|  |  |
| --- | --- |
| **编号:** | **第1章第1次** |

****

信息科学与工程学院实验报告

《面向对象程序设计》

**Object-Oriented Programming**

|  |  |
| --- | --- |
| 姓名： | 颜丙超 |
| 学号： | 202311000415 |
| 班级： | 计工本2301 |
| 教师: | 张庆科 |
| 时间： | 2024年10月27日 |

**《面向对象程序设计》实验报告**

**报告要求：**实验报告包含实验目的、实验内容、实验过程（详细操作流程）、实验结果（程序运行结果高清截图）、实验分析总结五个部分。报告中若涉及代码程序，请在附录部分提供完整程序源码及源码托管地址(基于Highlight软件导入源码)。报告撰写完毕后请将PDF格式版本上传到坚果云作业提交系统。

1. **实验目的**
2. **理解指针和引用的区别与联系**
3. **掌握C++指针和引用的基本方法**
4. **理解引用作为形式参数和返回类型的用法**
5. **掌握C++动态内存申请和释放的基本用法**
6. **熟练运用VISUAL STUDIO进行算法代码的调试**

**二、实验内容**

1.实验一请采用C++语言编写重载函数VOID SWAP(...),实现对任意两个相同数值型数据的交换。

2.阅读分析代码

3.动态内存分配

4.综合编程

**三、实验过程**

**实验一**：这个C++程序展示了两种交换两个整数值的方法：一种是使用引用传递，另一种是使用指针传递。下面是对程序各部分的详细分析：

头文件包含：

cpp

#include<iostream>

这行代码包含了C++标准库中的输入输出流库，使得程序可以使用cin和cout进行输入输出操作。

命名空间声明：

cpp

using namespace std;

这行代码使得程序可以直接使用std命名空间中的所有成员，而不需要在它们前面加上std::前缀。

swap函数（使用引用）：

cpp

void swap(int& a, int& b)

{

int t = a;

a = b;

b = t;

}

这个函数接受两个整数的引用作为参数。通过引用传递，函数内部对参数的修改会影响到原始变量。函数内部使用一个临时变量t来交换a和b的值。

swap函数（使用指针）：

cpp

void swap(int\* a, int\* b)

{

int t = \*a;

\*a = \*b;

\*b = t;

}

这个函数接受两个指向整数的指针作为参数。通过解引用指针（使用\*），函数可以访问并修改指针指向的整数值。同样，这个函数也使用一个临时变量t来交换两个指针指向的整数值。

main函数：

cpp

int main()

{

int a, b;

cout << "请输入两个数字";

cin >> a >> b;

swap(a, b);

cout << a << "\n" << b << "\n";

int x, y;

cout << "请输入两个数字";

cin >> x >> y;

swap(&x, &y);

cout << x << "\n" << y;

