Student Name: 马越
Student ID: 2022141460097

C++ 面向对象程序设计

Assignment 2

2 Makefile & Cmake

Exercise

源代码与CMakeList.txt

Listing 1: stuinfo.h

```h
#ifndef STUINFO_H
#define STUINFO_H

struct stuinfo {
  char name[20];
  double score[3];
  double ave;
};

// asks the user to enter each of the preceding items of
// information to set the corresponding members of the structure.
void inputstu(stuinfo stu[], int n);

//displays the contents of the structure, one student one line.
void showstu(stuinfo stu[], int n);

//sorts in descending order of average of three scores.
void sortstu(stuinfo stu[], int n);

//finds if given characters is the student's name.
bool findstu(stuinfo stu[], int n, char ch[]);

#endif //STUINFO_H
```

Listing 2: stuinfo.cpp

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

```cpp
#include <iomanip>
#include <algorithm>
#include <cstring>

using std::cin; using std::cout; using std::endl; using std::setw;
using std::sort;

void inputstu(stuinfo stu[], int n) {
  for (int i = 0; i < n; ++i) {
    cout << "Please input the name of student " << i + 1 << ": ";
    cin >> stu[i].name;
    cout << "Please input the 3 scores of student " << i + 1 << ": ";
    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) {
  cout << "\tName\tScore1\tScore2\tScore3\tAverage" << endl;
  for (int i = 0; i < n; ++i) {
    cout << setw(8) << stu[i].name << "\t"; /// XXX: cut off too long names to make the layout more
        compact
    cout << stu[i].score[0] << "\t\t" << stu[i].score[1] << "\t\t" << stu[i].score[2] << "\t\t";
    cout << stu[i].ave << endl;
  }
}

void sortstu(stuinfo stu[], int n) {
  sort(stu, stu + n, [](const stuinfo &a, const stuinfo &b) { return a.ave > b.ave; });
}

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

Listing 3: main.cpp

```cpp
#include <iostream>
#include <sstream>
#include <string>
#include "stuinfo.h"
using std::cout; using std::endl; using std::cin;
```

```cpp
using std::stringstream; using std::string;

int main() {
  stuinfo stu[3];
//  stringstream ss("Anne 100 100 100\nClay 80 80 80\nBob 90 90 90\n");
//  cin.rdbuf(ss.rdbuf());
  inputstu(stu, 3);
  showstu(stu, 3);
  sortstu(stu, 3);
  showstu(stu, 3);
  findstu(stu, 3, "Anne") ? cout << "Student Anne found\n" : cout << "Student Anne not found\n";
  findstu(stu, 3, "Ann") ? cout << "Student Ann found\n" : cout << "Student Ann not found\n";
  return 0;
}
```

Listing 4: CMakeLists.txt

```cmake
cmake_minimum_required(VERSION 3.18)
project(StuInfo)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_BUILD_TYPE Debug)

add_compile_options(-Wall -O2)
add_executable(StuInfo main.cpp stuinfo.cpp stuinfo.h)
```

编译与运行结果

Listing 5: 编译过程

```
# ls
CMakeLists.txt  main.cpp  stuinfo.cpp  stuinfo.h
# mkdir build
# cd build
# cmake ..
-- The C compiler identification is GNU 12.2.0
-- The CXX compiler identification is Clang 11.0.1
-- 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/stuinfo/build
# make
Scanning dependencies of target stuinfo
[ 33%] Building CXX object CMakeFiles/stuinfo.dir/main.cpp.o
/ws/stuinfo/main.cpp:16:21: warning: ISO C++11 does not allow conversion from string literal to 'char *'
      [-Wwritable-strings]
findstu(stu, 3, "Anne") ? cout << "Student Anne found" << endl : cout << "Student Anne not found" <<
    endl;
                ^
/ws/stuinfo/main.cpp:17:21: warning: ISO C++11 does not allow conversion from string literal to 'char *'
      [-Wwritable-strings]
findstu(stu, 3, "Donald") ? cout << "Student Donald found" << endl : cout << "Student Donald not found"
    << endl;
                ^
2 warnings generated.
[ 66%] Building CXX object CMakeFiles/stuinfo.dir/stuinfo.cpp.o
[100%] Linking CXX executable stuinfo
[100%] Built target stuinfo
```

运行结果

```
root@118e996adc3e:/ws/stuinfo/build# ./stuinfo
Please input the name of student 1: Anne
Please input the 3 scores of student 1: 100 100 100
Please input the name of student 2: Clay
Please input the 3 scores of student 2: 80 80 80
Please input the name of student 3: Bob
Please input the 3 scores of student 3: 90 90 90
        Name    Score1  Score2  Score3  Average
    Anne        100             100             100             100
    Clay        80              80              80              80
     Bob        90              90              90              90
        Name    Score1  Score2  Score3  Average
    Anne        100             100             100             100
     Bob        90              90              90              90
    Clay        80              80              80              80
Student Anne found
Student Donald not found
```

图 1: stuinfo

3 Types

3.1 问答题

3.1.1 static 的用法和作用

static关键字定义的变量成为静态变量。静态变量被存储在全局数据区中，其生命周期与程序的整个运行周期有关，在程序开始执行时被初始化，在程序结束时被销毁。对于定义在函数外的静态变量，其作用域为整个文件。对于函数内部的静态变量，它们**仅在第一次进入函数时进行初始化**，作用域仅限于该函数。静态变量可以用来限制变量作用域、避免重名、记录某些函数被调用的次数，或者在函数之间共享数据等。（lambda表达式中使用静态变量不需要声明捕获方式。）

使用static关键字定义的函数称为静态函数。静态函数只能在当前文件中使用，不能被其他文件调用。静态函数可以用来实现一些不想让其他文件访问的辅助函数，也可以避免与其他文件中的函数或全局变量产生命名冲突。

在类中使用static关键字定义的变量称为静态成员变量。静态成员变量不属于类的任何对象，而是属于整个类 (**注意：静态成员变量需要在类外定义，类中的只是声明**)。静态成员变量可以用来记录某个类的所有对象共享的数据，或者在类的所有对象之间共享某些资源等。当需要一个数据对象为整个类而非某个对象服务，同时又要保持封装性（即将此成员隐藏在类内部，对外不可见）时，可以将其定义为静态数据。

在类中使用static关键字定义的成员函数称为静态成员函数，它是一种只能在类内调用或通过类名调用的函数，与该类的任何对象都没有关系，也**不能访问任何非静态成员**。静态成员函数可以被用来在多个对象之间共享数据时，避免不必要的内存开销。

3.1.2 什么是隐式转换, 如何消除隐式转换

隐式转换是指编译器在某些情况下，自动地将一种数据类型转换为另一种数据类型的过程。

C++中有三种隐式转换：

1. 数字类型的隐式转换当一个表达式中包含不同的数字类型时，C++编译器将自动将其中一个数字类型转换为另一个数字类型，以便进行计算。总的来说，运算过程中的隐式转换总是将运算对象转换成运算成员中最宽的类型。

2. 派生类向基类的隐式转换派生类对象可以隐式地转换为基类对象，而不需要进行显式的类型转换。

3. 用户定义类型的隐式转换一些自定义的类存在特定的单参数构造函数、自定义类型转换函数、重载运算符等情况时，可以隐式地将一个自定义类型转换为另一个自定义类型。

隐式转换可能会导致代码的可读性和安全性受到影响，因此在编写代码时需要谨慎使用。在代码中过于依赖隐式类型转换可能会出现意料之外的问题。C++语言标准中规定，对于单个值，最多只能隐式应用一个用户定义的转换（构造函数或转换函数）。对于单个对象，如果需要超过一次类型转换，必须使用显式类型转换。

为了避免隐式类型转换，对于第一、二种情况，可以使用显式的转换typename(expression)来强制从一种类型转换到另一种类型，也可以使用static_cast<typename>(expression)来实现更广泛的类型转换（包括窄化转换、上下行转换），与此类似的还有const_cast、dynamic_cast。对于第三种情

况，可以显式地声明单参数构造函数为explicit来避免无用的复制构造（可能与编译器 RVO 优化特性有关）或禁止意料之外的语义错误。

3.2 程序解释

3.2.1 Code1

程序输出如下：

