**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: **ДВУХМЕРНЫЕ СТАТИЧЕСКИЕ МАССИВЫ И УКАЗАТЕЛИ**

.

|  |  |  |
| --- | --- | --- |
| Студент гр. 3372 |  | Поконечный А.О |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2023

**Цель работы.**

Изучение и организация структур; получение практических навыков работы со структурами; определение преимуществ и недостатков использования структур.

**Основные теоретические положения.**

Структуры представляют собой группы связанных между собой, как правило, разнотипных переменных, объединенных в единый объект, в отличие от массива, все элементы которого однотипны. В языке C++ структура является видом класса и обладает всеми его свойствами. Чаще всего ограничиваются тем, как структуры представлены в языке С:

struct [имя\_типа] {

тип\_1 элемент\_1;

тип \_2 элемент\_2;

…

тип\_k элемент\_k;

} [ список\_описателей ];

Описание структуры начинается ключевым словом struct. Каждая входящая в структуру переменная называется членом (полем, элементом) структуры и описывается типом данных и именем. Поля структуры могут быть любого типа данных. Их количество не лимитировано.

Вся эта конструкция является инструкцией языка программирования, поэтому после нее всегда должен ставиться символ ‘;’.

При описании структуры память для размещения данных не выделяется. Работать с описанной структурой можно только после того, как будет определена переменная (переменные) этого типа данных, только при этом компилятор выделит необходимую память.

Для инициализации структуры значения ее элементов перечисляют в фигурных скобках в порядке их описания:

struct complex{

float real, im;

} data [2][2] = {

{{1,1}, {2,2}},

{{3,3}, {4,4}}

};

Все поля структурных переменных располагаются в непрерывной области памяти одно за другим. Общий объем памяти, занимаемый структурой, равен сумме размеров всех полей структуры. Для определения размера структуры следует использовать инструкцию sizeof().

Для того чтобы записать данные в структурную переменную, необходимо каждому полю структуры присвоить определенное значение. Для этого необходимо использовать оператор ‘’ («точка»):

struct Stack { // Cтек

float arr[100];

short topIndex;

};

…

Stack stack; // Объявляем переменную типа Stack

Stack.arr[0] = 1;

…

При доступе к определенному полю его следует рассматривать как обычную переменную, тип данных которой соответствует типу этого поля. Поля структур могут участвовать в качестве операндов любых выражений, допускающих использование операндов соответствующего типа данных.

Копирование данных из одной структурной переменной в другую осуществляется простой операцией присваивания, независимо от количества полей и размера структуры (это можно делать только в том случае, когда обе переменные одного и того же типа).

В программировании очень часто используются такие конструкции, как массивы структур. Например, сведения о студентах некоторой учебной группы можно хранить в массиве студентов:

t\_Student Gruppa\_N [30];

Был определен 30-элементный массив, каждый элемент которого предназначен для хранения данных одного студента. Получение доступа к данным некоторого студента из группы *N* осуществляется обычной индексацией переменной массива. Поскольку поля структуры могут быть любого типа данных, то они в свою очередь могут быть другой структурой или массивом других структур:

struct Stud

{

char FN[100];

short listNumber;

};

struct Group

{

int groupNumber;

short students;

Stud stud[30];

};

Но в структуре поля нельзя использовать элемент, тип которого совпадает с типом самой структуры, так как рекурсивное использование структур запрещено.

Любая структурная переменная занимает в памяти определенное положение, характеризующееся конкретным адресом. Для работы с адресами структурных переменных (как и для простых переменных) можно использовать указатели. Указатели на структурные переменные определяются точно так же, как и для обычных переменных. Разыменование указателя (обращение к данным по адресу, хранящемуся в указателе) осуществляется также обычным образом.

Через указатели можно работать с отдельными полями структур. Для доступа к полю структуры через указатель используется оператор ‘’ («стрелка»), а не «точка».

Структуры можно использовать в качестве параметров функций, как и обычные переменные. Для структур поддерживаются все три механизма передачи данных: по значению, через указатели и по ссылке.

Передачу структур в функции по значению необходимо использовать аккуратно:

void WriteStudent ( t\_Student S )

{

cout << "Фамилия: " << S.Fam << endl;

cout << "Имя: " << S.Name << endl;

cout << "Год рождения: " << S.Year << endl;

if ( S.Sex )

cout << "Пол: " << "М\n";

else

cout << "Пол: " << "Ж\n";

cout << "Средний балл: " << S.Grade << endl;

}

Вызов такой функции сопровождается дополнительным расходом памяти для создания локальной переменной *S*и дополнительными затратами времени на физическое копирование данных из аргумента в параметр *S*. Учитывая то, что объем структур может быть очень большим, эти дополнительные затраты вычислительных ресурсов могут быть чрезмерными.