}

首先，程序提示用户输入两个整数a和b，然后调用使用引用的swap函数来交换这两个数的值，并打印交换后的结果。

接着，程序再次提示用户输入两个整数x和y，但这次调用使用指针的swap函数来交换这两个数的值，并打印交换后的结果。

**实验二：这个程序试图通过多个方面来分析C++中变量传递、引用、以及动态内存管理的行为。然而，程序中包含了一些误导性的操作和概念混淆，特别是在function函数和动态内存管理部分。以下是对程序各部分的详细分析：**

**1. function 函数分析**

**cpp**

**int& function(int v, int\* p, int& r)**

**{**

**v = (\*p)++;**

**\*p = 100;**

**p = new int; // 注意：这里创建了一个新的动态内存，但之前的指针p所指向的内存泄露了！**

**\*p = 1000;**

**r = 2 + (\*p)++;**

**cout << "\*p = " << \*p << endl;**

**cout << "请输入一个值";**

**cin >> r;**

**return v; // 注意：返回的是局部变量v的引用，这是未定义行为！**

**}**

**v 是通过值传递的，所以函数内部对v的修改不会影响外部变量。**

**p 是一个指向整数的指针，函数内部通过解引用\*p来修改它所指向的值。然而，当p = new int;执行时，p被重新指向了一个新的动态分配的整数，而之前p所指向的内存地址被泄露（因为没有相应的delete操作）。**

**r 是一个引用传递，所以函数内部对r的修改会影响外部变量。**

**函数返回v的引用是未定义行为，因为v是一个局部变量，它在函数返回后会被销毁。**

**2. 主函数分析**

**2.1 传值，传址和传引用分析**

**cpp**

**cout << "a = " << a << ", b = " << b << ", c = " << c << endl;**

**function(a, &b, c);**

**cout << "a = " << a << ", b = " << b << ", c = " << c << endl;**

**调用function后，a的值不变，因为a是按值传递的。**

**b的值在function内部被修改为100（第一次），然后又被修改为1001（由于\*p++操作，但注意p指针已经改变指向）。然而，由于p指针的重新分配和泄露，外部变量b的最终值取决于function内部p指针重新分配之前的操作，即b最终为100（因为\*p = 100;后p的指向改变了，后续对\*p的修改不影响b）。**

**c的值被修改为用户输入的值加上2（如果考虑\*p++的第一次修改，但同样由于p的重新分配，这里的分析变得复杂且不准确）。然而，由于r = 2 + (\*p)++;和随后的用户输入，c的最终值将是用户输入的值。**

**2.2 引用规律分析**

**cpp**

**int& r = a;**

**r = a + b + c;**

**a = r + a + b + c;**

**r = b;**

**2.3 局部变量存储分析**

**这部分代码简单地打印了变量a、b、c的地址，用于说明它们在内存中的位置。**

**2.4 动态内存申请分析**

**cpp**

**int\* p1 = new int{ 10 };**

**int\* p2 = new int[10] {10, 20, 30};**

**cout << "变量p1地址：" << &p1 << endl;**

**cout << "变量p2地址：" << &p2 << endl;**

**cout << "变量p1指向的地址：" << p1 << endl;**

**cout << "变量p2指向的地址：" << p2 << endl;**

**p1和p2是指针变量，它们存储了动态分配内存的地址。**

**&p1和&p2是这些指针变量自身的地址，而不是它们所指向的动态内存的地址。**

**p1和p2分别指向动态分配的单个整数和整数数组的起始地址。**

**注意：程序中有一个内存泄露的问题，即在function内部为p重新分配内存后，没有释放之前p所指向的内存。此外，function返回局部变量的引用也是未定义行为。**

**实验三：1. 生成随机数组**

**函数generateRandomArray负责生成一个包含size个随机整数的数组，这些整数的值在minVal和maxVal之间（包括这两个值）。**

**参数：**

**size：数组的大小。**

**minVal：数组元素的最小值。**

**maxVal：数组元素的最大值。**

**过程：**

**使用new操作符动态分配一个大小为size的整数数组。**

**通过循环，为每个数组元素生成一个随机值。这个随机值是通过minVal + std::rand() % (maxVal - minVal + 1)计算得到的，它确保了随机值在minVal和maxVal之间。**

**返回指向新分配数组的指针。**

**2. 寻找最大子序列和**

**函数findMaxSubsequenceSum使用Kadane算法来寻找数组中具有最大和的子序列（注意：这里说的是子序列，而不是子数组，即元素可以是不连续的，但在这个特定实现中，由于算法的特性，它实际上找到的是最大和的子数组）。然而，根据代码实现和常见的术语混淆，我们在这里将其解释为最大和的子数组。**