```
sum = -1825831516
fsum = 1.23457e+09
(fsum == f1) is 1
sum10x = 1
mul10x = 1
(sum10x == 1) is 0
(mul10x == 1) is 1
```

对于第1行，两个正数相加出现了负数，原因是两加数之和超出了数据类型的表示范围，发生了溢出。

计算机中的浮点数是以二进制表示的，并且只能存储有限位的数字。因此，有些浮点数无法精确表示，只能近似地表示。查看fsum与f1的十六进制数据，发现都为06 2c 93 4e（小端），说明在求和的过程中，低于加和最低表示精度的数据被丢弃，导致fsum与f1大小相等。

由于十进制数0.1无法在二进制中精确表示，因此在运算中，由于多次加法，两加数之间数量级差距加大，类似计算 fsum 时产生误差的原因，导致舍入误差累积，sum10x与真实结果之间的误差在一步步求和的过程中被放大。而计算mul10x时仅舍入一次，故误差较小。查看sum10x的十六进制数据为01 00 80 3f，mul10x的十六进制数据为00 00 80 3f，发现乘10的结果与预期结果相同，获得了比累加10次更小的误差。

3.2.2 Code2

程序输出如下：

```
f1 = 1.000000
f2 = 1.000000
f1 != f2
```

原因同3.2.1，由于无法精确浮点数0.1，且多次加法导致舍入误差累积，但误差又恰好在1e-6以内，因此打印出的十进制近似值相同，但是浮点数比较的结果不相等。

3.2.3 Code3

在 Linux 平台上，程序正常结束，输出结果如下：

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

```
0/0= 1026567984
```

在 Windows 平台上，程序未能打印出0/0的结果并异常退出，返回值0xC0000094，输出结果如下：

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

a = int(19.99 + 21.99) = int(41.98) = 41

b = int(19.99) + int(21.99) = 19 + 21 = 40

c = double(23 / 8) = double(2) = 2.0

d = double(23 / 8.0) = double(23.0 / 8.0) = 2.875

整形除 **0** 的结果是未定义的。

3.3 编程题

1,2,3,4 支持函数与自定义变量的计算器

Listing 6: calculator.cpp

```cpp
#include <iostream>
#include <cctype>
#include <algorithm>
#include <string>
#include <functional>
#include <stack>
#include <utility>
#include <vector>
#include <map>
#include <set>
#include <cmath>

using namespace std;

class Parser {
typedef long double ld;
public:
  explicit Parser(string exp = "") : expression(std::move(exp)) {
    expression.erase(remove_if(expression.begin(), expression.end(), ::isspace), expression.end());
  }
  void setExpression(const string &exp) {
    expression = exp;
    expression.erase(remove_if(expression.begin(), expression.end(), ::isspace), expression.end());
  }
  ld calculate() {
    if (expression.empty())
```

```
      throw invalid_argument("Empty expression");
    // 检查括号匹配
    int left = 0, right = 0;
    for (const char &c : expression) {
      if (c == '(') {
        left++;
      } else if (c == ')') {
        right++;
      }
    }
    if (left != right)
    throw invalid_argument("Mismatched brackets in expression: " + expression);
    if (std::count(expression.begin(), expression.end(), '=') == 1) {  // 赋值表达式
      auto pos = expression.begin();
      string var = ::move(getToken(pos));
      if (pos == expression.end() || *pos != '=') {
        throw invalid_argument("Invalid expression: " + expression);
      }
      ++pos;
      variables[var] = calc(pos);
      return variables[var];
    } else {  // 计算式
      auto pos = expression.begin();
      return calc(pos);
    }
  }
private:
  string expression;
  map<char, int> opt_priorities = {
    {'+', 1},
    {'-', 1},
    {'*', 2},
    {'/', 2},
    {'^', 3},
  };
  map<string, int> functions = {
    {"sqrt", 1},
    {"ln", 1},
    {"exp", 1},
    {"sin", 1},
    {"cos", 1},
    {"tan", 1},
    {"asin", 1},
    {"acos", 1},
    {"atan", 1},
    {"sinh", 1},
```

