**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по практической работе №3**

**по дисциплине «Программирование»**

Тема: Указатели.

|  |  |  |
| --- | --- | --- |
| Студент гр. 2372 |  | Братко В.В. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2022

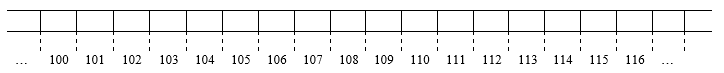
**Цель работы.**

Изучение указателей, работа с указателями.

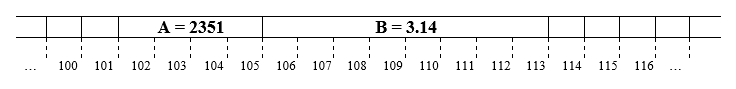
**Основные теоретические положения.**

Указатели и ссылки являются одними из самых важных и достаточно сложных для понимания и использования средств языка программирования. Они ориентированы на прямую работу с памятью компьютера. С помощью этих средств реализуется работа с динамической памятью и динамическими объектами, возвращение из функций измененных данных и многое другое. К использованию указателей и ссылок мы будем неоднократно возвращаться в последующих разделах.

Все данные (переменные, константы и др.) хранятся в памяти. Память представляет собой непрерывную последовательность ячеек (байтов), каждая из которых имеет свой номер – адрес:



При определении, например, некоторой переменной, она располагается в памяти по определенному адресу и занимает столько ячеек, сколько требует тип этой переменной. Пусть, например, имеется переменные **int A = 2351** и **double B = 3.1** и пусть они располагаются в памяти так:



Говорят, что переменная **А** располагается по адресу 101 и занимает 4 байта, а переменная **B** имеет адрес 105 и занимает 8 байт памяти.

**Указатели** – это тоже обычные переменные, но они **служат для хранения адресов памяти**.

Указатели определяются в программе следующим образом:

**<тип данных> \*<имя переменной>**

 Здесь <**тип данных**> определяет так называемый **базовый тип указателя**.

**<Имя переменной>**является идентификатором переменной-указателя.

Признаком того, что это переменная указатель, является символ \*, располагающийся между базовым типом указателя и именем переменной-указателя.

**Постановка задачи.**

1. Используя арифметику указателей, заполняет квадратичную целочисленную матрицу порядка N (6,8,10) случайными числами от 1 до  N\*N согласно схемам, приведенным на рисунках. Пользователь должен видеть процесс заполнения квадратичной матрицы.
2. Получает новую матрицу, из матрицы п. 1, переставляя ее блоки в соответствии со схемами:



1. Используя арифметику указателей, сортирует элементы любой сортировкой.
2. Уменьшает, увеличивает, умножает или делит все элементы матрицы на введенное пользователем число.

**Выполнение работы.**

|  |  |
| --- | --- |
| Ввод пользователем и обработка данных | Работа алгоритма и вывод на экран |
| 1.Заполнение матрицы | |
| Программа создает, создает матрицу N\*N |  |

Продолжение Таблицы

|  |  |
| --- | --- |
| 2. Получение новой матрицы | |
| Получает новую матрицу путем перестановки ее блоков |  |

|  |  |
| --- | --- |
| 4 Сортировка | |
| Сортируется матрица с помощью указателей |  |

|  |  |
| --- | --- |
| 5 Изменение значений матрицы | |
| Уменьшает, увеличивает, умножает или делит все элементы матрицы на введенное пользователем число. |  |

|  |  |
| --- | --- |
| Цикличность программы | |
| Eсли пользователь желает, то он может запустить выполнение программы заново, нажав цифру 1 |  |

Продолжение Таблицы

|  |  |
| --- | --- |
| Проверка на ошибки | |
| Eсли пользователь вводит неправильный тип данных, то программа на дает ему это сделать |  |

|  |  |
| --- | --- |
| Полный вывод программы | |
| Здесь представлен весь вывод консоли. | Вывод в консоли: |

