进程已结束,退出代码0
```

Section 5 Debug 和 Release

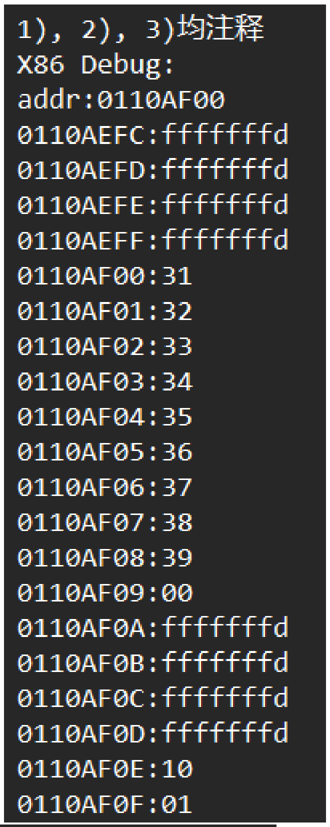
5.1: Debug 和 Release 的区别

5.2: 如何判断动态申请越界（C）

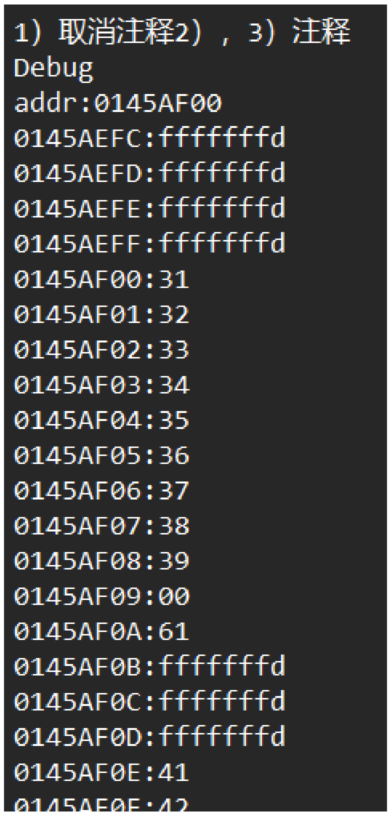
Code

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
  printf("%d\n", '\xdf');
  char* p;
  p = (char*)malloc(10 * sizeof(char));
  if (p == NULL)
    return -1;
  strcpy(p, "123456789");
  //p[10] = 'a'; //此句越界 (1)
  //p[14] = 'A'; //此句越界
  //p[15] = 'B'; //此句越界
  //p[10] = '\xfd'; //此句越界 (2) An incorrect character
  printf("addr:%p\n", p);
  for (int i = -4; i < 16; i++) //注意, 只有 0-9 是合理范围, 其余都是越界读
    printf("%p:%02x\n", (p + i), p[i]);
  //free(p);//(3)
  return 0;
}
```

