Передмова

Кодування

Мова моєї системи - англійська, тому способи на кшталт:

setlocale (LC\_ALL, "Russian");

Не допоможуть використовувати російський текст у консолі.

Тому я використовував спосіб який буде працювати на будь-якому комп'ютері: Я використовував 2-х байтові символи, що зберігають текст в кодуванні Unicode.

Для цього треба:

1. На початку програми необхідно налаштувати потоки виведення, введення і виведення помилок:

if (\_setmode(\_fileno(stdout), \_O\_U16TEXT) == -1) return -1;

if (\_setmode(\_fileno(stdin ), \_O\_U16TEXT) == -1) return -1;

if (\_setmode(\_fileno(stderr), \_O\_U16TEXT) == -1) return -1;

Для цього можуть знадобиться такі заголовки:

#include <io.h>

#include <fcntl.h>

1. Замість функцій printf і scanf\_s необхідно використовувати функції wprintf і wscanf\_s відповідно;
2. І також, замість std::cout і std::cin необхідно використовувати std::wcout і std::wcin;
3. Замість типу char необхідно використовувати тип wchar\_t;
4. Перед літералами рядків необхідно ставити букву L.

Загальний шаблон всіх програм

#include <iostream>

#include <io.h>

#include <fcntl.h>

//ToDo Другие заголовки

int main ()

{

if (\_setmode(\_fileno(stdout), \_O\_U16TEXT) == -1) return -1;

if (\_setmode(\_fileno(stdin ), \_O\_U16TEXT) == -1) return -1;

if (\_setmode(\_fileno(stderr), \_O\_U16TEXT) == -1) return -1;

//ToDo Тело программы

}

ЛР4: Побітові операції

Завдання

Формат команди каналу в система введення-виведення має вигляд:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **╧ розряду** | 15 | 14 | 13 | 12 | 11 | 10 | 09 | 08 | 07 | 06 | 05 | 04 | 03 | 02 | 01 | 00 |
| **значення** | O | O | O | O | O | C | D | N | N | N | N | N | N | N | N | N |

де:

* O..O - код операції
* C - ознака ланцюжка команд
* D - ознака ланцюжка даних
* N..N - кількість байтів для передачі

Коментарі до вирішення

Щоб уникнути дублювання коду - я додав 2 функції:

inline void WriteBits(uint16\_t& n, int bits, int from, int count)

{

auto mask = (1 << count) - 1;

n |= (bits & mask) << from;

}

inline int ReadBits(uint16\_t source, int from, int count)

{

auto mask = (1 << count) - 1;

return (source >> from) & mask;

}

Мінлива mask є числом, у якого тільки count нижніх біт заповнене одиницями, а всі інші біти - нулі.

Тип першого агрумент WriteBits - uint16\_t&, де & значить передачу по посиланню.

Так само, я додав макро RE\_READ\_ENCODED в початок програми, щоб можна було швидко вмикати / вимикати необхідність перезапису зашифрованого значення з клавіатури.

Увесь код

#include <iostream>

#include <io.h>

#include <fcntl.h>

#define RE\_READ\_ENCODED true

/// <summary>

/// Записывает count битов из числа bits в разряд from числа n

/// </summary>

inline void WriteBits(uint16\_t& n, int bits, int from, int count)

{

auto mask = (1 << count) - 1;

n |= (bits & mask) << from;

}

/// <summary>

/// Вырезает и возвращается область из count бит числа source, начиная с разряда from

/// </summary>

inline int ReadBits(uint16\_t source, int from, int count)

{

auto mask = (1 << count) - 1;

return (source >> from) & mask;

}

int main()