Продолжение Таблицы

**Полный код программы:**

#include <iostream>  
#include <windows.h>  
#include <chrono>  
#include <thread>  
  
using namespace std;  
  
void gotoXY(int x, int y)  
{  
 COORD coord;  
 coord.X = x;  
 coord.Y = y + 1;  
 SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), coord);  
 this\_thread::sleep\_for(chrono::milliseconds(500));  
}  
  
  
int checkInput(){  
 int input;  
 try {  
 cin >> input;  
 if (cin.fail()) {  
 throw 1;  
 }  
 } catch (int exeption) {  
 cout << "ERROR!!!";  
 exit(0);  
 }  
 cin.sync();  
 cout << "\n";  
 return input;  
}  
  
void clear() {  
 COORD topLeft = { 0, 0 };  
 HANDLE console = GetStdHandle(STD\_OUTPUT\_HANDLE);  
 CONSOLE\_SCREEN\_BUFFER\_INFO screen;  
 DWORD written;  
  
 GetConsoleScreenBufferInfo(console, &screen);  
 FillConsoleOutputCharacterA(  
 console, ' ', screen.dwSize.X \* screen.dwSize.Y, topLeft, &written  
 );  
 FillConsoleOutputAttribute(  
 console, FOREGROUND\_GREEN | FOREGROUND\_RED | FOREGROUND\_BLUE,  
 screen.dwSize.X \* screen.dwSize.Y, topLeft, &written  
 );  
 SetConsoleCursorPosition(console, topLeft);  
}  
  
//выводит матрицу на экран  
void printMatrix(int \*pointer,int N ,int \*end){  
 for (int \*i = pointer, count = 1; i <= end; count++, i++){  
 printf("%2d ", \*i);  
 if (count % N == 0){ cout << "\n" ;}  
 }  
 cout << "\n";  
}  
  
void quickSort(int \*pointer, int N, int \*end){  
 int \*start = pointer, \*finish = end;  
 int \*pivot = pointer;  
 while (start <= finish){  
 while (\*start < \*pivot)  
 start++;  
 while (\*finish > \*pivot)  
 finish--;  
 if (start <= finish)  
 {  
 swap(\*start, \*finish);  
 start++;  
 finish--;  
 }  
 }  
 if (finish > pointer)  
 quickSort(pointer, N, finish);  
 if (start < end)  
 quickSort(start, N, end);  
}  
  
//Используя арифметику указателей, заполняет квадратичную целочисленную матрицу порядка N (6,8,10) случайными числами от 1 до N\*N согласно схемам, приведенным на рисунках.  
// Пользователь должен видеть процесс заполнения квадратичной матрицы.  
void function1(int \*pointer,int N, int \*end){  
 char exercise;  
 cout << "Choose the type of matrix (a or b):";  
 cin >> exercise;  
 cin.sync();  
 if (exercise == 'a'){  
 int value = 1;  
 for (int i = 0; i <= N / 2 - 1; i++){  
 bool flag = false;  
 for (int \*x = pointer + i \* N + i, j = 0; x <= pointer + (i + 1) \* N - i - 1; x++, j+=4){  
 \*x = value++;  
// gotoXY(j + i\*4 , i);  
// cout << \*x;  
 flag = true;  
 }  
 if(!flag){ break; }  
 flag = false;  
 for (int \*y = pointer + (i + 2) \* N - i - 1, j = 1; y <= end - i \* N - i; y += N, j++){  
 \*y = value++;  
// gotoXY( (N - i - 1)\*4, j);  
// cout << \*y;  
 flag = true;  
 }  
 if(!flag){ break; }  
  
 flag = false;  
 for (int \*z = end - i \* N - i - 1; z >= end - (i + 1) \* N + i + 1; z--){  
 \*z = value++;  
 flag = true;  
 }  
 if(!flag){ break; }  
  
 flag = false;  
 for (int \*w = end - (i + 2) \* N + i + 1; w >= pointer + (i + 1) \* N + i; w -= N){  
 \*w = value++;  
 flag = true;  
 }  
 if(!flag){ break; }  
  