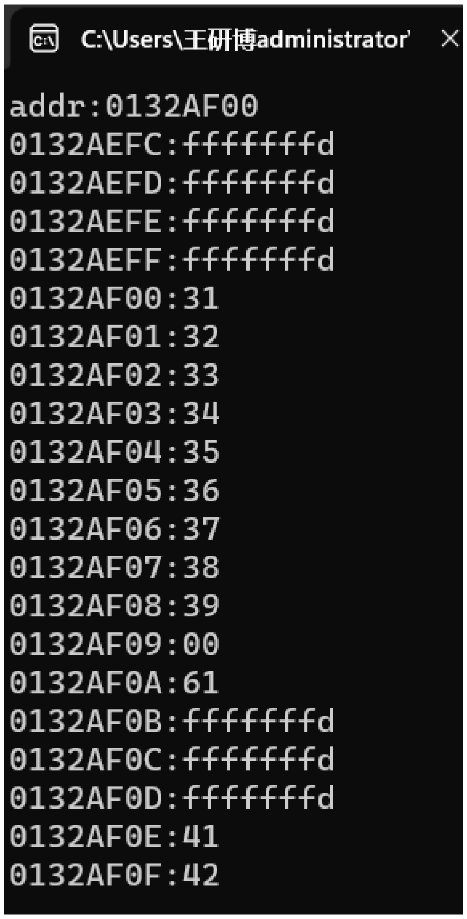
Output and Explanation



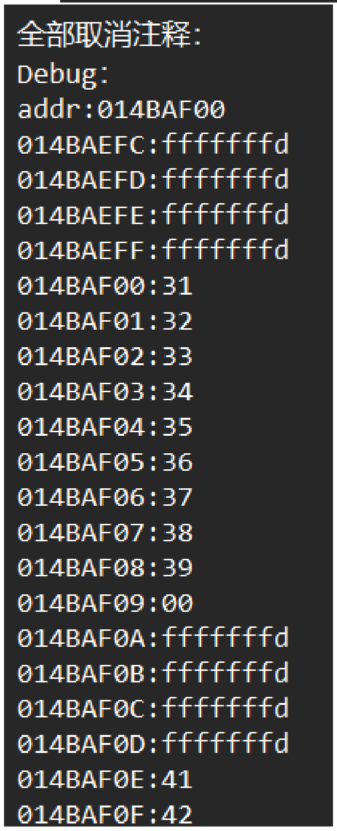
```
1), 2), 3)均注释
X86 Debug:
addr:0110AF00
0110AEFC:fffffffd
0110AEFD:fffffffd
0110AEFE:fffffffd
0110AEFF:fffffffd
0110AF00:31
0110AF01:32
0110AF02:33
0110AF03:34
0110AF04:35
0110AF05:36
0110AF06:37
0110AF07:38
0110AF08:39
0110AF09:00
0110AF0A:fffffffd
0110AF0B:fffffffd
0110AF0C:fffffffd
0110AF0D:fffffffd
0110AF0E:10
0110AF0F:01
```



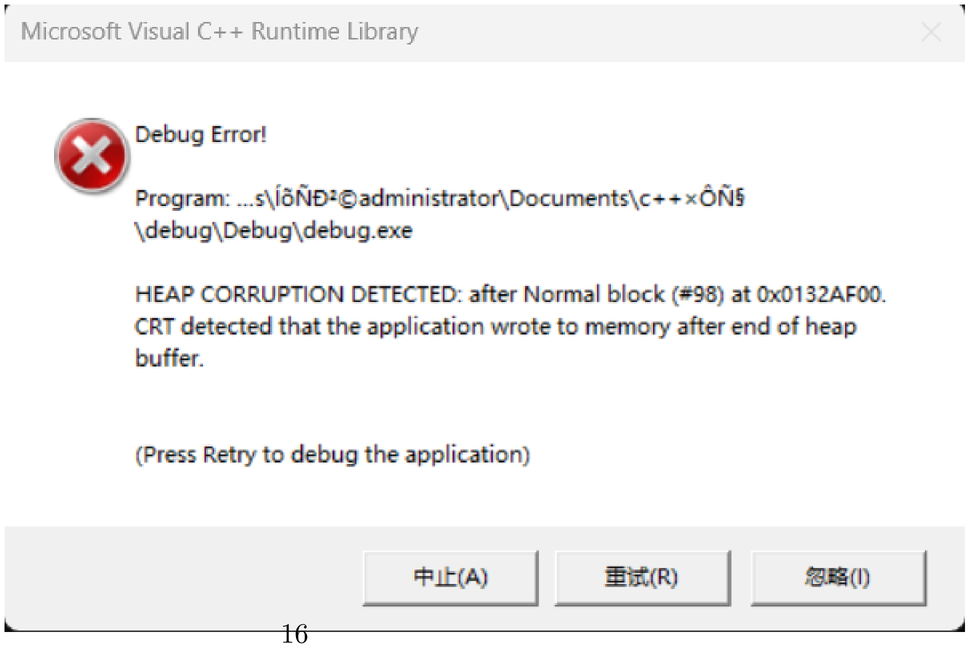
```
1) 取消注释2) , 3) 注释
Debug
addr:0145AF00
0145AEFC:fffffffd
0145AEFD:fffffffd
0145AEFE:fffffffd
0145AEFF:fffffffd
0145AF00:31
0145AF01:32
0145AF02:33
0145AF03:34
0145AF04:35
0145AF05:36
0145AF06:37
0145AF07:38
0145AF08:39
0145AF09:00
0145AF0A:61
0145AF0B:fffffffd
0145AF0C:fffffffd
0145AF0D:fffffffd
0145AF0E:41
0145AF0F:42
```



```
C:\Users\王研博administrator'
addr:0132AF00
0132AEFC:fffffffd
0132AEFD:fffffffd
0132AEFE:fffffffd
0132AEFF:fffffffd
0132AF00:31
0132AF01:32
0132AF02:33
0132AF03:34
0132AF04:35
0132AF05:36
0132AF06:37
0132AF07:38
0132AF08:39
0132AF09:00
0132AF0A:61
0132AF0B:fffffffd
0132AF0C:fffffffd
0132AF0D:fffffffd
0132AF0E:41
0132AF0F:42
```



```
全部取消注释:
Debug:
addr:014BAF00
014BAEFC:fffffffd
014BAEFD:fffffffd
014BAEFE:fffffffd
014BAEFF:fffffffd
014BAF00:31
014BAF01:32
014BAF02:33
014BAF03:34
014BAF04:35
014BAF05:36
014BAF06:37
014BAF07:38
014BAF08:39
014BAF09:00
014BAF0A:fffffffd
014BAF0B:fffffffd
014BAF0C:fffffffd
014BAF0D:fffffffd
014BAF0E:41
014BAF0F:42
```



```
Microsoft Visual C++ Runtime Library

Debug Error!

Program: ...s\ÍõÑÐ²©administrator\Documents\c++×ÔÑ§
\debug\Debug\debug.exe

HEAP CORRUPTION DETECTED: after Normal block (#98) at 0x0132AF00.
CRT detected that the application wrote to memory after end of heap
buffer.

(Press Retry to debug the application)
```

```
结论：在Visual Studio X86/Debug的情况下，编译器会自动给申请的内存前后四个字节设置了判断标志fffffffd，若
    释放内存的时候标志fffffffd被改为其他数据，则判断为数组越界
在Visual Studio X86/Debug和Linux gdb调试运行的情况下，并没有标志，两者运行结果类似
```