```
    {"cosh", 1},
    {"tanh", 1},
    {"asinh", 1},
    {"acosh", 1},
    {"atanh", 1},
    {"abs", 1},
    {"ceil", 1},
    {"floor", 1},
    {"round", 1},
    {"trunc", 1},
    {"log", 2},
    {"pow", 2},
};
map<string, ld> variables;
//    const string starting = "0123456789+-("; // 有效表达式开头
ld calc(string::iterator &pos) {
  stack<ld> operands;
  stack<char> opts;
  operands.push(0.0);  // 处理正负号开头情况
  if (pos == expression.end())
    throw invalid_argument("Invalid expression: " + expression);
  while (pos != expression.end()) {
    if (*pos == '(') {
      operands.push(calc(++pos));  // 结束递归调用后，pos指向')'或','
      pos++;
    } else if (*pos == ')' || *pos == ',') {
      break;
    } else if (isdigit(*pos) || *pos == '.') {  // 提取数字
      operands.push(strtol(pos));
    } else if (isOperator(*pos)) {  // 匹配加减乘除、幂函数单独处理
      char opt = *pos++;
      if (opt == '^') {  // 幂函数运算顺序自右向左，先入栈而不计算
        opts.push(opt);
        continue;
      }
      while (!opts.empty() && opt_priorities[opt] <= opt_priorities[opts.top()]) {
        calcOnce(operands, opts);
      }
      opts.push(opt);
    } else {
      string token = ::move(getToken(pos));  // 提取函数名，pos指向参数列表的左括号 或 提取变量名
      if (functions.find(token) != functions.end()) {  // 匹配函数
        calcFunc(token, pos, operands, opts);
        ++pos;
      } else if (variables.find(token) != variables.end()) {  // 匹配变量
        operands.push(variables[token]);
```

```
119         } else {
120           throw invalid_argument("Invalid token '" + token + "' in expression: " + expression);
121         }
122       }
123     }
124     while (!opts.empty())   // 完成栈中剩余运算符的计算
125     calcOnce(operands, opts);
126     return operands.top();
127   }
128   string getToken(string::iterator &pos) {
129     string token;
130     while (pos != expression.end() && isalpha(*pos))
131       token += *pos++;
132     return token;
133   }
134   inline ld strtol(string::iterator &pos) {
135     ld num = 0.0;
136     while (pos != expression.end() && isdigit(*pos))
137       num = num * 10 + (*pos++ - '0');
138     if (pos != expression.end() && *pos == '.') {
139       ++pos;
140       ld base = 0.1;
141       while (pos != expression.end() && isdigit(*pos)) {
142         num += base * (*pos - '0');
143         base /= 10;
144         ++pos;
145       }
146     }
147     return num;
148   }
149   inline bool isOperator(char c) {
150     return opt_priorities.find(c) != opt_priorities.end();
151   }
152   inline void calcOnce(stack<ld> &operands, stack<char> &opts) {
153     ld num2 = operands.top();
154     operands.pop();
155     ld num1 = operands.top();
156     operands.pop();
157     char top = opts.top();
158     opts.pop();
159     if (top == '+') {
160       operands.push(num1 + num2);
161     } else if (top == '-') {
162       operands.push(num1 - num2);
163     } else if (top == '*') {
164       operands.push(num1 * num2);
```

```
    } else if (top == '/') {
      if (num2 == 0) {
        throw invalid_argument("Divided by zero in expression: " + expression);
      }
      operands.push(num1 / num2);
    } else if (top == '^') {
      operands.push(pow(num1, num2));
    }
  }
  inline void calcFunc(const string &funcName, string::iterator &pos, stack<ld> &operands, stack<char> &
      opts) {
    ++pos;  // pos指向参数列表的第一个字符
    vector<ld> args;
    while (pos != expression.end() && *pos != ')') {  // 提取参数， pos指向参数列表后的右括号
      args.push_back(calc(pos));
      if (*pos == ',') ++pos;
    }
    if (args.size() != functions[funcName]) {  // 检查参数个数
      throw invalid_argument("Mismatched number of arguments in function '"
      + funcName + "' in expression: " + expression);
    }
    operands.push(func(funcName, args));
  }
  static inline ld func(const string &funcName, vector<ld> args) {
    if (funcName == "sqrt") {
      return sqrtl(args[0]);
    } else if (funcName == "ln") {
      return logl(args[0]);
    } else if (funcName == "sin") {
      return sinl(args[0]);
    } else if (funcName == "cos") {
      return cosl(args[0]);
    } else if (funcName == "tan") {
      return tanl(args[0]);
    } else if (funcName == "asin") {
      return asinl(args[0]);
    } else if (funcName == "acos") {
      return acosl(args[0]);
    } else if (funcName == "atan") {
      return atanl(args[0]);
    } else if (funcName == "sinh") {
      return sinhl(args[0]);
    } else if (funcName == "cosh") {
      return coshl(args[0]);
    } else if (funcName == "tanh") {
      return tanhl(args[0]);
```

```
    } else if (funcName == "asinh") {
      return asinhl(args[0]);
    } else if (funcName == "acosh") {
      return acoshl(args[0]);
    } else if (funcName == "atanh") {
      return atanhl(args[0]);
    } else if (funcName == "abs") {
      return fabsl(args[0]);
    } else if (funcName == "ceil") {
      return ceill(args[0]);
    } else if (funcName == "floor") {
      return floorl(args[0]);
    } else if (funcName == "round") {
      return roundl(args[0]);
    } else if (funcName == "trunc") {
      return truncl(args[0]);
    }else if (funcName == "log") {
      return logl(args[0]) / logl(args[1]);
    } else if (funcName == "pow") {
      return powl(args[0], args[1]);
    }
    return 0;
  }
};

int main() {
  string exp;
  Parser parser;
  while (getline(cin, exp)){
    parser.setExpression(exp);
    try {
      cout << parser.calculate() << endl;
    } catch (exception &e) {
      cerr << e.what() << endl;
    }
  }
  return 0;
}
```

效果如下：

```
-10 + sqrt(1+3)*5 + pow(2,3)/2*(4-2)*1 - 8 + 2^2^3
256
```

图 2: 计算器

要求 4，5 待实现

4 Structs

4.1 结构体对齐问题

内存对齐的规则：

1. 结构体第一个成员的偏移量（offset）为0，以后每个成员相对于结构体首地址的 offset 都是该成员大小与**有效对齐值**（max(默认对齐值，指定的对齐字节数)）中较小那个的整数倍，如有需要编译器会在成员之间填充字节。

2. 结构体的总大小为有效对齐值的整数倍，如有需要编译器会在最后一个成员之后填充字节。

当alignas指定的结构体对齐字节数小于默认对齐值（最大成员的大小）时，alignas失效，此时结构体内的成员任以默认对齐值对齐。当alignas指定的结构体对齐字节数大于自然对齐值时，alignas生效，以指定值对齐。同时，alignas指定的对齐字节数必须为 2 的非负整数幂。

4.2 Exercise

在struct Point中重载+、-、*、/运算符以简化后续坐标运算。

Listing 7: Geo2D.h