{

if (\_setmode(\_fileno(stdout), \_O\_U16TEXT) == -1) return -1;

if (\_setmode(\_fileno(stdin ), \_O\_U16TEXT) == -1) return -1;

if (\_setmode(\_fileno(stderr), \_O\_U16TEXT) == -1) return -1;

int oper\_code, command\_chain, data\_chain, byte\_count;

wprintf(L"Код операции (0..31): ");

wscanf\_s(L"%d", &oper\_code);

wprintf(L"Признак цепочки команд (0..1): ");

wscanf\_s(L"%d", &command\_chain);

command\_chain = command\_chain ? 1 : 0;

wprintf(L"Признак цепочки данных (0..1): ");

wscanf\_s(L"%d", &data\_chain);

data\_chain = data\_chain ? 1 : 0;

wprintf(L"Количество байтов для передачи (0..511): ");

wscanf\_s(L"%d", &byte\_count);

uint16\_t encoded{};

WriteBits(encoded, byte\_count, 0, 9);

WriteBits(encoded, data\_chain, 9, 1);

WriteBits(encoded, command\_chain, 10, 1);

WriteBits(encoded, oper\_code, 11, 5);

wprintf(L"В зашифрованном виде: 0x%X\n", encoded);

if (RE\_READ\_ENCODED)

{

wprintf(L"Введите зашифрованное состояние в 16СС: ");

int temp\_encoded;

wscanf\_s(L"%x", &temp\_encoded);

encoded = temp\_encoded;

}

wprintf(L"Код операции: %d\n", ReadBits(encoded, 11, 5));

wprintf(L"Признак цепочки команд: %d\n", ReadBits(encoded, 10, 1));

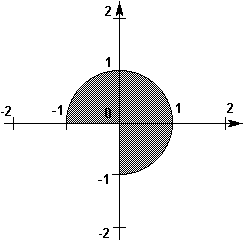
wprintf(L"Признак цепочки данных: %d\n", ReadBits(encoded, 9, 1));

wprintf(L"Количество байтов для передачи: %d\n", ReadBits(encoded, 0, 9));

}

ЛР 5: Умовний оператор

Завдання



Коментарі до вирішення

Спочатку я перевіряю чи потрапляє точка у коло з радіусом 1 і центром в (0; 0), потім - не\_прінадлежність до III чверті.

Взагалі, ще можна було б написати так:

auto res\_descr = умова? "Потрапляє": "НЕ потрапляє";

І при виведенні використовувати цей рядок, щоб мінімізувати дублі коду.

Увесь код

#include <iostream>

#include <io.h>

#include <fcntl.h>

int main()

{

if (\_setmode(\_fileno(stdout), \_O\_U16TEXT) == -1) return -1;

if (\_setmode(\_fileno(stdin ), \_O\_U16TEXT) == -1) return -1;

if (\_setmode(\_fileno(stderr), \_O\_U16TEXT) == -1) return -1;

wprintf(L"Введите [x] и [y] через пробел: ");

double x, y;

wscanf\_s(L"%lf %lf", &x, &y);

if (x \* x + y \* y < 1 \* 1 && (x > 0 || y > 0))

wprintf(L"Точка (x=%lf; y=%lf) попадает в область", x, y); else

wprintf(L"Точка (x=%lf; y=%lf) НЕ попадает в область", x, y);

}

ЛР 6: Оператори циклу

Завдання



Коментарі до коду

Для обчислення суми перших 10 елементів я використовував цикл for.

Для обчислення суми елементів >e так само можна було використовувати цикл for, ось так:

for (double last\_item, int i = 0; abs(last\_item = item\_at(i)) > e; ++i)

Це, крім усього іншого, трохи ефективніше, тому що ++i вважається після тіла циклу. Але так виглядає більш громіздко, тому я використав цикл while і розписав все докладніше.

Так само, я додав додаткові вкладені блоки коду {}, щоб відокремити області видимості змінних. Таким чином я можу оголосити 2 різні змінні sum в одній функції.