**参数：**

**arr：指向整数数组的指针。**

**size：数组的大小。**

**过程：**

**初始化两个变量max\_current和max\_global为数组的第一个元素。max\_current用于跟踪当前子数组的和，而max\_global用于记录迄今为止找到的最大子数组和。**

**遍历数组（从第二个元素开始），对于每个元素，更新max\_current为当前元素和max\_current + 当前元素中的较大值。这一步决定了是否将当前元素添加到当前子数组中，还是开始一个新的子数组。**

**如果max\_current大于max\_global，则更新max\_global。**

**遍历结束后，max\_global包含最大子数组的和。**

**3. 主函数**

**过程：**

**使用std::srand(std::time(0))初始化随机数生成器，以确保每次运行程序时都能得到不同的随机序列。**

**设置数组的大小、最小值和最大值。**

**调用generateRandomArray生成随机数组，并打印出来。**

**调用findMaxSubsequenceSum找到最大子数组和，并打印出来。**

**使用delete[]释放动态分配的数组内存。**

**注意事项**

**尽管函数名findMaxSubsequenceSum暗示寻找的是子序列的和，但实现实际上是寻找最大和的子数组。在Kadane算法中，子序列和子数组的区别并不明显，因为算法本身是通过连续添加元素来构建最大和的子数组的。然而，在更严格的数学或算法上下文中，这两者是有区别的。**

**在使用动态内存分配时，务必确保在不再需要时释放内存，以避免内存泄露。在这个实验中，通过delete[] arr;正确地释放了内存。**

**实验四：1. 问题定义**

**输入：一个整数数组，表示木棍的长度。**

**输出：一个包含三根木棍长度的数组，这三根木棍能构成一个三角形，并且它们的周长是所有可能组合中最长的。如果不存在这样的组合，则输出一个表示未找到的组合。**

**2. 解决方案**

**函数canFormTriangle：这个函数接受三个整数作为参数，分别代表三根木棍的长度。它使用三角形不等式定理来判断这三根木棍是否能构成一个三角形。三角形不等式定理指出，对于任意三角形，任意两边之和大于第三边。**

**函数findLongestTriangle：这个函数接受一个整数数组作为参数，表示木棍的长度。它使用三重循环来遍历所有可能的木棍组合，对于每种组合，它调用canFormTriangle函数来检查是否能构成一个三角形。如果能构成三角形，它还会计算这个三角形的周长，并更新最长周长和对应的木棍组合。**