```h
#ifndef GEO2D_H
#define GEO2D_H

#include <iostream>
#include <cmath>

namespace Geo2D {
  struct Point {
    double x, y;
    friend std::ostream &operator<<(std::ostream &os, const Point &p);
    Point operator+(const Point &A) const;
    Point operator-(const Point &A) const;
    Point operator*(double k) const;
    Point operator/(double k) const;
  }; // 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

  bool  isTriangle  (const Point &A, const Point &B, const Point &C);
  Point intersection(const Line &L1, const Line &L2);  // 两直线交点
  std::pair<bool, Point> barycenter  (const Point &A, const Point &B, const Point &C);  // 重心
  std::pair<bool, Point> circumcenter(const Point &A, const Point &B, const Point &C);  // 外心
  std::pair<bool, Point> incenter    (const Point &A, const Point &B, const Point &C);  // 内心
  std::pair<bool, Point> orthocenter (const Point &A, const Point &B, const Point &C);  // 垂心
};

#endif //GEO2D_H
```

Listing 8: Geo2D.cpp

```
#include "Geo2D.h"

namespace Geo2D {
  std::ostream &operator<<(std::ostream &os, const Geo2D::Point &p) {
    os << "(" << p.x << ", " << p.y << ")";
    return os;
  }
  Point Point::operator+(const Point &A) const {
    return {x + A.x, y + A.y};
  }
  Point Point::operator-(const Point &A) const {
    return {x - A.x, y - A.y};
  }
  Point Point::operator*(double k) const {
    return {x * k, y * k};
  }
  Point Point::operator/(double k) const {
    return {x / k, y / k};
  }

  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);
  }
  Point intersection(const Line &L1, const Line &L2) {
    double a1 = L1.v.y, b1 = -L1.v.x, c1 = L1.v.x * L1.P.y - L1.v.y * L1.P.x;
    double a2 = L2.v.y, b2 = -L2.v.x, c2 = L2.v.x * L2.P.y - L2.v.y * L2.P.x;
    double det = a1 * b2 - a2 * b1;
```

```
  if (fabs(det) < EPS) {
    std::cerr << "Lines: " << L1.P << " + t * " << L1.v << " and ";
    std::cerr << L2.P << " + t * " << L2.v << " are parallel.\n";
    return 0;
  }
  return {(b1 * c2 - b2 * c1) / det, (a2 * c1 - a1 * c2) / det};
}

std::pair<bool, Point> barycenter(const Point &A, const Point &B, const Point &C) {
  if (!isTriangle(A, B, C)) {
    std::cerr << "Cannot construct a triangle with Points: ";
    std::cerr << A << ", " << B << ", " << C << ".\n";
    return {false, 0};
  }
  return {true, (A + B + C) / 3};
}
std::pair<bool, Point> circumcenter(const Point &A, const Point &B, const Point &C) {
  if (!isTriangle(A, B, C)) {
    std::cerr << "Cannot construct a triangle with Points: ";
    std::cerr << A << ", " << B << ", " << C << ".\n";
    return {false, 0};
  }
  Line L1 = {(A + B) / 2, {B.y - A.y, A.x - B.x}};
  Line L2 = {(B + C) / 2, {C.y - B.y, B.x - C.x}};
  return {true, intersection(L1, L2)};
}
std::pair<bool, Point> incenter(const Point &A, const Point &B, const Point &C) {
  if (!isTriangle(A, B, C)) {
    std::cerr << "Cannot construct a triangle with Points: ";
    std::cerr << A << ", " << B << ", " << C << ".\n";
    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 + B * b + C * c) / (a + b + c)};
}
std::pair<bool, Point> orthocenter(const Point &A, const Point &B, const Point &C) {
  if (!isTriangle(A, B, C)) {
    std::cerr << "Cannot construct a triangle with Points: ";
    std::cerr << A << ", " << B << ", " << C << ".\n";
    return {false, 0};
  }
  Line L1 = {A, {B.y - C.y, C.x - B.x}};
  Line L2 = {B, {C.y - A.y, A.x - C.x}};
  return {true, intersection(L1, L2)};
```

```
  }
}
```

在 CMakeList.txt 中设置动态链接库导出并链接dl库以实现动态加载。

Listing 9: CMakeLists.txt

```
cmake_minimum_required(VERSION 3.18)
project(project1)
set(CMAKE_CXX_STANDARD 20)
set(SRC Geo2D.cpp)
add_compile_options(-Wall)
add_library(Geo2D SHARED ${SRC})
add_executable(project1 main.cpp)
target_link_libraries(project1 dl)
target_link_libraries(project1 Geo2D)
```

编译后，CMake 导出了动态链接库 libGeo2D.so。在终端执行nm -D libGeeo2D.so查看动态链接库中的函数符号。

```
# nm -D libGeo2D.so
                 w _ITM_deregisterTMCloneTable
                 w _ITM_registerTMCloneTable
00000000000016b0  T _ZN5Geo2D10barycenterERKNS_5PointES2_S2_
0000000000001410  T _ZN5Geo2D10isTriangleERKNS_5PointES2_S2_
0000000000001ca0  T _ZN5Geo2D11orthocenterERKNS_5PointES2_S2_
00000000000017f0  T _ZN5Geo2D12circumcenterERKNS_5PointES2_S2_
0000000000001490  T _ZN5Geo2D12intersectionERKNS_4LineES2_
00000000000019e0  T _ZN5Geo2D8incenterERKNS_5PointES2_S2_
0000000000001260  T _ZN5Geo2DlsERSoRKNS_5PointE
00000000000013c0  T _ZNK5Geo2D5PointdvEd
0000000000001320  T _ZNK5Geo2D5PointmiERKS0_
0000000000001370  T _ZNK5Geo2D5PointmlEd
00000000000012d0  T _ZNK5Geo2D5PointplERKS0_
                 U _ZNSolsEd@GLIBCXX_3.4
0000000000001e30  W _ZNSt4pairIbN5Geo2D5PointEEC2IbLb1EEEOT_RKS1_
0000000000001e80  W _ZNSt4pairIbN5Geo2D5PointEEC2IbS1_Lb1EEEOT_OT0_
                 U _ZNSt8ios_base4InitC1Ev@GLIBCXX_3.4
                 U _ZNSt8ios_base4InitD1Ev@GLIBCXX_3.4
                 U _ZSt4cerr@GLIBCXX_3.4
0000000000001ef0  W ZSt7forwardIN5Geo2D5PointEEOT_RNSt16remove_referenceIS2_E4typeE
0000000000001ee0  W _ZSt7forwardIbEOT_RNSt16remove_referenceIS0_E4typeE
                 U _ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc@GLIBCXX_3.4
                 U __cxa_atexit@GLIBC_2.2.5
                 w __cxa_finalize@GLIBC_2.2.5
                 w __gmon_start__
                 U acos@GLIBC_2.2.5
                 U sqrt@GLIBC_2.2.5
```

在主函数中动态加载 libGeo2D.so 库并获取函数指针，使用一个简单的等边三角形的例子来测试。

Listing 10: main.cpp