Увесь код

#include <iostream>

#include <io.h>

#include <fcntl.h>

const double e = 0.000001;

/// <summary>

/// Вычисляет элемент ряда в номер n (считая с 0)

/// </summary>

double item\_at(int n)

{

return

((n % 2) ? -1 : 1) \*

((double)n + 1) /

pow(2, n - 1);

}

int main()

{

if (\_setmode(\_fileno(stdout), \_O\_U16TEXT) == -1) return -1;

if (\_setmode(\_fileno(stdin ), \_O\_U16TEXT) == -1) return -1;

if (\_setmode(\_fileno(stderr), \_O\_U16TEXT) == -1) return -1;

{

auto sum = 0.0;

for (auto i = 0; i < 10; ++i)

sum += item\_at(i);

wprintf(L"Сумма первых 10 элементов равна %lf\n", sum);

}

{

auto sum = 0.0;

double last\_item;

auto i = 0;

while (abs(last\_item = item\_at(i++)) > e)

sum += last\_item;

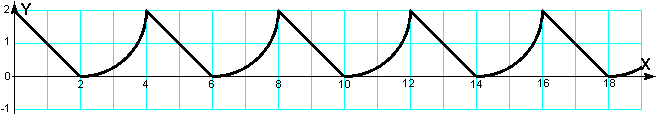
wprintf(L"Сумма первых %d элементов, каждый из которых >%e, равна %lf\n", i, e, sum);

}

}

ЛР 7: Оператори циклу

Завдання



Коментарі до коду

Для можливості легко змінювати основні параметри графіка я додав наступні макро для кордонів значень іксів:

#define XMin 0.0

#define XMax 18.0

І, для цього ж, записав цю математичну функцію у вигляді програмної:

double f(double x)

{

auto pos = fmod(x, 4);

return pos < 2 ?

2 - pos :

2 - sqrt(4 \* pos - pos \* pos);

}

Графік я розділив на лінії, кожна з яких має висоту = 1 рядку консолі. Малювання 1 лінії я зробив у вигляді функції draw\_line.

При цьому весь графік маштабірован так, щоб уміщатися в рядки візуального буфера консолі.

Ну і пару слів про graph\_x\_scale: У мене символи консолі мають розмір 8x16 пікселів, тому графік треба ще разстянуть в 2 рази, щоб він виглядав рівно.

Я знаю що розмір пікселів можна отримати програмно, але я не хотів додавати такий складний код в лабораторну. Через коду для графіків програма і так роздулася, а потім її вёрд ріже між сторінками ...

Увесь код

#include <iostream>

#include <io.h>

#include <fcntl.h>

// Для получения размера окна консоли

#include <windows.h>

#define CONSOLE\_HANDLE GetStdHandle(STD\_OUTPUT\_HANDLE)

using namespace std;

#define XMin 0.0

#define XMax 18.0

double f(double x)

{

auto pos = fmod(x, 4);

return pos < 2 ?

2 - pos :

2 - sqrt(4 \* pos - pos \* pos);

}

BOOL draw\_line(double x1, double y1, double x2, double y2)

{

auto point\_c = (int)ceil( max(abs(x1-x2), abs(y1-y2)) + 1);

for (auto i = 0; i < point\_c; ++i)

{

if (!SetConsoleCursorPosition(CONSOLE\_HANDLE, COORD{

(SHORT)round(x1 + (x2 - x1) \* (i / ((double)point\_c - 1))),

(SHORT)round(y1 + (y2 - y1) \* (i / ((double)point\_c - 1)))

})) return false;

wcout.put(L'\*');

}

return true;

}

int GetConsoleWindowHeigth()

{

CONSOLE\_SCREEN\_BUFFER\_INFO info;

if (!GetConsoleScreenBufferInfo(CONSOLE\_HANDLE, &info))

return -1;

return info.srWindow.Bottom - info.srWindow.Top + 1;

}

int main()

{

if (\_setmode(\_fileno(stdout), \_O\_U16TEXT) == -1) return -1;

if (\_setmode(\_fileno(stdin ), \_O\_U16TEXT) == -1) return -1;

if (\_setmode(\_fileno(stderr), \_O\_U16TEXT) == -1) return -1;

auto lines\_c = GetConsoleWindowHeigth();

if (lines\_c == -1) return -1;

#if \_DEBUG

lines\_c -= 4; // В конце выводит "press Enter to exit" и т.п.