Findings

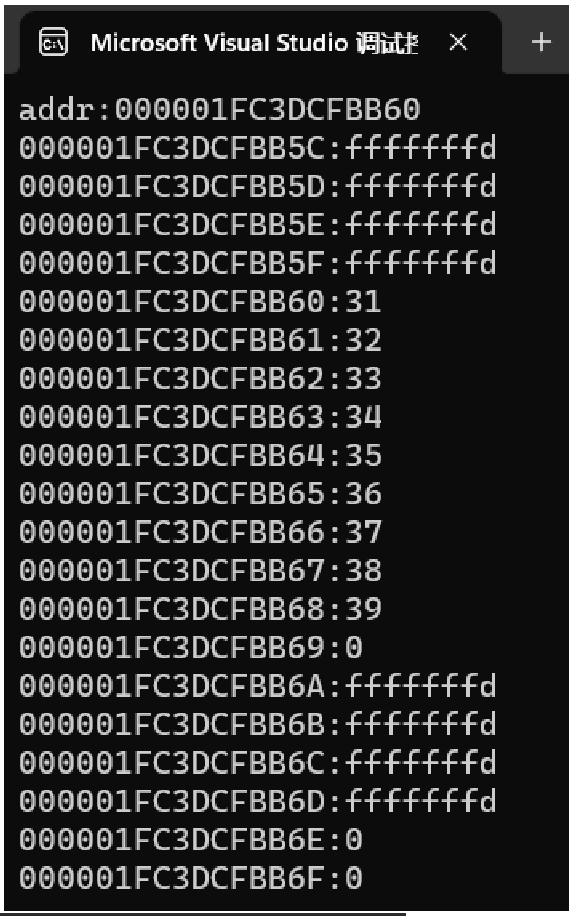
```
（基本同C语言）在第三次实验的时候，Visual Studio X86/Debug会报错，但是第四次实验的时候却不会报错，
两次相差的就是p[10] = '\xfd'; 的更改。
思考得知：第三次报错的原因是因为越界改动了10， 14， 15，但是标志位fffffffd只检查4个
字节，所以其中起作用的是第10位，改动成了'a'，被编译器检查出了越界，故报错。但是第四次
又将第10位改动成了'\xfd'，是一个无效的字符，所以第十位又恢复成了fffffffd这个标志位的
形式，所以第四次没报错。
```

5.3：如何判断动态申请越界（C++）

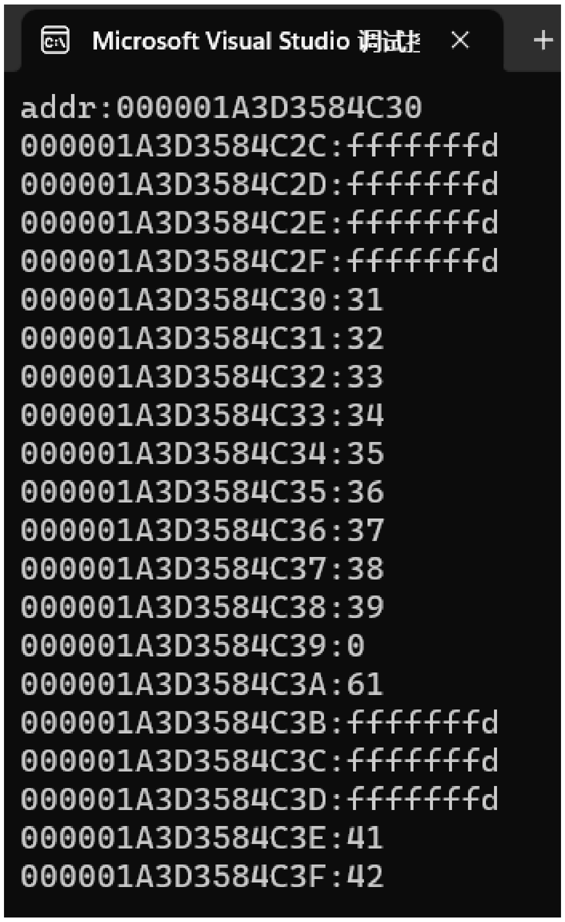
Code

```cpp
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
  char* p;
  p = new(nothrow) char[10];
  if (p == NULL)
    return -1;
  strcpy(p, "123456789");
  p[10] = 'a'; //此句越界 (1)
  p[14] = 'A'; //此句越界
  p[15] = 'B'; //此句越界
  //p[10] = '\xfd'; //此句越界 (2)
  cout << "addr:" << hex << (void*)(p) << endl;
  for (int i = -4; i < 16; i++) //注意，只有 0-9 是合理范围，其余都是越界读
    cout << hex << (void*)(p + i) << ":" << int(p[i]) << endl;
  delete[]p;//(3)

  return 0;
}
```