 }  
 } else if(exercise == 'b'){  
 int value = 1;  
 for (int i = 0; i <= N - 1; i++){  
 bool flag = false;  
 for (int \*j = pointer + i; j <= end - N + i + 1; j += N){  
 \*j = value++;  
 flag = true;  
 }  
 i++;  
 if(!flag){ break; }  
 flag = false;  
 for (int \*z = end - N + i + 1; z >= pointer + i; z -= N){  
 \*z = value++;  
 flag = true;  
 }  
 if(!flag){ break; }  
  
 }  
 }else {  
 function1(pointer, N, end);  
 }  
 printMatrix(pointer, N ,end);  
}  
//2.Получает новую матрицу, из матрицы п. 1, переставляя ее блоки в соответствии со схемами:  
void function2(int \*pointer,int N, int \*end){  
 int arr1[N/2][N/2], arr2[N/2][N/2], arr3[N/2][N/2], arr4[N/2][N/2];  
 int \*pointer1 = (int \*) arr1, \*pointer2 = (int \*) arr2, \*pointer3 = (int \*) arr3, \*pointer4 = (int \*) arr4;  
 int \*end1 = arr1[0] + N \* N / 4 - 1, \*end2 = arr2[0] + N \* N / 4 - 1, \*end3 = arr3[0] + N \* N / 4 - 1, \*end4= arr4[0] + N \* N / 4 - 1;  
 for (int \*x = pointer1, i = 0; x <= end1; x++){  
 \*x = \*(pointer + i);  
 i++;  
 if (i % (N/2) == 0){  
 i+=N/2;  
 }  
 }  
 for (int \*y = pointer2, i = N / 2; y <=end2 ; y++){  
 \*y = \*(pointer + i);  
 i++;  
 if (i % (N/2) == 0){  
 i+=N/2;  
 }  
 }  
 for (int \*z = pointer3, i = N \* N / 2; z <=end3 ; z++){  
 \*z = \*(pointer + i);  
 i++;  
 if (i % (N/2) == 0){  
 i+=N/2;  
 }  
 }  
 for (int \*w = pointer4, i = N \* N / 2 + N / 2; w <=end4 ; w++){  
 \*w = \*(pointer + i);  
 i++;  
 if (i % (N/2) == 0){  
 i+=N/2;  
 }  
 }  
 char exercise;  
 cout << "Choose the type of matrix (a or b or c or d):";  
 cin >> exercise;  
 cin.sync();  
 if (exercise == 'a'){  
 for (int \*x = pointer1, \*y = pointer2; x <= end1; x++, y++){  
 swap(\*x, \*y);  
 }  
  