#else \_DEBUG

lines\_c -= 1;

#endif \_DEBUG

const auto graph\_offset = 25;

auto graph\_scale = ((double)lines\_c - 1) / (XMax - XMin);

const auto graph\_x\_scale = 2.0;

double prev\_x{}, prev\_y{};

for (auto i = 0; i < lines\_c; ++i)

{

double graph\_x = XMin + (XMax - XMin) \* i / (lines\_c - (double)1);

double graph\_y = f(graph\_x);

if (!SetConsoleCursorPosition(CONSOLE\_HANDLE, COORD{ 0, (SHORT)i })) return -1;

wprintf(L"x = %5.2lf; y = %4.2lf |", graph\_x, graph\_y);

double x = graph\_y \* graph\_scale \* graph\_x\_scale + graph\_offset;

double y = graph\_x \* graph\_scale;

if (i != 0)

if (!draw\_line(prev\_x, prev\_y, x, y)) return -1;

prev\_x = x;

prev\_y = y;

}

}

ЛР 8: Робота з масивами

Завдання

|  |  |  |  |
| --- | --- | --- | --- |
| **№ варіанту** | **Размерностьмассіва** | **Діапазонзначеній** | **Що потрібно зробити** |
| **9** | 200 | 0 - 100 | Знайти безперервний ділянку з 10 елементів, сума яких максимальна |

Коментарі до коду

Функцію rand() я обернув в Random, для отримання випадкового числа в діапазоні.

Найпростіше рішення - складати по 10 елементів, починаючи з кожного з перших 191 елементів. Але це купа зайвих складань.

Краще спочатку знайти суму перший 10 елементів, а потім, для переміщення до наступного сумі - віднімати значення першого елемента суми і додавати наступний після ділянки суми елемент.

Увесь код

#include <iostream>

#include <io.h>

#include <fcntl.h>

using namespace std;

const int arr\_len = 200;

const int chain\_len = 10;

inline int Random(int x, int y)

{

return rand() % (y - x + 1) + x;

}

int main()

{

if (\_setmode(\_fileno(stdout), \_O\_U16TEXT) == -1) return -1;

if (\_setmode(\_fileno(stdin ), \_O\_U16TEXT) == -1) return -1;

if (\_setmode(\_fileno(stderr), \_O\_U16TEXT) == -1) return -1;

int arr[arr\_len];

for (auto i = 0; i < arr\_len; ++i)

arr[i] = Random(0, 100); // Можно и rand() % 101, но так код более читабельный //ToDo в отчёт

for (auto i = 0; i < arr\_len; ++i)

wcout << arr[i] << ' ';

wcout << endl;

int sum = 0;

for (auto i = 0; i < chain\_len; ++i)

sum += arr[i];

int max\_sum = sum;

int max\_sum\_ind = chain\_len - 1;

for (auto i = chain\_len; i < arr\_len; ++i)

{

sum -= arr[i - chain\_len];

sum += arr[i];

if (sum > max\_sum)

{

max\_sum = sum;

max\_sum\_ind = i;

}

}

wcout << L"Участок с максимальной суммой:" << endl;

for (auto i = max\_sum\_ind - (chain\_len-1); i < max\_sum\_ind; ++i)

wcout << arr[i] << ' ';

wcout << endl;

wcout << L"Имеет сумму: " << max\_sum << endl;

}

ЛР 9: Робота з матрицями

Завдання

|  |  |  |
| --- | --- | --- |
| **№ варіанту** | **зміст завдання** | **Ілюстрація** |
| **9** | Заповнити матрицю ЛП, від лівого нижнього кута по діагоналі: вліво - вгору. |  |

Коментарі до коду

Для простоти алгоритму я розділив процес заповнення матриці на 2 частини: під- і над головною діагоналлю.