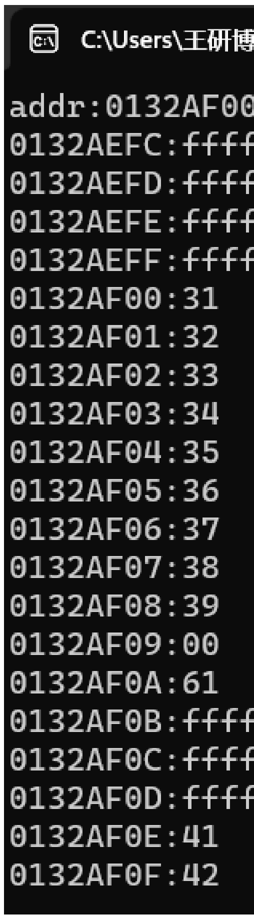
Output and Explanation



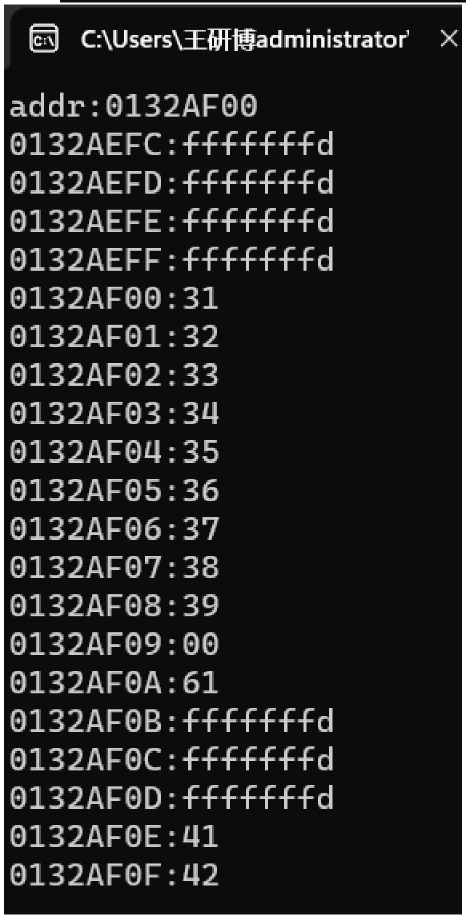
```
addr:000001FC3DCFBB60
000001FC3DCFBB5C:fffffffd
000001FC3DCFBB5D:fffffffd
000001FC3DCFBB5E:fffffffd
000001FC3DCFBB5F:fffffffd
000001FC3DCFBB60:31
000001FC3DCFBB61:32
000001FC3DCFBB62:33
000001FC3DCFBB63:34
000001FC3DCFBB64:35
000001FC3DCFBB65:36
000001FC3DCFBB66:37
000001FC3DCFBB67:38
000001FC3DCFBB68:39
000001FC3DCFBB69:0
000001FC3DCFBB6A:fffffffd
000001FC3DCFBB6B:fffffffd
000001FC3DCFBB6C:fffffffd
000001FC3DCFBB6D:fffffffd
000001FC3DCFBB6E:0
000001FC3DCFBB6F:0
```