```cpp
#include <iostream>
#include <cmath>
#include <algorithm>
#include <dlfcn.h>
#include "Geo2D.h"

using Geo2D::Point;
using Geo2D::O;

int main() {
  void* handle = dlopen("libGeo2D.so", RTLD_LAZY);  // 加载动态链接库
  if (!handle) {
    std::cerr << dlerror() << '\n';
    return 1;
  }
  Point A{sqrt(3), 1}, B{sqrt(3), -1};
  typedef std::pair<bool, Point> (*point_t)(const Point&, const Point&, const Point&);
  // 通过符号获取函数指针
  auto barycenter   = (point_t)dlsym(handle, "_ZN5Geo2D10barycenterERKNS_5PointES2_S2_");
  auto circumcenter = (point_t)dlsym(handle, "_ZN5Geo2D12circumcenterERKNS_5PointES2_S2_");
  auto incenter     = (point_t)dlsym(handle, "_ZN5Geo2D8incenterERKNS_5PointES2_S2_");
  auto orthocenter  = (point_t)dlsym(handle, "_ZN5Geo2D11orthocenterERKNS_5PointES2_S2_");
  if (!barycenter || !circumcenter || !incenter || !orthocenter) {
    std::cerr << dlerror() << '\n';
    return 1;
  }

  auto [flag1, x] = barycenter(A, B, O);
  auto [flag2, y] = circumcenter(A, B, O);
  auto [flag3, z] = incenter(A, B, O);
  auto [flag4, w] = orthocenter(A, B, O);
  flag1 ? std::cout << "Barycenter: "   << x << "\n" : std::cout << "No barycenter\n";
  flag2 ? std::cout << "Circumcenter: " << y << "\n" : std::cout << "No circumcenter\n";
  flag3 ? std::cout << "Incenter: "     << z << "\n" : std::cout << "No incenter\n";
  flag4 ? std::cout << "Orthocenter: "  << w << "\n" : std::cout << "No orthocenter\n";
  dlclose(handle);
  return 0;
}
```

```
Barycenter: (1.1547, 0)
Circumcenter: (1.1547, -0)
Incenter: (1.1547, 0)
Orthocenter: (1.1547, -0)
```

可以看到，代码成功链接了共享库并动态加载了函数，输出结果符合预期。

5 C++ 动态内存申请

5.1 C++ 的内存分区

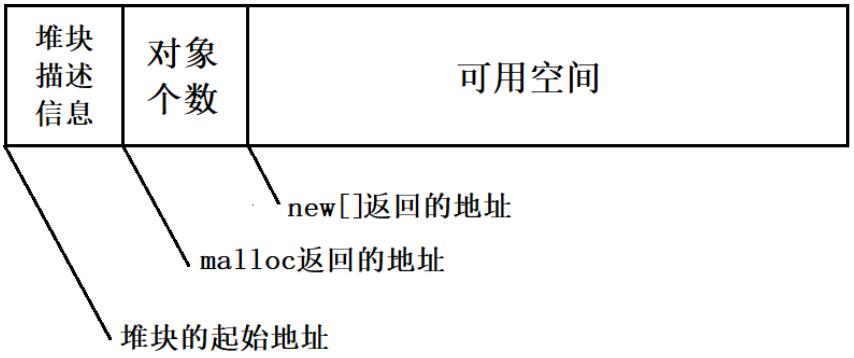
```cpp
class A {
  int num;
};

static int a;  // (1) 静态变量，全局/静态存储区
auto b = 0;  // (2) 全局变量，全局/静态存储区：

int main() {
  char s[] = "abc";  // (3) s在栈区，"abc"在常量存储区
  char *p = "123456";  // (4) p在栈区，"123456"在常量存储区
  p1 = (char *) malloc(10);  // (5) 堆区
  A *a = new A();  // (6) 自由存储区
}
```

5.2 问答题

1. new 和 malloc 的区别 new是 C++ 提供的一个运算符，而malloc是 C 语言提供的一个库函数。new运算符可以自动计算出所需内存空间占用的字节大小，无需用户手动计算，而malloc需要用户提供需要的字节数。new运算符返回的是指定类型的指针，而malloc函数返回的是void *类型的指针，需要用户使用强制类型转换手动转换。new在动态分配内存的时候可以初始化对象，调用其构造函数，失败时抛出std::bad_alloc，并在delete释放内存时调用对象的析构函数，而malloc只分配一段给定大小的内存，并返回该内存首地址指针，失败时返回nullptr。此外，new运算符可以重载，其分配的内存需要使用delete运算符进行释放，而malloc分配的内存则需要使用free函数进行释放。

2. delete p、delete[] p、allocator 都有什么作用 delete p用于释放由new分配的单个对象的内存。delete[] p用于释放由 new[]分配的数组对象的内存。C++ 中的allocator是一个内存分配器，用于管理STL容器的动态内存分配和释放。

3. malloc 申请的存储空间能用 delete 释放吗 原则上不建议这么做，malloc申请的空间应当使用free释放。经过尝试，发现在对基础类型操作时，代码看起来正常运行，然而对自定义结构体进行操作时，malloc分配的连续内存空间在通过delete[]释放时可能会因为错误读取对象个数而出现内存超界访问，这与new[] 和 malloc 实现方式之间的差异有关。



4. malloc 与 free 的实现原理 malloc的是通过系统调用实现的。操作系统中的堆管理器维护了一个内存块列表，每个内存块包含一个头部信息和实际的数据区域。头部信息包含了该内存块的大小和状态等信息。当调用malloc函数时，堆管理器会遍历内存块列表，查找一个大小足够的空闲内存块。如果找到了空闲内存块，则将该内存块标记为已分配状态，并返回其起始地址。如果没有找到足够大小的空闲内存块，则会向操作系统请求更多的内存空间，并将新分配的内存块添加到内存块列表中。由于存在头部信息，实际占用的空间会高于申请的空间。free函数的作用是释放先前由malloc或calloc函数分配的内存空间。在调用free函数时，堆管理器会检查被释放的内存块是否处于已分配状态，并将其标记为可用状态。如果相邻的空闲内存块存在，则会将它们合并成一个更大的空闲内存块，以提高内存的利用率。

5.3 Exercise

源代码如下：

Listing 11: RandomGenerator.h