Решта дуже просто, i1 перераховує діагоналі, а i2 - елементи в поточній діагоналі.

Увесь код

#include <iostream>

#include <iomanip>

#include <io.h>

#include <fcntl.h>

using namespace std;

const int matr\_w = 9;

int main()

{

if (\_setmode(\_fileno(stdout), \_O\_U16TEXT) == -1) return -1;

if (\_setmode(\_fileno(stdin ), \_O\_U16TEXT) == -1) return -1;

if (\_setmode(\_fileno(stderr), \_O\_U16TEXT) == -1) return -1;

int mtr[matr\_w][matr\_w];

int c = 1;

// Заполнение части под главной диагональю (включительно)

for (auto i1 = 0; i1 < matr\_w; ++i1)

for (auto i2 = 0; i2 <= i1; ++i2)

mtr[matr\_w - 1 - i2][i1 - i2] = c++;

// Заполнение части над главной диагональю (НЕ\_включительно)

for (auto i1 = 1; i1 < matr\_w; ++i1)

for (auto i2 = 0; i2 < matr\_w-i1; ++i2)

mtr[matr\_w-1-i1 - i2][matr\_w-1 - i2] = c++;

for (auto row = 0; row < matr\_w; ++row)

{

for (auto col = 0; col < matr\_w; ++col)

wcout << setw(3) << mtr[row][col];

wcout << endl;

}

}

РГЗ: Методи сортування

Завдання

Варіант 9: Radix sort і Tree sort.

Коментарі до коду

Заради уникнення дублів коду я оголосив структуру test\_info, що містить 1 поле (адреса функції сортування) і метод, що виконує цю функцію + заміряє, скільки часу знадобилося на її виконання.

І, для того ж, я додав функцію CombineTestsArr, приймаючу список адрес функцій і створює статичний масив структур test\_info.

Все що стосується самих алгоритмів сортувань я поклав в окремий простір імен SortAlgorithms, щоб не змішуватиїх з іншим кодом.

І, так само, допоміжні типи і функції для кожної сортування я поклав в відповідне простір імен всередині SortAlgorithms.

При цьому у TreeSort 2 пронстранства імен допоміжних типів (Tree і Tree2). Перший простір імен містить прямий спосіб реалізації TreeSort, з виділенням пам'яті в міру створення нових елементів. Це виявилося черезвичаяно повільно, тому я зробив другий варіант, де пам'ять виділяється 1 раз, у вигляді динамічного масиву, відразу під всі вузли дерева.

Порівняння сортувань

У даній реалізації TreeSort значно гірше RadixSort.

Особливо відрив спостерігається на відсортованих (в будь-яку сторону) даних, тому що заповнюються тільки чи всі ліві гілки, або всі праві.

У підсумку виходить одинзв'язні список і O(n^2) перевірок, що чітко видно на вимірах часу.

Найпростіша ідея - зберігати значення і адреса мінімального і максимального вузла. Це допоможе в гірших випадках, але алгоритм все одно не буде швидше RadixSort, що має складність O(n).

Перевага у TreeSort з'являється тільки при більш маленьких n, тому що він простіший, ніж RadixSort.

Тому ж я не беру до уваги варіант використання само-балансир дерева. Воно може тільки ускладнити алгоритм, зробивши його повільніше для маленьких n, при цьому все одно не давши перевагу на великих n, тому що алгоритм сортування, який використовує порівняння, не може бути краще O(n).

У той же час, хоч алгоритм RadixSort і більш складний, його легко распараллелити. Тобто, якщо є GPU з кількістю ядер >=n, складність алгоритму можна зробити, по суті, O(1). Звичайно, час все одно буде залежить від n, хоча б через збільшення витрат пам'яті (і відповідно часу на її виділення).