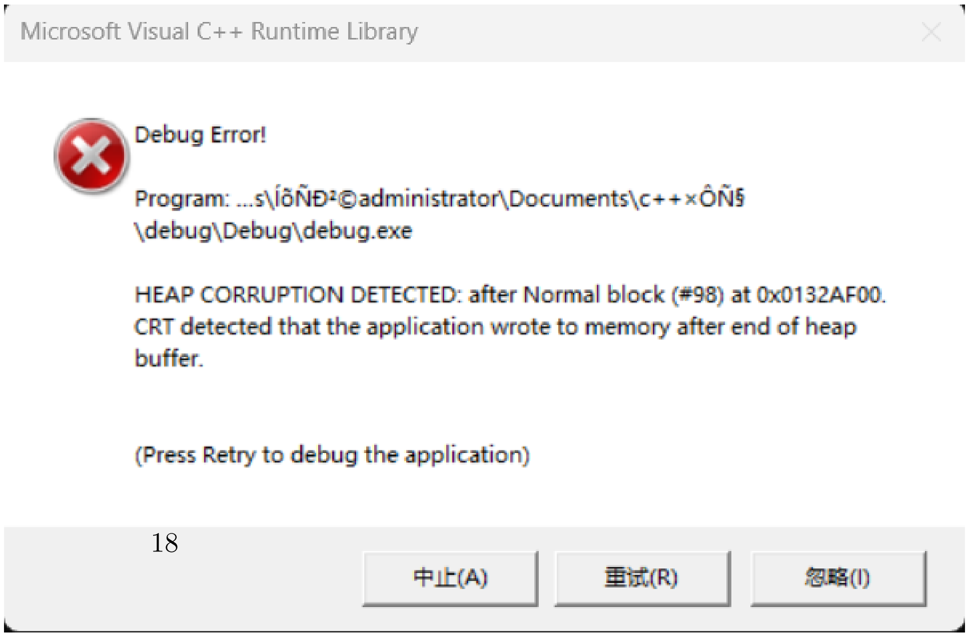
```
addr:000001A3D3584C30
000001A3D3584C2C:fffffffd
000001A3D3584C2D:fffffffd
000001A3D3584C2E:fffffffd
000001A3D3584C2F:fffffffd
000001A3D3584C30:31
000001A3D3584C31:32
000001A3D3584C32:33
000001A3D3584C33:34
000001A3D3584C34:35
000001A3D3584C35:36
000001A3D3584C36:37
000001A3D3584C37:38
000001A3D3584C38:39
000001A3D3584C39:0
000001A3D3584C3A:61
000001A3D3584C3B:fffffffd
000001A3D3584C3C:fffffffd
000001A3D3584C3D:fffffffd
000001A3D3584C3E:41
000001A3D3584C3F:42
```



```
addr:0132AF00
0132AEFC:ffff
0132AEFD:ffff
0132AEFE:ffff
0132AEFF:ffff
0132AF00:31
0132AF01:32
0132AF02:33
0132AF03:34
0132AF04:35
0132AF05:36
0132AF06:37
0132AF07:38
0132AF08:39
0132AF09:00
0132AF0A:61
0132AF0B:ffff
0132AF0C:ffff
0132AF0D:ffff
0132AF0E:41
0132AF0F:42
```



```
addr:0132AF00
0132AEFC:fffffffd
0132AEFD:fffffffd
0132AEFE:fffffffd
0132AEFF:fffffffd
0132AF00:31
0132AF01:32
0132AF02:33
0132AF03:34
0132AF04:35
0132AF05:36
0132AF06:37
0132AF07:38
0132AF08:39
0132AF09:00
0132AF0A:61
0132AF0B:fffffffd
0132AF0C:fffffffd
0132AF0D:fffffffd
0132AF0E:41
0132AF0F:42
```



Microsoft Visual C++ Runtime Library

Debug Error!

Program: ...s\ÍõÑÐ²©administrator\Documents\c++×ÔÑ§
\debug\Debug\debug.exe

HEAP CORRUPTION DETECTED: after Normal block (#98) at 0x0132AF00.
CRT detected that the application wrote to memory after end of heap buffer.

(Press Retry to debug the application)

```
结论：在Visual Studio X86/Debug的情况下，编译器会自动给申请的内存前后四个字节设置了判断标志fffffffd，若释放内存的时候标志fffffffd被改为其他数据，则判断为数组越界
在Visual Studio X86/Debug和Linux gdb调试运行的情况下，并没有标志，两者运行结果类似
```