**3. 试验过程**

**初始化：在main函数中，首先定义了一个包含木棍长度的数组sticks。这个数组可以根据需要进行修改。**

**调用函数：然后，调用findLongestTriangle函数，传入sticks数组作为参数。这个函数会返回能构成三角形的最长周长的木棍组合。**

**输出结果：最后，根据findLongestTriangle函数的返回值，输出能构成三角形的最长周长的木棍组合及其周长。如果没有找到能构成三角形的木棍组合，则输出相应的提示信息。**

**4. 复杂度分析**

**时间复杂度：由于findLongestTriangle函数使用了三重循环来遍历所有可能的木棍组合，因此其时间复杂度为O(n^3)，其中n是木棍的数量。对于较大的n，这可能会导致性能问题。**

**空间复杂度：除了存储输入数组和结果数组外，该函数没有使用额外的空间（不考虑递归栈或系统栈空间）。因此，其空间复杂度为O(1)（不考虑结果数组的情况下），或者O(k)，其中k是结果数组的大小（即3，因为总是返回三根木棍的组合）。**

**5. 改进建议**

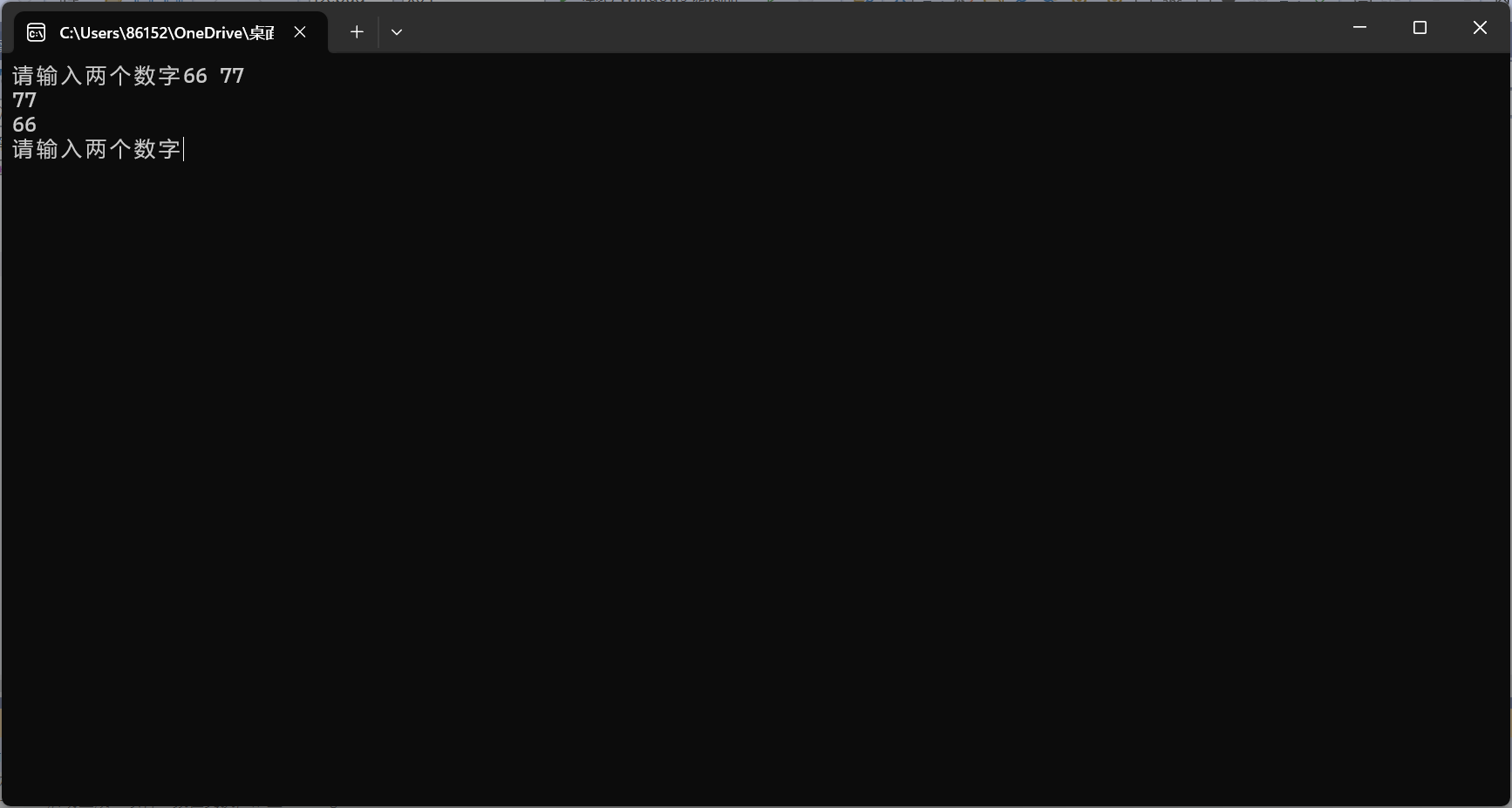
**排序：在调用findLongestTriangle函数之前，可以先对sticks数组进行排序。这样，在遍历木棍组合时，可以更早地排除那些不可能构成三角形的组合，从而在一定程度上减少不必要的计算。然而，由于排序的时间复杂度为O(n log n)，并且排序后仍然需要O(n^3)的时间来遍历所有组合，因此这种改进在渐进时间复杂度上并没有显著的优势。但在实践中，它可能会减少一些不必要的比较操作。**

**双指针：在排序后，可以使用双指针技术来优化遍历过程。具体来说，对于每个固定的木棍a（从最短到最长遍历），可以使用两个指针b和c来分别指向剩余木棍中的最短和最长木棍。然后，根据三角形不等式定理来调整指针的位置，从而避免不必要的比较。这种方法可以将时间复杂度降低到O(n^2 log n)（排序的时间加上双指针遍历的时间）。然而，由于排序的存在，这种方法仍然不是最优的。**