Предпочтительно использование передачи структуры по указателю или ссылке:

void WriteStudent ( t\_Student \*S )

{

cout << "Фамилия: " << S -> Fam << endl;

cout << "Имя: " << S -> Name << endl;

cout << "Год рождения: " << S -> Year << endl;

if ( S -> Sex )

cout << "Пол: " << "М\n";

else

cout << "Пол: " << "Ж\n";

cout << "Средний балл: " << S -> Grade << endl;

}

Фактической передачи данных в функцию не осуществляется. Дополнительные затраты памяти для создания локальной переменной небольшие – это адрес памяти (4 байта, независимо от размера самой структуры). Вызов такой функции будет происходить быстрее, а расход памяти будет существенно меньше, чем при передаче данных по значению.

Передача по ссылке по эффективности эквивалентна передаче данных через указатель. Однако, поскольку при передаче данных по ссылке все адресные преобразования берет на себя компилятор, существенно упрощается программирование действий со структурами. При использовании ссылочных параметров структурных типов доступ к членам структуры осуществляется обычным способом – с помощью оператора «точка».

Недостатком этих способов является то, что случайные изменения значений полей структуры внутри функции отразятся на значении аргумента после окончания работы функции. Если необходимо предотвратить изменения переданных по адресу аргументов, можно при определении соответствующего параметра объявить его константой (использовать спецификатор const).

**Постановка задачи.**

* Необходимо написать программу, которая:
* 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 (\*Для манипуляции с элементами используйте только арифметику указателей):
* 
* 2)    Получает новую матрицу, из матрицы п. 1, переставляя ее блоки в соответствии со схемами (\*Для манипуляции с элементами используйте только арифметику указателей):
* 
* 3)    Используя арифметику указателей, сортирует элементы любой сортировкой из списка ниже (если во 2 ПР вы реализовывали одну из представленных сортировок, рекомендуется переиспользовать написанный код и модернизировать его для работы с указатями).   
  Варианты сортировок:
* Shaker sort;
* Comb sort;
* Insert sort;
* Quick sort;
* 4)    Уменьшает, увеличивает, умножает или делит все элементы матрицы на введенное пользователем число (\*Для манипуляции с элементами используйте только арифметику указателей).

**Выполнение работы.**

**Код программы представлен в приложении А**.

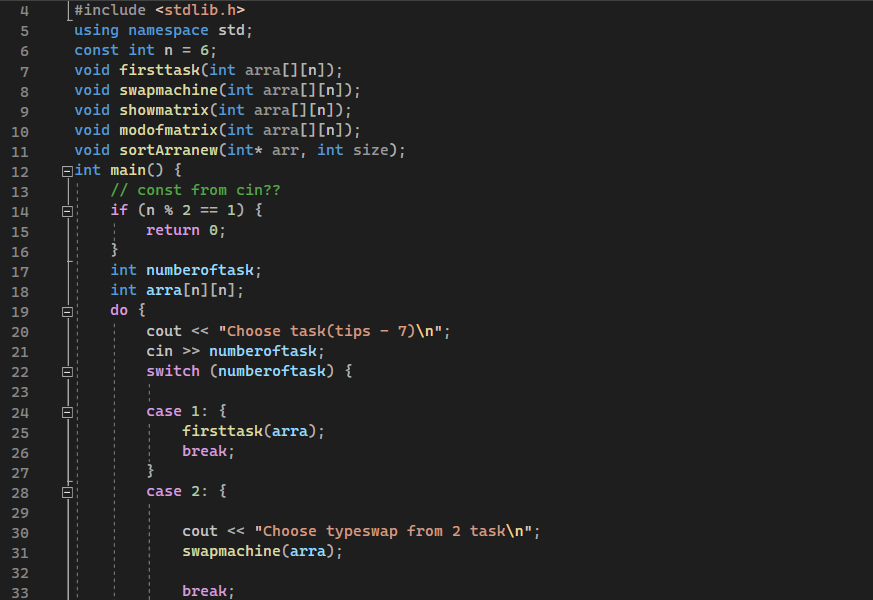
Часть кода

**Блок описания кода и использованных алгоритмов**

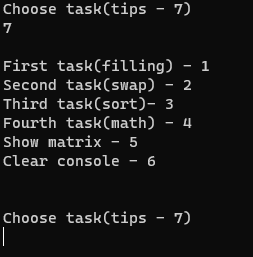
Использованы указатели,arra[][](ДВУХМЕРНЫЙ МАССИВ!!!), switch, do, bubble сортировка, cout, cin, if. Это очень нужные конструкции, без которых бы данный код не заработал!!!