 for (int \*z = pointer1, \*w = pointer4; z <= end1; z++, w++){  
 swap(\*z, \*w);  
 }  
 for (int \*q = pointer1, \*r = pointer3; q <= end1; q++, r++){  
 swap(\*q, \*r);  
 }  
 } else if(exercise == 'b'){  
 for (int \*x = pointer1, \*y = pointer4; x <= end1; x++, y++){  
 swap(\*x, \*y);  
 }  
 for (int \*z = pointer2, \*w = pointer3; z <= end2; z++, w++){  
 swap(\*z, \*w);  
 }  
 }else if(exercise == 'c'){  
 for (int \*x = pointer1, \*y = pointer3; x <= end1; x++, y++){  
 swap(\*x, \*y);  
 }  
 for (int \*z = pointer2, \*w = pointer4; z <= end2; z++, w++){  
 swap(\*z, \*w);  
 }  
 }else if(exercise == 'd'){  
 for (int \*x = pointer1, \*y = pointer2; x <= end1; x++, y++){  
 swap(\*x, \*y);  
 }  
 for (int \*z = pointer3, \*w = pointer4; z <= end3; z++, w++){  
 swap(\*z, \*w);  
 }  
 }else{  
 function2(pointer, N, end);  
 }  
 for (int \*x = pointer1, i = 0; x <= end1; x++){  
 \*(pointer + i) = \*x;  
 i++;  
 if (i % (N/2) == 0){  
 i+=N/2;  
 }  
 }  
 for (int \*y = pointer2, i = N / 2; y <=end2 ; y++){  
 \*(pointer + i) = \*y;  
 i++;  
 if (i % (N/2) == 0){  
 i+=N/2;  
 }  
 }  
 for (int \*z = pointer3, i = N \* N / 2; z <=end3 ; z++){  
 \*(pointer + i) = \*z;  
 i++;  
 if (i % (N/2) == 0){  
 i+=N/2;  
 }  
 }  
 for (int \*w = pointer4, i = N \* N / 2 + N / 2; w <=end4 ; w++){  
 \*(pointer + i) = \*w;  
 i++;  
 if (i % (N/2) == 0){  
 i+=N/2;  
 }  
 }  
 printMatrix(pointer, N, end);  
}  
//3.Используя арифметику указателей, сортирует элементы сортировкой quick sort  
void function3(int \*pointer, int N, int \*end){  
 quickSort(pointer, N, end);  
 cout << "Sorted array:\n";  
 printMatrix(pointer, N, end);  
}  
//4. Уменьшает, увеличивает, умножает или делит все элементы матрицы на введенное пользователем число.  
void function4(int \*pointer, int N, int \*end){  
 cout << R"(Which operation do you want to do(1 : "-", 2 : "+", 3 : "\*", 4 : "/",):)";  
 int operand = checkInput();  
 cout << "Input the value you want to change to the matrix by:";  
 int value = checkInput();  
 switch (operand) {  
 case 1:  
 for(int \*i = pointer; i <= end; i++){  
 \*i -= value;  
 }  
 break;  
 case 2:  
 for(int \*i = pointer; i <= end; i++){  
 \*i += value;  
 }  
 break;  
 case 3:  
 for(int \*i = pointer; i <= end; i++){  
 \*i \*= value;  
 }  
 break;  
 case 4:  
 for(int \*i = pointer; i <= end; i++){  
 \*i /= value;  
 }  
 break;  
 default:  
 function4(pointer, N, end);  
 }  
 cout << "Result:\n";  
 printMatrix(pointer, N, end);  
}  
  
int main() {  
 int circle = 1;  
 while (circle == 1){  
 cout << "Set the matrix size(6,8,10):";  
 int N = checkInput();  
 int arr[N][N];  
 int \*pointer = (int \*) arr;  
 int \*end = arr[0] + N \* N - 1;  
 //заполняем матрицу нулевыми значениями  
 for (int \*i = arr[0]; i <= end; i++){  
 \*i = 0;  
 }  
 clear();  
 //1.Используя арифметику указателей, заполняет квадратичную целочисленную матрицу порядка N (6,8,10) случайными числами от 1 до N\*N согласно схемам, приведенным на рисунках.  
 // Пользователь должен видеть процесс заполнения квадратичной матрицы.  
 function1(pointer, N, end);  
 cout << "\n";  
 //2.Получает новую матрицу, из матрицы п. 1, переставляя ее блоки в соответствии со схемами:  
 function2(pointer, N, end);  
 //3.Используя арифметику указателей, сортирует элементы сортировкой quick sort  
 function3(pointer, N, end);  
 //4.Уменьшает, увеличивает, умножает или делит все элементы матрицы на введенное пользователем число.  
 function4(pointer, N, end);  
 cout << "If you want to repeat then input 1: ";  
 circle = checkInput();  
 }  
  
 return 0;  
}

**Вывод:**

В процессе написания программы для практической работы было изучено:

1. Что такое указатель
2. Как работает с указателями
3. Зачем нужны указатели