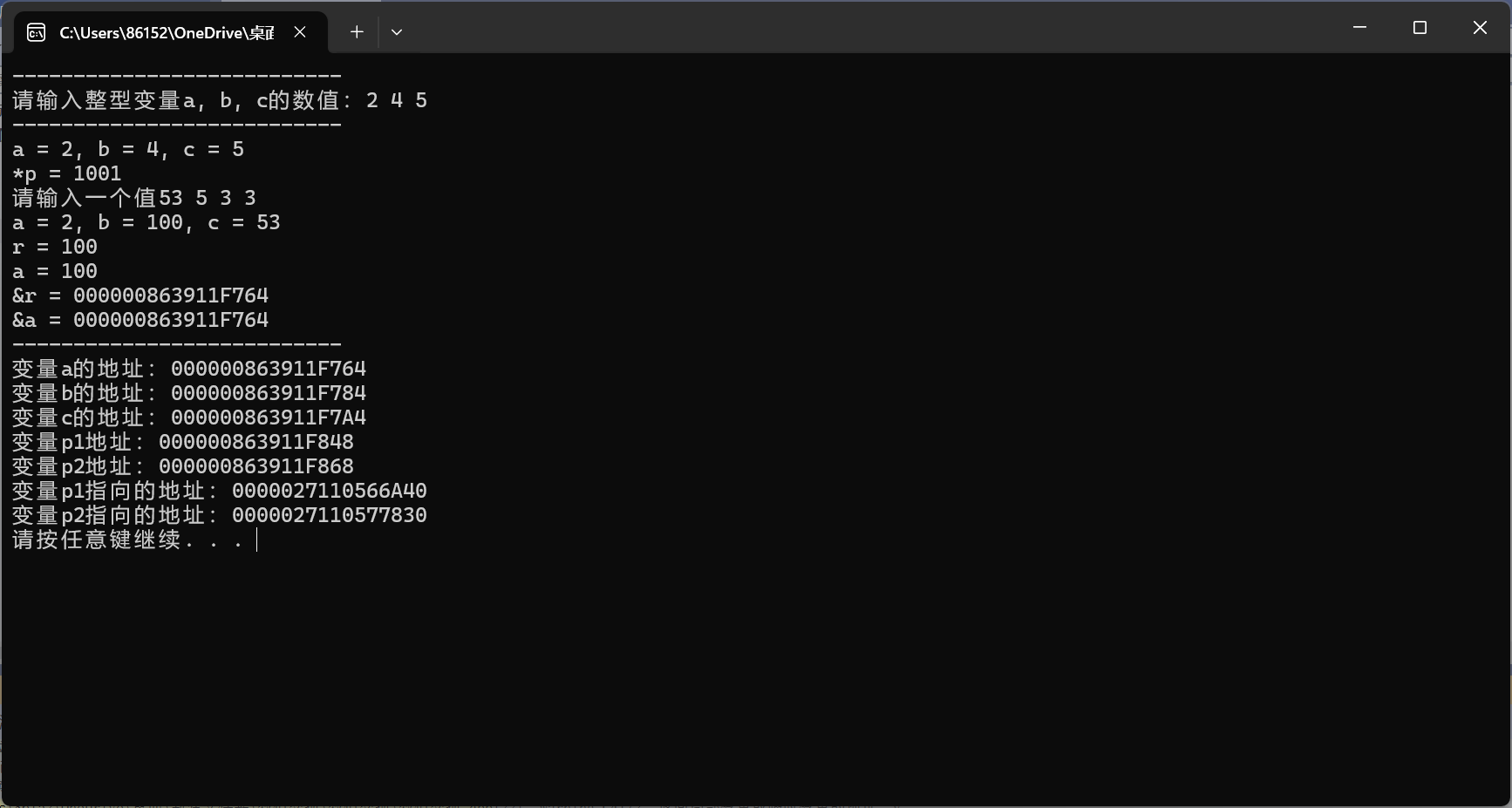
**最优算法：要找到最优的算法，可以考虑使用动态规划或其他高级算法来减少时间复杂度。然而，这个问题并没有一个简单的动态规划解决方案，因为它涉及到三个变量的组合和条件判断。因此，可能需要更复杂的算法或启发式方法来找到近似最优解或最优解。在实际应用中，如果n的值不是很大（例如n <= 1000），则三重循环的方法可能是可行的。如果n的值非常大，则可能需要考虑使用近似算法或启发式方法来找到解。**

