Министерство науки и высшего образования Российской Федерации

Пензенской государственный университет

Кафедра "Вычислительная техника"

**Отчёт**

по лабораторной работе №2

по курсу «Логика и Основы Алгоритмизации в Инженерных Задачах»

на тему «Простые структуры данных»

Выполнил:

Студент группы 23ВВ1

Юров Д.М.

Принял:

Митрохин М.А.

Юрова О.В.

Пенза 2024

**Цель работы:** изучить простые структуры данных в языке Си.

**Лабораторное задание:**

***Задание 1:*** написать программу, вычисляющую разницу между максимальным и минимальным элементами массива.

***Задание 2****:* написать программу, реализующую инициализацию массива случайными числами.

***Задание 3****:* написать программу, реализующую создание массива произвольного размера, вводимого с клавиатуры.

***Задание 4****:* написать программу, вычисляющую сумму значений в каждом столбце (или строке) двумерного массива.

***Задание 5:*** написать программу, осуществляющую поиск среди структур student структуру с заданными параметрами (фамилией, именем и т.д.).

**Теоретическая часть:**

К простым структурам данных языка Си относятся массивы, строки (массивы символов). ***Структуры*** (struct) – простейшая составная структура данных.

***Массив*** – это конечная совокупность данных одного типа.

Массивы могут состоять из целых чисел, чисел с плавающей запятой, символов и других типов данных. Существуют даже массивы массивов, – многомерные массивы.

***Динамический массив.*** Для использования функций динамического распределения памяти необходимо подключение библиотеки <malloc.h> или <stdlib.h> (в зависимости от компилятора и используемого стандарта языка).

*Функции динамического выделения памяти*

void\* malloc(*РазмерМассиваВБайтах*);  
 void\* calloc(*ЧислоЭлементов*, *РазмерЭлементаВБайтах*);

выделяют блок памяти, размером *РазмерМассиваВБайтах* или *ЧислоЭлементов*\**РазмерЭлементаВБайтах* байт, и возвращает указатель на начало блока.

Элементами массива могут быть как данные простых типов, так и составных типов (например, структуры), поэтому для точного определения размера элемента массива в общем случае используется функция

int sizeof(*ТипЭлемента*);  
Эта функция возвращает количество байт, занимаемое элементом указанного типа.

Память, динамически выделенная с использованием функций calloc(), malloc(), должна быть освобождена после окончания использования функцией

 free(*указатель*);

Для ***генерации случайных чисел*** в языке Си используется функция rand(), которая содержится в библиотеке <stdlib.h>.

При повторных запусках программы функция rand() будет возвращать одинаковые последовательности чисел. Для получения уникальных последовательностей необходимо инициализировать генератор случайных чисел различными значениями. Для этого используется функция srand(), аргументом которой является инициализирующее целое число.

***Структура*** - это совокупность переменных одного или нескольких типов, сгруппированных в один элемент.

**Практическая часть:**

**Листинг:**

**#define \_CRT\_SECURE\_NO\_WARNINGS**

**#include <stdlib.h>**

**#include <stdio.h>**

**#include <iostream>**

**#include <time.h>**

**#include <iomanip>**

**using namespace std;**

**int main()**

**{**

**setvbuf(stdin, NULL, \_IONBF, 0);**

**setvbuf(stdout, NULL, \_IONBF, 0);**

**int i = 0, j = 0, r, elem\_c;**

**int volume[7] = { 100, 200, 400, 1000, 2000, 4000, 10000 };**

**long int start\_time = 0, end\_time = 0, search\_time = 0;**

**for (int y = 0; y < 7; y++)**

**{**

**int\*\* a = new int\* [volume[y]];**

**int\*\* b = new int\* [volume[y]];**

**int\*\* c = new int\* [volume[y]];**

**for (int z = 0; z < volume[y]; z++)**

**{**

**a[z] = new int[volume[y]];**

**b[z] = new int[volume[y]];**

**c[z] = new int[volume[y]];**

**}**

**std::cout << "volume elements:" << volume[y] << endl;**

**start\_time = clock();**

**i = 0; j = 0;**

**srand(time(NULL));**

**while (i < volume[y])**

**{**

**while (j < volume[y])**

**{**

**a[i][j] = rand() % 100 + 1;**

**j++;**

**}**

**i++;**

**}**

**i = 0; j = 0;**

**srand(time(NULL));**

**while (i < volume[y])**

**{**

**while (j < volume[y])**

**{**

**b[i][j] = rand() % 100 + 1;**

**j++;**

**}**

**i++;**

**}**

**for (i = 0; i < volume[y]; i++)**

**{**

**for (j = 0; j < volume[y]; j++)**

**{**

**elem\_c = 0;**

**for (r = 0; r < volume[y]; r++)**

**{**

**elem\_c = elem\_c + a[i][r] \* b[r][j];**

**c[i][j] = elem\_c;**

**}**

**}**

**}**

**end\_time = clock();**

**search\_time = end\_time - start\_time;**

**delete[]a;**

**delete[]b;**

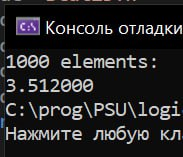
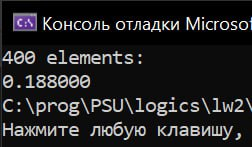
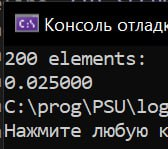
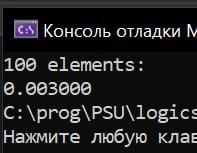
**delete[]c;**