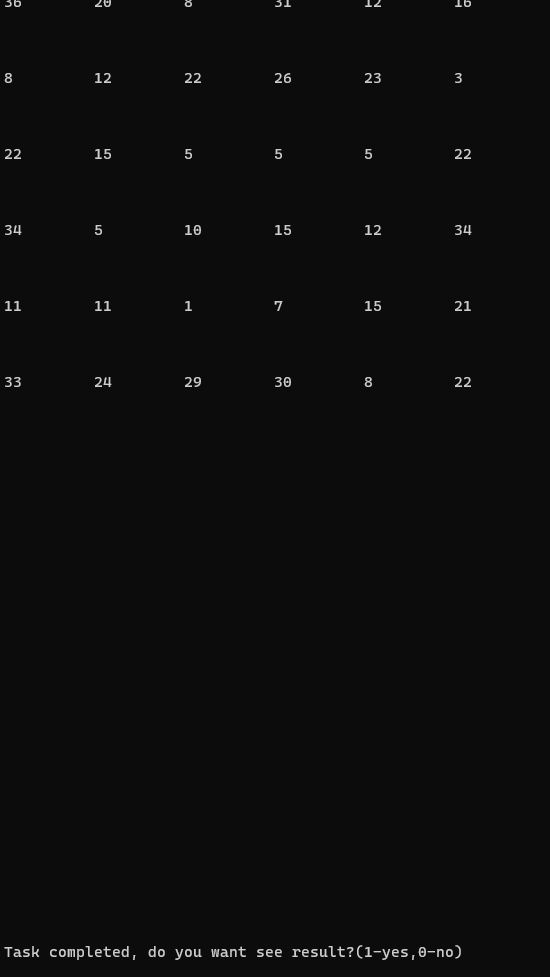
**Блок скриншотов работы программы**

Выбор задания



**Блок таблицы с тестовыми данными**





**Выводы.**

Выполнив данную работу мы поняли, как долго выполняет сортировку компьютер, что сортировка может ускорить или замедлить работу программы, как отсортировать массив. Так же мы усовершенствовали свои навыки использования конструкций if, else, работы с массивами

Приложение А

рабочий код

#include <iostream>

#include <Windows.h>

#include <algorithm>

#include <stdlib.h>

using namespace std;

const int n = 6;

void firsttask(int arra[][n]);

void swapmachine(int arra[][n]);

void showmatrix(int arra[][n]);

void modofmatrix(int arra[][n]);

void sortArranew(int\* arr, int size);

int main() {

// const from cin??

if (n % 2 == 1) {

return 0;

}

int numberoftask;

int arra[n][n];

do {

cout << "Choose task(tips - 7)\n";

cin >> numberoftask;

switch (numberoftask) {

case 1: {

firsttask(arra);

break;

}

case 2: {

cout << "Choose typeswap from 2 task\n";

swapmachine(arra);

break;

}

case 3: {

cout << "Sorting\n";

sortArranew(&arra[0][0], n);

cout << "Completed\n";

break;

}

case 4: {

modofmatrix(arra);

break;

}

case 5: {

showmatrix(arra);

break;

}

case 6: {

cout << "Clearing...";

Sleep (1500);

system("cls");

break;

}

case 7: {

cout << "\nFirst task(filling) - 1";

cout << "\nSecond task(swap) - 2";

cout << "\nThird task(sort)- 3";

cout << "\nFourth task(math) - 4";

cout << "\nShow matrix - 5";

cout << "\nClear console - 6\n\n\n";

break;

}

}

}while (numberoftask);

}

void firsttask(int arra[][n]) {

int k = 0;

HANDLE hStdout;

COORD destCoord;

hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);

int firsttask = 0;

cout << "Choose type of filling from first task\n";

cin >> firsttask;

switch (firsttask) {

case 1: {

system("cls");

do {

for (int x = k; x < n - k; x++) {

arra[k][x] = 1 + rand() % (n \* n);

destCoord.X = x \* 10;

destCoord.Y = k \* 4;

SetConsoleCursorPosition(hStdout, destCoord);

cout << arra[k][x] << '\r';

Sleep(100);

cout.flush();

}

for (int y = k + 1; y < n - k; y++) {

arra[y][n - 1 - k] = 1 + rand() % (n \* n);

destCoord.X = (n - 1 - k) \* 10;

destCoord.Y = y \* 4;

SetConsoleCursorPosition(hStdout, destCoord);

cout << arra[y][n - 1 - k] << '\r';

Sleep(100);

cout.flush();

}

for (int x = n - 2 - k; x >= k; x--) {

arra[n - 1 - k][x] = 1 + rand() % (n \* n);

destCoord.X = x \* 10;

destCoord.Y = (n - 1 - k) \* 4;

SetConsoleCursorPosition(hStdout, destCoord);

cout << arra[n - 1 - k][x] << '\r';