**四、实验结果**

实验一



实验二



实验三



实验四



1. **实验总结**

**理解指针和引用的区别与联系**

**掌握C++指针和引用的基本方法**

**理解引用作为形式参数和返回类型的用法**

**掌握C++动态内存申请和释放的基本用法**

**熟练运用VISUAL STUDIO进行算法代码的调试**

* **附录：实验源代码（基于Highlight软件粘贴带有行号的源码）**
* **实验一**

01 #include <iostream>

02 #include <cstdlib>

03 #include <windows.h>

04 using namespace std;

05

06 **int**& function(**int** v, **int**\* p, **int**& r)

07 {

08 v = (\*p)++;

09

10 \*p = 100;

11

12 p = new **int**;

13

14 \*p = 1000;

15

16 r = 2 + (\*p)++;

17

18 cout << "\*p = " << \*p << endl;

19 cout << "������һ��ֵ";

20 cin >> r;

21

22 return v;

23 }

24

25

26 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ������ \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

27

28 **int** main()

29 {

30

31 cout << "---------------------------" << endl;

32

33 //1. ��ֵ����ַ�ʹ����÷���

34 **int** a, b, c;

35

36 cout << "���������ͱ���a, b, c����ֵ��";

37

38 cin >> a >> b >> c;

39

40 cout << "---------------------------" << endl;

41 cout << "a = " << a << ", b = " << b << ", c = " << c << endl;

42 function(a, &b, c); /\*function(a, b, c);�ڶ����β�Ӧ��Ϊָ������\*/

43 cout << "a = " << a << ", b = " << b << ", c = " << c << endl;

44

45

46 //2. ���ù��ɷ���

47 **int**& r = a;

48 r = a + b + c;

49 a = r + a + b + c;

50

51 r = b;

52

53 cout << "r = " << r << endl;

54 cout << "a = " << a << endl;

55

56 cout << "&r = " << &r << endl;

57 cout << "&a = " << &a << endl;

58

59

60 cout << "---------------------------" << endl;

61

62 //3. �ֲ������洢����

63 **int** i = 10, j = 20, k = 30;

64

65 cout << "����a�ĵ�ַ��" << &a << endl;

66 cout << "����b�ĵ�ַ��" << &b << endl;

67 cout << "����c�ĵ�ַ��" << &c << endl;

68

69

70 //4. ��̬�ڴ���������

71 **int**\* p1 = new **int**{ 10 };

72 **int**\* p2 = new **int**[10] {10, 20, 30};

73

74 cout << "����p1��ַ��" << &p1 << endl;/\*ָ������p�ĵ�ַӦ����& p\*/

75 cout << "����p2��ַ��" << &p2 << endl;

76

77 cout << "����p1ָ���ĵ�ַ��" << p1 << endl;/\*pָ��ָ���ĵ�ַӦ�þ���p��������������ͷ��ַ\*/

78 cout << "����p2ָ���ĵ�ַ��" << p2 << endl;

79

80 /\*cout << "����p1��ַ��" << p1 << endl;

81 cout << "����p2��ַ��" << p2 << endl;

82

83 cout << "����p1ָ���ĵ�ַ��" << &p1 << endl;

84 cout << "����p2ָ���ĵ�ַ��" << &p2 << endl;\*/

85

86

87 system("pause");

88 }

89

* **实验二**

01 #include<iostream>

02 using namespace std;

03 **void** swap(**int**& a, **int**& b)

04 {

05 **int** t = a;

06 a = b;

07 b = t;

08 }

09 **void** swap(**int**\* a, **int**\* b)

10 {

11 **int** t = \*a;

12 \*a = \*b;

13 \*b = t;

14 }

15 **int** main()

16 {

17 **int** a, b;

18 cout << "��������������";

19 cin >> a >> b;

20 swap(a, b);

21 cout << a << "\n" << b << "\n";

22 **int** x, y;

23 cout << "��������������";

24 cin >> x >> y;