Findings

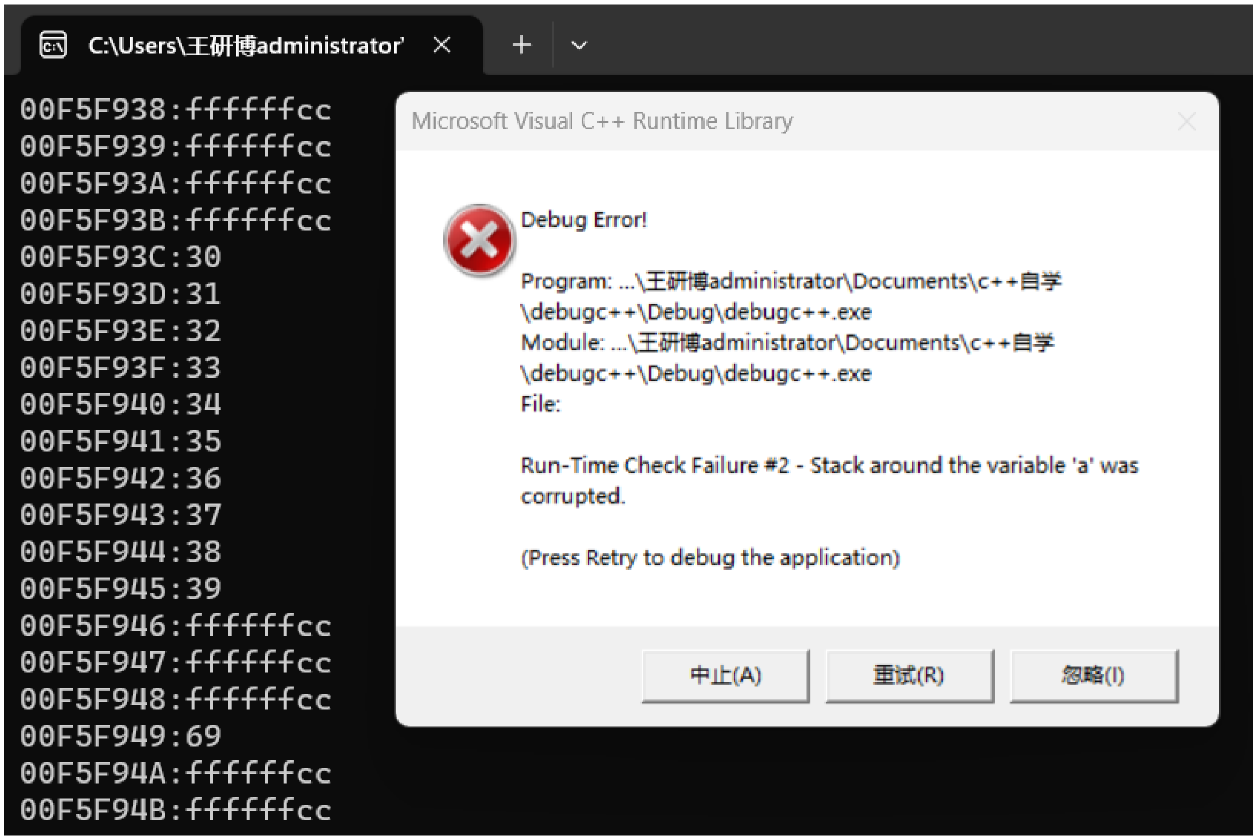
```
（基本同C语言）在第三次实验的时候，Visual Studio X86/Debug会报错，但是第四次实验的时候却不会报错，
两次相差的就是p[10] = '\xfd'; 的更改。
思考得知：第三次报错的原因是因为越界改动了10， 14， 15，但是标志位fffffffd只检查4个
字节，所以其中起作用的是第10位，改动成了'a'，被编译器检查出了越界，故报错。但是第四次
又将第10位改动成了'\xfd'，是一个无效的字符，所以第十位又恢复成了fffffffd这个标志位的
形式，所以第四次没报错。
```

5.4: 如何判断普通数组的越界访问（C++）

My C++ code

```cpp
//Detecting char arrays
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
    char a[10];
    for (int i = 0; i < 10; i++)
    {
        a[i] = (char)('0' + i);
    }
    //a[-2] = ’ i ‘; （检测数据，报错）
    //a[-5] = 'i';（检测数据，未报错）
    //a[14] = 'i';(检测数据，未报错)
    //a[13] = 'i';(检测数据，报错)
    for (int i = -4; i < 16; i++)
        cout << hex << (void*)(a + i) << ":" << int(a[i]) << endl;

    return 0;
}
//Detecting int arrays

#include <iostream>
using namespace std;
int main()
{
    int a[10];
    for (int i = 0; i < 10; i++)
    {
        a[i] = i;
    }
    //a[-2] = -2; (未报错)
     //a[-1] = -1; (报错)
    //a[10] = 10;//(报错 )
    //a[11] = 11;//(报错)
    for (int i = -4; i < 16; i++)
        cout << hex << (void*)(a + i) << ":" << int(a[i]) << endl;

    return 0;
}
```

Char Arrays:



```
00F5F938:ffffffcc
00F5F939:ffffffcc
00F5F93A:ffffffcc
00F5F93B:ffffffcc
00F5F93C:30
00F5F93D:31
00F5F93E:32
00F5F93F:33
00F5F940:34
00F5F941:35
00F5F942:36
00F5F943:37
00F5F944:38
00F5F945:39
00F5F946:ffffffcc
00F5F947:ffffffcc
00F5F948:ffffffcc
00F5F949:69
00F5F94A:ffffffcc
00F5F94B:ffffffcc
```

Microsoft Visual C++ Runtime Library

Debug Error!