```
#ifndef RANDOMGENERATOR_H
#define RANDOMGENERATOR_H

#include <cstdint>

class RandomGenerator {
public:
  RandomGenerator();
  void reserve(uint32_t size);  // 用C++动态内存分配n个整数空间
  void randomize();             // 用随机数初始化这n个空间，随机数范围为 [0,n-1]
  void printMinMax();           // 输出这n个空间的最大和最小数值
  void freeup();                // 释放这n个空间
  ~RandomGenerator();
private:
  int *data;
  int size;
};

```

```
#endif //RANDOMGENERATOR_H
```

Listing 12: RandomGenerator.cpp

```
#include "RandomGenerator.h"
#include <random>
#include <iostream>

RandomGenerator::RandomGenerator(): data(nullptr), size(0) {}
RandomGenerator::~RandomGenerator() {
  delete[] data;
  data = nullptr;
}

void RandomGenerator::reserve(uint32_t size) {
  try {
    data = new int[size];
  }
  catch (std::bad_alloc &e) {  // 内存分配失败
    std::cerr << e.what() << std::endl;
    return;
  }
  this->size = size;
}

void RandomGenerator::randomize() {
  if (!data || !size) {
    std::cerr << "data is not initialized" << std::endl;
    return;
  }
  std::random_device rd;
  std::mt19937 gen(rd());
  std::uniform_int_distribution<> distrib(0, size - 1);
  for (int i = 0; i < size; ++i) {
    data[i] = distrib(gen);
  }
  /// ref: https://en.cppreference.com/w/cpp/numeric/random/uniform_int_distribution
}

void RandomGenerator::printMinMax() {
  if (!data || !size) {
    std::cerr << "data is not initialized" << std::endl;
    return;
  }
  int min = data[0], max = data[0];
  for (int i = 1; i < size; ++i) {
    min = min < data[i] ? min : data[i];
```

```
    max = max > data[i] ? max : data[i];
  }
  std::cout << "max: " << max << ", min: " << min << std::endl;
}

void RandomGenerator::freeup() {
  delete[] data;
  data = nullptr;
  size = 0;
}
```

Listing 13: main.cpp

```
#include <iostream>
#include "RandomGenerator.h"

int main() {
  RandomGenerator rg;
  int n;
  std::cin >> n;
  rg.reserve(n);
  rg.randomize();
  rg.printMinMax();
  rg.freeup();
  return 0;
}
```

CMakeList.txt 如下：

Listing 14: CMakeLists.txt

```
cmake_minimum_required(VERSION 3.18)
project(project1)
set(CMAKE_CXX_STANDARD 17)
add_executable(project1 main.cpp RandomGenerator.h RandomGenerator.cpp)
```

运行截图:

```
root@118e996adc3e:/ws/project2/cmake-build-debug# ./project1
10000
max: 9998, min: 0
```

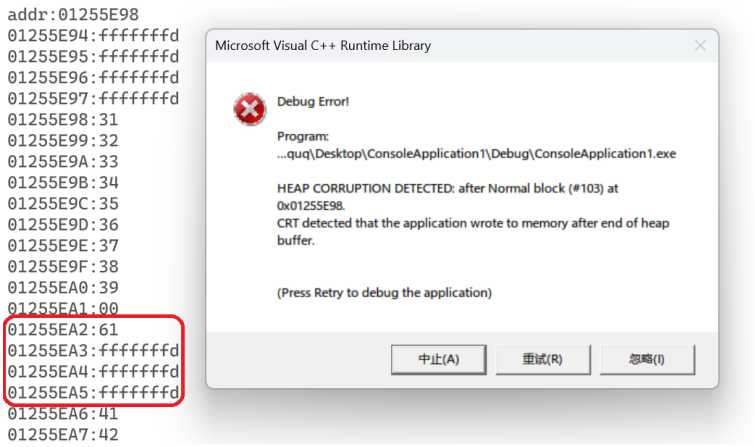
图 3: RandomGenerator

6 Debug 和 Release

6.2 如何判断动态申请越界（C）

VS Debug 模式 在 Debug 模式下，VS 中的CRT（C 运行时库）调试堆会检测内存泄漏与缓冲区溢出。对堆函数（例如 `malloc`、`free`、`calloc`、`realloc`、`new`和`delete`）的所有调用均会被解析为这些函数在调试堆中运行的调试版本。例如，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。

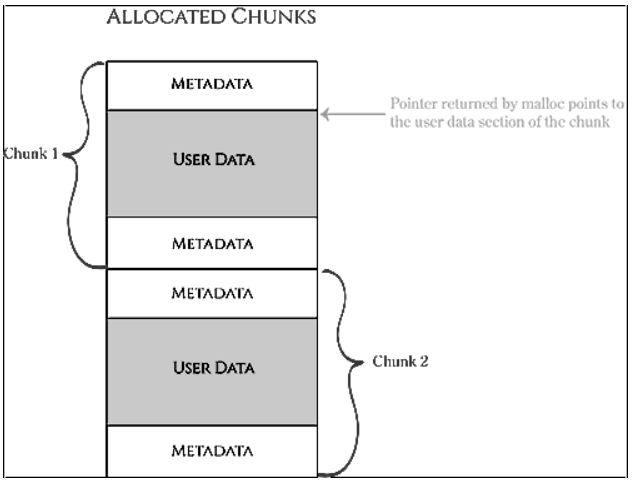
第 3 种情况中，代码恰好修改了用户数据后的`no_mans_land`块，导致在调用`free`释放内存块时检测到意外改写，因而提示错误`HEAP CORRUPTION DETECTED`，告知代码出现了越界写入的情况。



在第 1、2、4 种情况中，由于不满足同时覆写`no_mans_land`块（并改写其中的已知值）和释放内存块，所以代码表现出正常运行，并且不会给出越界错误的提示。

Linux gdb 经过测试，在 linux 环境下使用`gdb`进行调试，4 种情况都能正常运行。

在调用 malloc 请求内存时，堆管理器不仅要找到一个足够大的区域，还需要存储关于分配的元数据（metadata）。堆管理器还需要确保在 32 位系统上分配是 8 字节对齐，在 64 位系统上分配是 16 字节对齐。因此，堆管理器会查找或创建一个新块以容纳所需大小加上元数据和填充（`padding bytes`）用于对齐目标所需空间大小之后得到最终实际占用空间大小，并将该块标记为“已分配”，然后返回指向该块中已经按照要求进行过地址调整并且长度符合用户期望值（即去掉了元数据及填充之后剩余可供用户自由利用部份）的指针作为`malloc`调用结果传递给应用层代码执行相应操作。



典型的元数据位于用户数据的前一个字节，包含数据段大小与分配状态的信息。依此，我们可以构造出下面的代码损坏内存块头部的元数据，使`free`函数抛出`invalid pointer`异常：

Listing 15: C_free_error.c