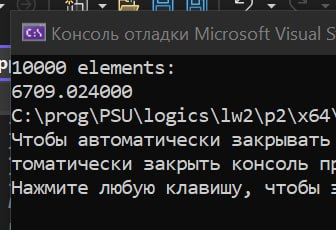
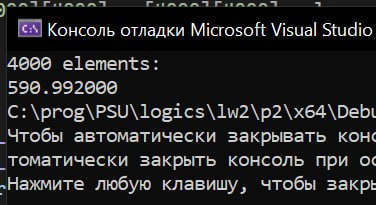
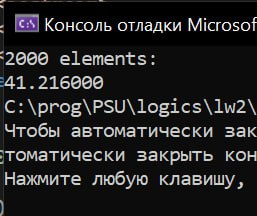
**cout << "time: " << search\_time / 1000.0 << endl << endl;**

**}**

**}**

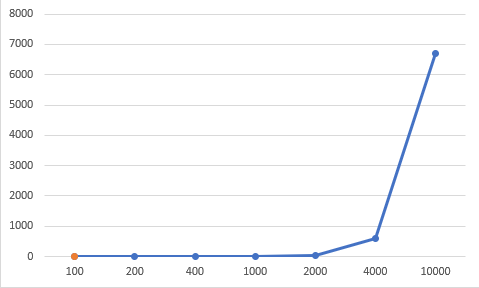
**Результат работы программы:**





**Сложность программы: О(n2).**

**График результатов программы:**

****

**Построив диаграмму зависимости времени выполнения программы от размера матриц, мы подтверждаем то, что сложность данной программы: О(n2).**