Program: ...\王研博administrator\Documents\c++自学\debugc++\Debug\debugc++.exe
Module: ...\王研博administrator\Documents\c++自学\debugc++\Debug\debugc++.exe
File:

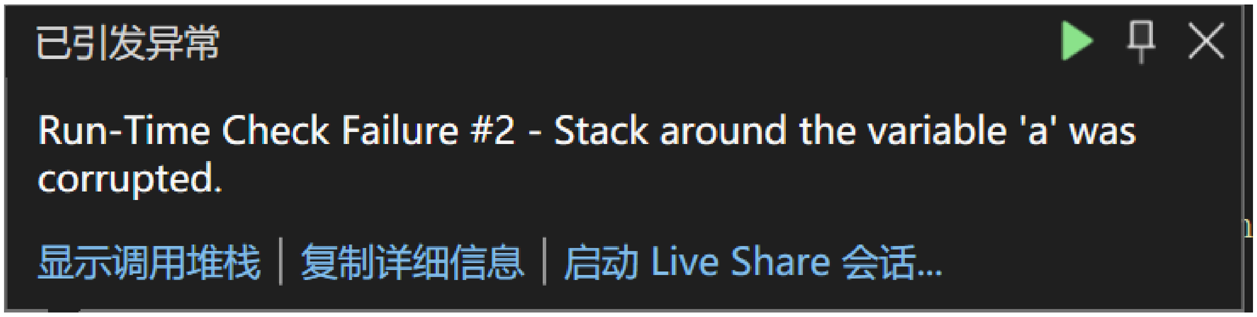
Run-Time Check Failure #2 - Stack around the variable 'a' was corrupted.

(Press Retry to debug the application)

When executing a[13] = 'i', an error is reported

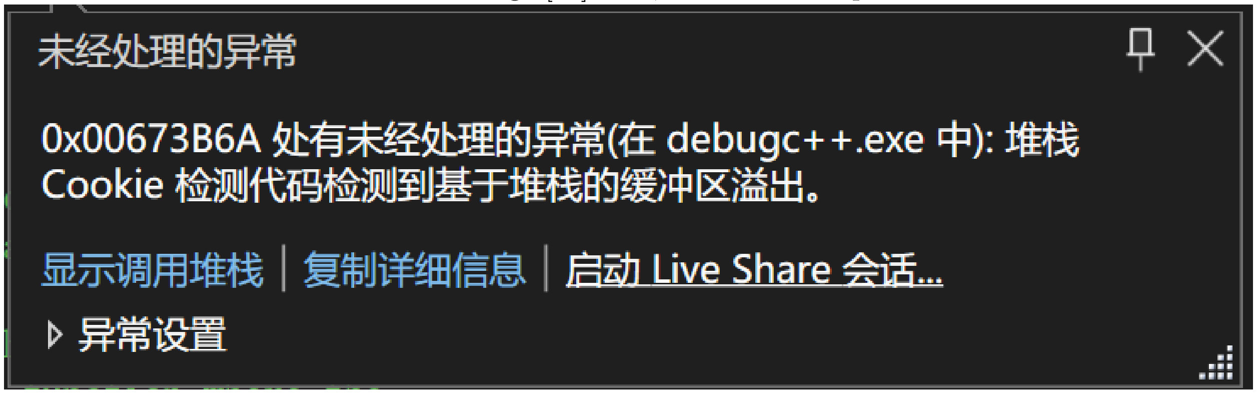
```
总结: 普通字符数组的越界访问与动态申请的字符数组类似, 均设有标志位, 标志位中修改会报错, 标志位外修改不一
    定会报错。 (在普通int型数组中体现了出来)
```

Int Arrays:


已引发异常

Run-Time Check Failure #2 - Stack around the variable 'a' was corrupted.