Увесь код

#include <iostream>

#include <io.h>

#include <fcntl.h>

#include <type\_traits>

#include <array>

#include <chrono>

#include <windows.h>

#include "local\_utils.h"

using namespace std;

using namespace std::chrono;

#define TEST\_COUNT 10000

namespace ConsoleCursor {

HANDLE hnd = GetStdHandle(STD\_OUTPUT\_HANDLE);

COORD GetCoord() {

CONSOLE\_SCREEN\_BUFFER\_INFO cbsi;

if (GetConsoleScreenBufferInfo(hnd, &cbsi))

return cbsi.dwCursorPosition;

COORD invalid{};

return invalid;

}

void Teleport(int X, int Y) {

SetConsoleCursorPosition(hnd, COORD{ (SHORT)X, (SHORT)Y });

}

void TeleportOn(int dX, int dY) {

auto pos = GetCoord();

Teleport(pos.X + dX, pos.Y + dY);

}

struct {

int& operator = (int& new\_x) {

Teleport(new\_x, GetCoord().Y);

return new\_x;

}

operator int() { return GetCoord().X; }

} X;

struct {

int& operator = (int& new\_y) {

Teleport(GetCoord().X, new\_y);

return new\_y;

}

operator int() { return GetCoord().Y; }

} Y;

}

template <class CALLABLE>

struct test\_info {

CALLABLE proc;

test\_info(CALLABLE proc) {

this->proc = proc;

}

template <class TData>

auto Execute(TData a[], int len) {

nanoseconds res{};

auto data = new TData[len];

for (auto test = 0; test < TEST\_COUNT; ++test)

{

for (auto i = 0; i < len; ++i) data[i] = a[i];

auto start = high\_resolution\_clock::now();

proc(data, len);

auto stop = high\_resolution\_clock::now();

res += stop - start;

}

delete[] data;

return duration\_cast<milliseconds>( res ).count();

}

};

template <class ... CALLABLE>

auto CombineTestsArr(CALLABLE ... args) {

using common\_t = common\_type\_t<CALLABLE...>;

return array<test\_info<common\_t>, sizeof...(CALLABLE)> { test\_info<common\_t>(args)... };

}

namespace SortAlgorithms

{

namespace Radix {

template <class TData>

void CountSort(TData data[], size\_t len, TData temp[], int base, int exp) {

auto count = new int[base] {};

for (size\_t i = 0; i < len; ++i)

// Вообще это не будет работать для отрицательных значений...

// Для CountSort нужна функция ключа, возвращающая только не\_отрицательные значения

// Но в данном случае значения генерируются с помощью "rand() % ...", поэтому они всегда не\_отрицательные

++count[data[i] / exp % base];

for (int i = 1; i < base; ++i)

count[i] += count[i - 1];

// Перечисление в обратном порядке, чтоб не менять местами элементы с одинаковой цифрой в данном порядке

for (int i = len - 1; i >= 0; --i)

temp[--count[data[i] / exp % base]] = data[i];

for (size\_t i = 0; i < len; ++i) data[i] = temp[i];

}

}

template <class TData>

void RadixSort(TData data[], size\_t len) {

size\_t base = len;

decltype(base) m = data[0];

for (size\_t i = 1; i < len; ++i)

m = max(m, static\_cast<decltype(base)>(data[i]) );

auto temp = new TData[len];

for (decltype(base) exp = 1; exp <= m; exp \*= base)

Radix::CountSort(data, len, temp, base, exp);

delete[] temp;

}

namespace Tree {

template <class TData>

class Node {

private:

TData data;

Node\* l = NULL;

Node\* r = NULL;

public:

Node(TData data) : data(data) {};

void Add(TData new\_data) {

Node\*& branch = new\_data < data ? l : r;

if (branch)

branch->Add(new\_data);

else

branch = new Node(new\_data);