**Листинг 2:**

**#define \_CRT\_SECURE\_NO\_WARNINGS**

**#include <iostream>**

**#include <stdio.h>**

**#include <time.h>**

**#include <locale.h>**

**#include <conio.h>**

**#include <string.h>**

**#include <stdlib.h>**

**using namespace std;**

**void shell(int\* items, int count);**

**void qs(int\* items, int left, int right);**

**int cmp(const void\* a, const void\* b);**

**int main()**

**{**

**int z = 0, volume = 100000;**

**long int start\_time = 0, end\_time = 0, compare\_time = 0;**

**int\* a = new int [volume];**

**int\* b = new int [volume];**

**int\* c = new int [volume];**

**cout << "now sorting randomized arrays" << endl;**

**srand(time(NULL));**

**for (int i = 0; i < volume; i++)**

**{**

**a[i] = rand() % 100 + 1;**

**b[i] = rand() % 100 + 1;**

**c[i] = rand() % 100 + 1;**

**}**

**cout << "shell() sort" << endl;**

**start\_time = clock();**

**shell(a, volume);**

**end\_time = clock();**

**compare\_time = end\_time - start\_time;**

**cout << "sort time: " << compare\_time / 1000.0 << endl << endl;**

**cout << "qs() sort" << endl;**

**start\_time = clock();**

**qs(b, 0, volume - 1);**

**end\_time = clock();**

**compare\_time = end\_time - start\_time;**

**cout << "sort time: " << compare\_time / 1000.0 << endl << endl;**

**cout << "qsort() sort" << endl;**

**start\_time = clock();**

**qsort(c, volume, sizeof(int), cmp);**

**end\_time = clock();**

**compare\_time = end\_time - start\_time;**

**cout << "sort time: " << compare\_time / 1000.0 << endl;**

**z++;**

**cout << "----------------------" << endl << endl;**

**cout << "now sorting arrays cons. of increasing sequence" << endl;**

**for (int i = 0; i < volume; i++)**

**{**

**a[i] = i;**

**b[i] = i;**

**c[i] = i;**

**}**

**cout << "shell() sort" << endl;**

**start\_time = clock();**

**shell(a, volume);**

**end\_time = clock();**

**compare\_time = end\_time - start\_time;**

**cout << "sort time: " << compare\_time / 1000.0 << endl << endl;**

**cout << "qs() sort" << endl;**

**start\_time = clock();**

**qs(b, 0, volume - 1);**

**end\_time = clock();**

**compare\_time = end\_time - start\_time;**

**cout << "sort time: " << compare\_time / 1000.0 << endl << endl;**

**cout << "qsort() sort" << endl;**

**start\_time = clock();**

**qsort(c, volume, sizeof(int), cmp);**

**end\_time = clock();**

**compare\_time = end\_time - start\_time;**

**cout << "sort time: " << compare\_time / 1000.0 << endl;**

**z++;**

**cout << "----------------------" << endl << endl;**

**cout << "now sorting arrays cons. of decreasing sequence" << endl;**

**for (int i = 0, j = volume-1; i < volume; i++, j--)**

**{**

**a[i] = j;**

**b[i] = j;**

**c[i] = j;**

**}**

**cout << "shell() sort" << endl;**

**start\_time = clock();**

**shell(a, volume);**

**end\_time = clock();**

**compare\_time = end\_time - start\_time;**

**cout << "sort time: " << compare\_time / 1000.0 << endl << endl;**

**cout << "qs() sort" << endl;**

**start\_time = clock();**

**qs(b, 0, volume - 1);**

**end\_time = clock();**

**compare\_time = end\_time - start\_time;**

**cout << "sort time: " << compare\_time / 1000.0 << endl << endl;**

**cout << "qsort() sort" << endl;**

**start\_time = clock();**

**qsort(c, volume, sizeof(int), cmp);**

**end\_time = clock();**

**compare\_time = end\_time - start\_time;**

**cout << "sort time: " << compare\_time / 1000.0 << endl;**

**z++;**

**cout << "----------------------" << endl << endl;**

**cout << "now sorting half-increasing and half-decreasing sequence of numbers arrays" << endl;**

**for (int i = 0, j = volume-1; i < volume; i++, j--)**

**{**

**if (0 <= i <= volume/2-1)**

**{**

**a[i] = i;**

**b[i] = i;**

**c[i] = i;**

**}**

**else if (volume/2-1 <= i <= volume-1)**

**{**

**a[i] = j;**

**b[i] = j;**

**c[i] = j;**

**}**

**}**

**cout << "shell() sort" << endl;**

**start\_time = clock();**

**shell(a, volume);**

**end\_time = clock();**

**compare\_time = end\_time - start\_time;**

**cout << "sort time: " << compare\_time / 1000.0 << endl << endl;**

**cout << "qs() sort" << endl;**

**start\_time = clock();**

**qs(b, 0, volume - 1);**

**end\_time = clock();**

**compare\_time = end\_time - start\_time;**

**cout << "sort time: " << compare\_time / 1000.0 << endl << endl;**

**cout << "qsort() sort" << endl;**

**start\_time = clock();**

**qsort(c, volume, sizeof(int), cmp);**

**end\_time = clock();**

**compare\_time = end\_time - start\_time;**

**cout << "sort time: " << compare\_time / 1000.0 << endl;**

**delete[]a;**

**delete[]b;**

**delete[]c;**

**z++;**

**cout << "----------------------" << endl << endl;**

**cout << "done!" << endl;**

**}**

**int cmp(const void\* a, const void\* b)**

**{**

**{**

**return \*(int\*)a - \*(int\*)b;**

**}**

**}**

**void shell(int\* items, int count)**

**{**

**int i, j, gap, k;**

**int x, a[5];**

**a[0] = 9; a[1] = 5; a[2] = 3; a[3] = 2; a[4] = 1;**

**for (k = 0; k < 5; k++) {**

**gap = a[k];**

**for (i = gap; i < count; ++i) {**

**x = items[i];**

**for (j = i - gap; (x < items[j]) && (j >= 0); j = j - gap)**

**items[j + gap] = items[j];**

**items[j + gap] = x;**

**}**

**}**

**}**

**void qs(int\* items, int left, int right)**

**{**

**int i, j;**

**int x, y;**

**i = left; j = right;**

**x = items[(left + right) / 2];**

**do {**

**while ((items[i] < x) && (i < right)) i++;**

**while ((x < items[j]) && (j > left)) j--;**

**if (i <= j) {**

**y = items[i];**

**items[i] = items[j];**

**items[j] = y;**

**i++; j--;**

**}**

**} while (i <= j);**

**if (left < j) qs(items, left, j);**

**if (i < right) qs(items, i, right);**

**}**

**Количество чисел в массиве: 50000. Время работы алгоритма измеряется в мс**

**Сортировка Шелла   Быстрая сортировка Ф-я б. сорт.**

**Сл.набор.данных 213             4 3**

**Возраст.послед. 0 4 1**

**Убыв.посл. 359 7 1**

**Убыв-возраст. 1 6 3**

**100.000 чисел**

**Сортировка Шелла Быстрая сортировка Ф-я б. сорт.**

**Сл.набор.данных** 743             9 8

**Возраст.послед.** 1 10 2

**Убыв.посл.** 1261 12 3

**Убыв-возраст.** 2 9 4

**Вывод:**

1. Выяснили, что алгоритм быстрой сортировки, реализованный вручную, одинаково быстро работает на разных наборах данных в массиве
2. С увеличением количества данных в массиве, увеличивается, время, затрачиваемое на сортировку
3. Сортировка Шелла требует наибольше количество времени на работу, так как он сравнивает не только элементы, стоящие рядом, но и на определённом расстоянии
4. Функция быстрой сортировки из библиотеки требует большее количество времени, чем реализованный вручную