```c
#include <stdlib.h>

int main() {
  char *p = (char *)malloc(sizeof(char) * 10);
  p[-8] &= 0x0f;  // 损坏元数据
  free(p);  // 抛出 invalid pointer
  return 0;
}
```

因此，我们可以猜测，Linux 中并没有像 VS Debug 模式类似的基于缓冲区的越界检测机制，如果代码出现越界并恰好损坏了内存块头部的元数据部分，那么在调用`free`释放空间时会发生错误。

VS Release 在 VS 的 Release 模式下，4 种情况的代码也都能正常运行，而损坏元数据的代码在运行后同样抛出异常。可以推测 Release 模式下与 Linux 环境类似，在动态分配空间时也**不会进行初始化**与越界检测，仅当使用被损坏的堆空间时抛出异常。

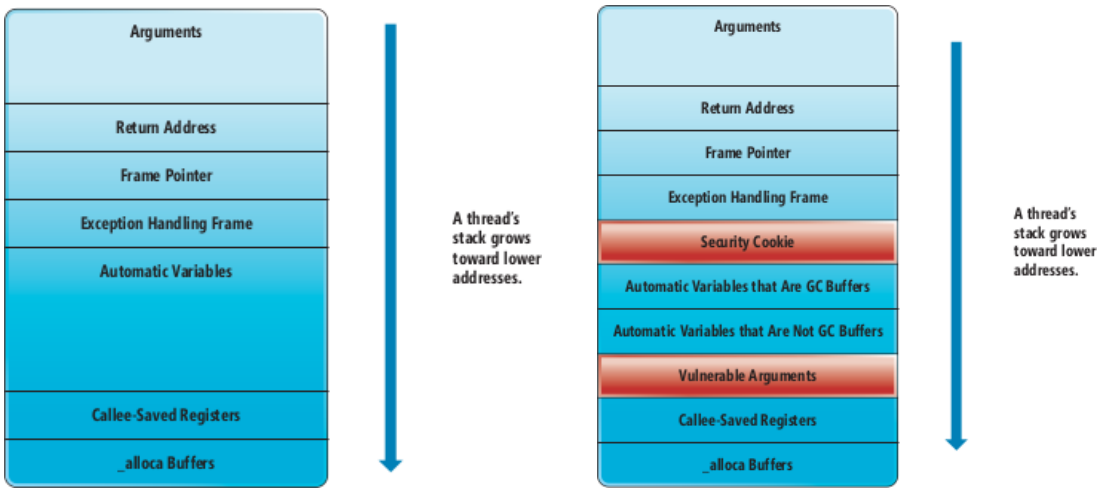
6.3 如何判断动态申请越界（C++）

对于 C++，在 VS 的 Debug、Release 与 Linux `gdb`环境下，越界代码的表现同 C 语言类似，所有情况中只有在 VS 的 Debug 模式下 (1)(3) 放开，(2) 注释，使代码改写`no_mans_land`段而后释放内存块提示了越界错误。具体情况可以参照6.2 如何判断动态申请越界（C），这里不再赘述。

6.4 如何判断普通数组的越界访问

VS 提供了面向基于堆栈的缓冲区保护的支持。一种机制是运行时错误检查的一部分，启用该机制时，编译器以交错方式从堆栈中分配额外的小内存块，以使堆栈中的每个局部变量夹在两个这样的块之间。每个这样的附加块都填充一个特殊值（`0xCC`）。在运行时，函数会检查这些块是否有损坏并报告潜

在的缓冲区溢出或下溢。另一种机制称为 GuardStack，是一种堆栈越界访问的动态缓解机制。它通过在在局部变量的上方分配一个称为 cookie（也称为 canary）的随机值，并在函数返回时验证该 Cookie 的值是否被篡改。如果 Cookie 的值被篡改，则表示发生了缓冲区溢出，程序会立即终止。



(a) 典型的 x86 堆栈帧　　(b) 使用 GuardStack 保护的典型 x86 堆栈帧

栈缓冲区溢出（超出缓冲区上限的写入）可能会覆盖存储在缓冲区上方的任何代码或数据指针。堆栈缓冲区下溢（在缓冲区下限之下的写入）可能会覆盖被调用方保存的寄存器的值（也可能是代码或数据指针）。任意的越界写入将导致应用程序崩溃或以未定义的方式运行。

我们通过下面的代码测试越界写入带来的后果。

Listing 16: Test_Out_Of_Bound.cpp

```cpp
#include <iostream>
#include <iomanip>
#include <cstring>
#include <cstdio>
using namespace std;

template <typename T, int N>
class OutOfBoundsTest {
public:
  OutOfBoundsTest() : n(N) {}
  ~OutOfBoundsTest() {}
  void display() {  // 从分配内存的前8个数据类型大小位置开始，打印到向后8个
    for (int i = -8; i < n + 8; i++) {
      cout << ((i >= 0 && i < n) ? "+ " : "  ") << dec << setw(2) << i << " ";
      cout << hex << static_cast<void*>(data + i) << ": " << int(data[i]) << endl;
    }
  }
  void clear() {
    memset(data, 0, sizeof(T) * N);
  }
```