When executing a[10] = ’i’, an error 1 is reported


未经处理的异常

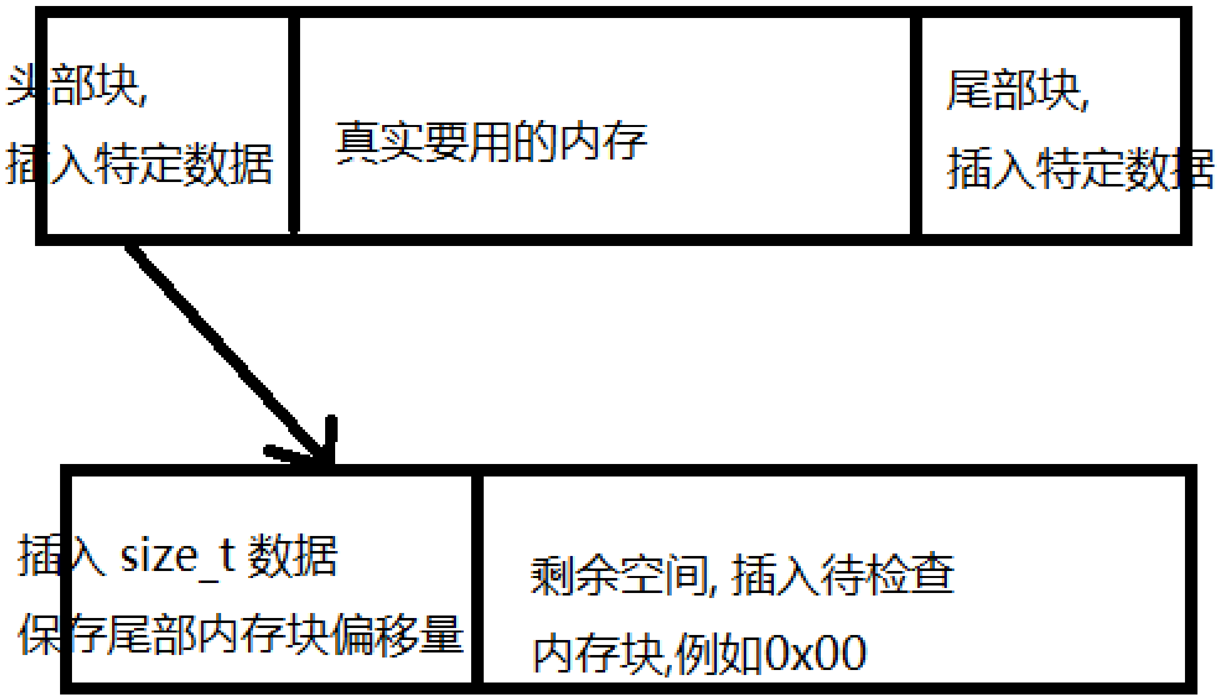
0x00673B6A 处有未经处理的异常(在 debugc++.exe 中): 堆栈 Cookie 检测代码检测到基于堆栈的缓冲区溢出。

When executing a[11] = ’i’, an error 2 is reported

My speculation on the reported error

```
原本根据堆区数组越界的审查方式，推测出编译器做了4个字节为标志位，相当于1位int型变量的大小，但是在这个例子中，我越界访问第10位报如上图图一的错误，
但是当我访问第11位的时候，按照我的推测并不应该报错，但是报了图二显示的错误。据此，我的推测是存在于堆区手动申请的数组是由程序员支配的，但是栈区的
普通数组受到的约束和限制可能更多，导致适用于堆区数组的判别方式并不能够完全应用于栈区的数组。
```

5.5:Summary



Boundary check illustration

Summary about the task

```
内存访问越界总结：内存越界访问是指程序试图访问已经分配给它的内存块之外的内存地址，这种访问可能会导致程序崩溃、数据损坏或安全漏洞等问题。
操作系统检查内存访问越界：
Linux操作系统提供了一些工具来检查内存访问越界。其中一个常用的工具是Valgrind，它是一个用于内存调试、内存泄漏检查、性能分析等的工具集合。Valgrind可以检测到内存访问越界、使用未初始化的内存、内存泄漏等问题。
编译器检查内存访问越界：
编译器可以检查内存访问越界的问题，这是因为编译器在编译代码时会对数组和指针的访问进行检查，确保它们不会越界。如果编译器发现了可能导致越界访问的代码，它会发出警告或错误提示。Visual Studio 在X86/Debug下加入了边界检查值，可以帮助我们发现错误。
应该养成以下习惯防止内存访问越界：
1.始终使用安全的API：在编写代码时，应该使用安全的API来操作内存，例如使用strcpy_s代替strcpy，使用memcpy_s代替memcpy等。

2.检查输入：在处理来自外部的数据时，应该始终检查输入的有效性，例如检查输入的长度是否超出了缓冲区的大小。

3.使用边界检查：在使用数组或指针时，应该始终进行边界检查，以确保不会越界访问。可以使用语言提供的边界检查机制，或者手动编写边界检查代码。

4.避免手动内存管理：手动管理内存容易出现内存泄漏、重复释放等问题，应该尽量避免手动管理内存，使用语言提供的垃圾回收机制或智能指针等工具来管理内存。
```