25 swap(&x, &y);

26 cout << x << "\n" << y;

27

28 }

* **实验三**

01 #include <iostream>

02 #include <cstdlib>

03 #include <ctime>

04 #include <vector>

05 #include <algorithm>

06

07 **int**\* generateRandomArray(**int** size, **int** minVal, **int** maxVal);

08 **int** findMaxSubsequenceSum(**int**\* arr, **int** size);

09

10 **int** main()

11 {

12 std::srand(std::time(0));

13 **int** size = 10;

14 **int** minVal = -1000;

15 **int** maxVal = 1000;

16 **int**\* arr = generateRandomArray(size, minVal, maxVal);

17 std::cout << "���ɵ�����: ";

18 for (**int** i = 0; i < size; ++i)

19 {

20 std::cout << arr[i] << " ";

21 }

22 std::cout << std::endl;

23 **int** maxSum = findMaxSubsequenceSum(arr, size);

24 std::cout << "���������кͣ� " << maxSum << std::endl;

25 delete[] arr;

26

27 return 0;

28 }

29 **int**\* generateRandomArray(**int** size, **int** minVal, **int** maxVal)

30 {

31 **int**\* arr = new **int**[size];

32 for (**int** i = 0; i < size; ++i)

33 {

34 arr[i] = minVal + std::rand() % (maxVal - minVal + 1);

35 }

36 return arr;

37 }

38

39 **int** findMaxSubsequenceSum(**int**\* arr, **int** size)

40 {

41 **int** max\_current = arr[0];

42 **int** max\_global = arr[0];

43

44 for (**int** i = 1; i < size; i++)

45 {

46 max\_current = std::max(arr[i], max\_current + arr[i]);

47 if (max\_current > max\_global)

48 {

49 max\_global = max\_current;

50 }

51 }

52

53 return max\_global;

54 }

* **实验四**

01 #include <iostream>

02 #include <vector>

03 #include <algorithm>

04

05 // ��������������ľ���ܷ񹹳�һ��������

06 **bool** canFormTriangle(**int** a, **int** b, **int** c)

07 {

08 return (a + b > c) && (a + c > b) && (b + c > a);

09 }

10

11 // �ҳ��ܹ��������ε���ܳ�������ľ��

12 std::vector<**int**> findLongestTriangle(**const** std::vector<**int**>& sticks)

13 {

14 **int** n = sticks.size();

15 std::vector<**int**> longestTriangle;

16 **int** maxLength = 0; // ������ε��ܳ�

17

18 // ����ѭ���������п��ܵ�ľ������

19 for (**int** i = 0; i < n - 2; ++i)

20 {

21 for (**int** j = i + 1; j < n - 1; ++j)

22 {

23 for (**int** k = j + 1; k < n; ++k)

24 {

25 **int** a = sticks[i];

26 **int** b = sticks[j];

27 **int** c = sticks[k];

28

29 if (canFormTriangle(a, b, c))

30 {

31 **int** currentLength = a + b + c;

32 if (currentLength > maxLength)

33 {

34 maxLength = currentLength;

35 longestTriangle = { a, b, c };

36 }

37 }

38 }

39 }

40 }

41

42 return longestTriangle;

43 }

44

45 **int** main()

46 {

47 // ʾ��ľ���������飨���Ը�����Ҫ�޸ģ�

48 std::vector<**int**> sticks = { 3,3,32,45,44,4,4,42};

49

50 // �����ܹ��������ε���ܳ�������ľ��

51 std::vector<**int**> result = findLongestTriangle(sticks);

52

53 // ��������

54 if (!result.empty())

55 {

56 std::cout << "�ܹ��������ε���ܳ���ľ��������: ";

57 for (**int** length : result)

58 {

59 std::cout << length << " ";

60 }

61 std::cout << "���ܳ�Ϊ: " << (result[0] + result[1] + result[2]) << std::endl;

62 }

63 else

64 {

65 std::cout << "û���ҵ��ܹ��������ε�ľ�����ϡ�" << std::endl;

66 }

67

68 return 0;

69 }