```
21    void set(int i, T value) {
22      cout << dec << "Setting " << i << " to " << hex << int(value);
23      if (i < 0 || i >= n) {
24        cout << " (out of bounds)";
25      }
26      cout << endl;
27      data[i] = value;
28    }
29  private:
30    T data[N];
31    int n;
32  };
33
34  int main() {
35    OutOfBoundsTest<int, 10> a;
36    //  OutOfBoundsTest<char, 10> a;
37    a.display();
38    a.set(12, 0x77);  // 溢出
39    //  a.set(-1, 0x77);  // 下溢
40    //  XXX: 在此添加更多越界测试代码
41    a.display();
42    getc(stdin);
43    return 0;
44  }
```

在 VS 的 Debug 模式下运行上面的代码，使用`int`型数组，在发生溢出时，GuardStack 检测到堆栈缓冲区溢出；发生下溢时，运行时错误检查提示栈损坏。对于`char`型数组，测试结果与`int`型数组类似。

(a) 溢出　　(b) 下溢

在 VS 的 Release 模式下，由于未对栈空间初始化以进行运行时错误检查，对栈溢出的检测较于 Debug 模式更加宽松，但是发生致命的栈溢出时依然会抛出错误，上溢出时也同样可能触发 GuardStack。

在 Linux `gdb`命令行中，下溢会抛出`SIGSEGV`，上溢出可能会覆盖代码或数据导致未定义的行为，也可能抛出`SIGSEGV`。

```
  -8 0x7ffc21fbe9f0: 11bf
  -7 0x7ffc21fbe9f4: 0
  -6 0x7ffc21fbe9f8: 0
  -5 0x7ffc21fbe9fc: 0
  -4 0x7ffc21fbea00: 21fbea50
  -3 0x7ffc21fbea04: 7ffc
  -2 0x7ffc21fbea08: 401207
  -1 0x7ffc21fbea0c: 0
+  0 0x7ffc21fbea10: 2
+  1 0x7ffc21fbea14: 0
+  2 0x7ffc21fbea18: 4017e5
+  3 0x7ffc21fbea1c: 0
+  4 0x7ffc21fbea20: 0
+  5 0x7ffc21fbea24: 0
+  6 0x7ffc21fbea28: 0
+  7 0x7ffc21fbea2c: 0
+  8 0x7ffc21fbea30: 4017a0
+  9 0x7ffc21fbea34: 0
  10 0x7ffc21fbea38: a
  11 0x7ffc21fbea3c: 0
  12 0x7ffc21fbea40: 21fbeb40
  13 0x7ffc21fbea44: 7ffc
  14 0x7ffc21fbea48: 0
  15 0x7ffc21fbea4c: 0
  16 0x7ffc21fbea50: 4017a0
  17 0x7ffc21fbea54: 0
Setting -1 to 77 (out of bounds)

Program received signal SIGSEGV, Segmentation fault.
```

(a) signal SIGSEGV

```
Setting 10 to 77 (out of bounds)
Setting 11 to 77
Setting 12 to 77
  -8 0x7ffe96914260: 77
  -7 0x7ffe96914264: c
  -6 0x7ffe96914268: 0
  -5 0x7ffe9691426c: 0
  -4 0x7ffe96914270: 969142c0
  -3 0x7ffe96914274: 7ffe
  -2 0x7ffe96914278: 401255
  -1 0x7ffe9691427c: 0
+  0 0x7ffe96914280: 2
+  1 0x7ffe96914284: 0
+  2 0x7ffe96914288: 401655
+  3 0x7ffe9691428c: 0
+  4 0x7ffe96914290: 0
+  5 0x7ffe96914294: 0
+  6 0x7ffe96914298: 0
+  7 0x7ffe9691429c: 0
+  8 0x7ffe969142a0: 401610
+  9 0x7ffe969142a4: 0
+ 10 0x7ffe969142a8: 77
+ 11 0x7ffe969142ac: 77
+ 12 0x7ffe969142b0: 77
+ 13 0x7ffe969142b4: 7ffe
+ 14 0x7ffe969142b8: 0
+ 15 0x7ffe969142bc: 0
+ 16 0x7ffe969142c0: 401610
+ 17 0x7ffe969142c4: 0
+ 18 0x7ffe969142c8: 620d0d0a
+ 19 0x7ffe969142cc: 7fc8
+ 20 0x7ffe969142d0: 969143b8
+ 21 0x7ffe969142d4: 7ffe
+ 22 0x7ffe969142d8: 11bf
```

(b) 上溢出导致未定义行为

6.5 总结

内存越界访问有时是为了测试代码健壮性或安全性而人为创造的，但是大部分情况下，内存越界访问都是由于代码的疏忽或漏洞而意外产生的。意外的内存越界访问会导致程序崩溃、数据丢失，甚至是安全漏洞。

操作系统、编译器都针对内存越界访问问题做出了一系列预防与缓解措施。例如 Windows 系统中的`DEP`（数据执行防护），Linux 系统中的`ASLR`（地址空间随机化）。编译器在不同的编译模式（Debug 和 Release）下也对内存越界访问有不同的检测与应对措施。在 Debug 模式下，编译器通常会包含额外的调试信息，并通过初始化内存和开启一些额外的安全检查来检测内存越界访问等问题，但是这可能会导致代码性能下降。在 Release 模式下，编译器会关闭一些安全检查和调试信息并优化代码以提高程序的性能。这意味着，内存越界访问等问题可能不会被及时检测和报告，从而导致程序崩溃或产生难以预料的行为，定位错误的难度也可能会更高。

我们在编写程序时，应当养成良好的习惯以防范越界。例如：始终检查程序的输入是否合法，使用`STL`库中的容器替代普通数组，使用基于范围的循环、引用、迭代器或智能指针等替代换传统`for`循环与传统指针，减少手动内存分配，在代码中减少魔法数的使用等等。我们也需要学习使用分析工具，及时测试代码，通过边界值、非法值测试代码的潜在错误。总之，在编写程序时，我们应当时刻保有一颗对内存越界警觉的心。