Sleep(100);

cout.flush();

}

for (int y = n - 2 - k; y > k; y--) {

arra[y][k] = 1 + rand() % (n \* n);

destCoord.X = k \* 10;

destCoord.Y = y \* 4;

SetConsoleCursorPosition(hStdout, destCoord);

cout << arra[y][k] << '\r';

Sleep(100);

cout.flush();

}

k++;

} while (!(k == n));

break;

}

case 2: {

system("cls");

k = 0;

do {

for (int i = 0; i < n; i++) {

if (k % 2 == 0) {

for (int y = n - 1; y > -1; y--) {

arra[y][i] = 1 + rand() % (n \* n);

destCoord.X = (i) \* 10;

destCoord.Y = y \* 4;

SetConsoleCursorPosition(hStdout, destCoord);

cout << arra[y][i] << '\r';

Sleep(100);

cout.flush();

}

}

else {

for (int y = 0; y < n; y++) {

arra[y][i] = 1 + rand() % (n \* n);

destCoord.X = (i) \* 10;

destCoord.Y = y \* 4;

SetConsoleCursorPosition(hStdout, destCoord);

cout << arra[y][i] << '\r';

Sleep(100);

cout.flush();

}

}

k++;

}

} while (!(k == n));

}

}

destCoord.X = 0;

destCoord.Y = 50;

SetConsoleCursorPosition(hStdout, destCoord);

cout << "Task completed, do you want see result?(1-yes,0-no)\n";

int seeresult = 0;

cin >> seeresult;

if (seeresult) {

showmatrix(arra);

}

}

void swapmachine(int arra[][n]) {

int typeswap;

cout << "Choose type of swap(1,2,3,4)\n";

cin >> typeswap;

switch (typeswap) {

//swap first type

case 1: {

for (int i = 0; i < (n / 2); i++) {

for (int g = 0; g < (n / 2); g++) {

swap(arra[i][g], arra[i][g + n / 2]);

swap(arra[i][g + n / 2], arra[i + n / 2][g + n / 2]);

swap(arra[i + n / 2][g + n / 2], arra[i][g]);

swap(arra[i][g], arra[n / 2 + i][g]);

}

}

break;

}

//swap second type

case 2: {

for (int i = 0; i < (n / 2); i++) {

for (int g = 0; g < (n / 2); g++) {

swap(arra[i][g], arra[i + n / 2][g + n / 2]);

swap(arra[i][g + n / 2], arra[i + n / 2][g]);

}

}

break;

}

case 3: {

for (int i = 0; i < (n / 2); i++) {

for (int g = 0; g < (n / 2); g++) {

swap(arra[i][g], arra[i + n / 2][g]);

swap(arra[i][g + n / 2], arra[i + n / 2][g + n / 2]);

}

}

break;

}

case 4: {

for (int i = 0; i < (n / 2); i++) {

for (int g = 0; g < (n / 2); g++) {

swap(arra[i][g], arra[i][g + n / 2]);

swap(arra[i + n / 2][g], arra[i + n / 2][g + n / 2]);

}

}

break;

}

}

cout << "Task completed, do you want see result?(1-yes,0-no)\n";

int seeresult = 0;

cin >> seeresult;

if (seeresult) {

showmatrix(arra);

}

}

void showmatrix(int arra[][n]) {

cout << "\n\n\n\n\n\n\n\n\n\n";

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

cout << arra[i][j] << " ";

}

cout << endl;

}

}

void modofmatrix(int arra[][n]) {

int num, type;

cout << "Choose type. \n 1 - minus\n2 - plus\n3 - \*\n4 - /";

cin >> type;

cout << "Enter Num";

cin >> num;

switch (type)

{

case 1: {

for (int i = 0; i < n; i++) {

for (int g = 0; g < n; g++) {

arra[i][g] = arra[i][g] - num;

}

}

break;

}

case 2: {

for (int i = 0; i < n; i++) {

for (int g = 0; g < n; g++) {

arra[i][g] = arra[i][g] + num;

}

}

break;

}

case 3: {

for (int i = 0; i < n; i++) {

for (int g = 0; g < n; g++) {

arra[i][g] = arra[i][g] \* num;

}

}

break;

}

case 4: {

for (int i = 0; i < n; i++) {

for (int g = 0; g < n; g++) {

arra[i][g] = arra[i][g] / num;

}

}

break;

}

}

cout << "Completed\n";

}

void sortArranew(int\* arr, int size) {

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

for (int j = i + 1; j < size \* size; j++) {

if (\*(arr + i) > \*(arr + j)) {

int temp = \*(arr + i);

\*(arr + i) = \*(arr + j);

\*(arr + j) = temp;

}

}

}

}