}

static Node\* MakeTree(TData a[], size\_t len) {

auto res = new Node(a[0]);

for (size\_t i = 1; i < len; ++i) res->Add(a[i]);

return res;

}

void StoreTo(TData\*& a) {

if (l) l->StoreTo(a);

\*(a++) = data;

if (r) r->StoreTo(a);

}

~Node() {

if (l) delete l;

if (r) delete r;

}

};

}

namespace Tree2 {

template <class TData>

class Node {

private:

TData data;

Node\* l = NULL;

Node\* r = NULL;

Node() : data() {};

public:

Node(TData data) : data(data) {};

void Add(TData new\_data, Node\*& last\_node) {

auto \_this = this;

Node\*\* el = &\_this;

while (\*el) el = new\_data < (\*\*el).data ? &(\*\*el).l : &(\*\*el).r;

\*(\*el = ++last\_node) = Node(new\_data);

}

static Node\* MakeTree(TData a[], size\_t len) {

auto res = new Node[len];

res[0] = Node(a[0]);

auto last\_node = res;

for (size\_t i = 1; i < len; ++i)

res->Add(a[i], last\_node);

return res;

}

void StoreTo(TData\*& a) {

if (l) l->StoreTo(a);

\*(a++) = data;

if (r) r->StoreTo(a);

}

};

}

template <class TData>

void TreeSort(TData data[], size\_t len) {

auto tree = Tree2::Node<TData>::MakeTree(data, len);

tree->StoreTo(data);

// Следущее понадобится только если data

// будет использоваться в этой функции ниже

// Или, лучше, посылать в StoreTo отдельную переменную temp

// data -= len;

// [] только для Tree2

delete[] tree;

}

template <class TData>

auto AllTestsArr = CombineTestsArr(

&RadixSort <TData>,

&TreeSort <TData>

);

}

#define test\_min\_size 20

#define test\_step\_size 20

#define test\_max\_size 300

template <class TData>

void TestIteration(TData a[], size\_t len) {

auto &tests = SortAlgorithms::AllTestsArr<TData>;

int x = ConsoleCursor::X;

int y = ConsoleCursor::Y;

wcout << len;

int max\_w = ConsoleCursor::X - x;

for (size\_t i = 0; i < tests.size(); ++i)

{

ConsoleCursor::Teleport(x, y + i + 1);

auto data = new TData[len];

for (size\_t i2 = 0; i2 < len; ++i2) data[i2] = a[i2];

wcout << tests[i].Execute(data, len);

max\_w = max(max\_w, ConsoleCursor::X - x);

}

ConsoleCursor::Teleport(x + max\_w + 1, y);

}

template <class TData>

void AllTestIterations(TData a[]) {

int x = ConsoleCursor::X;

int y = ConsoleCursor::Y;

for (size\_t len = test\_min\_size; len <= test\_max\_size; len += test\_step\_size)

TestIteration<TData>(a, len);

ConsoleCursor::Teleport(x, y + SortAlgorithms::AllTestsArr<TData>.size());

}

template <class TData>

void Println(TData a[], size\_t len) {

for (size\_t i = 0; i < len; ++i)

wcout << a[i] << L' ';

wcout << endl;

}

int main() {

if (\_setmode(\_fileno(stdout), \_O\_U16TEXT) == -1) return -1;

if (\_setmode(\_fileno(stdin ), \_O\_U16TEXT) == -1) return -1;

if (\_setmode(\_fileno(stderr), \_O\_U16TEXT) == -1) return -1;

auto data = new int[test\_max\_size];

for (auto i = 0; i < test\_max\_size; ++i)

data[i] = rand() % 500;

wcout << L"Не отсортированный массив:" << endl;

AllTestIterations(data);

wcout << endl << endl;

auto sorted = new int[test\_max\_size];

for (auto i = 0; i < test\_max\_size; ++i) sorted[i] = data[i];

SortAlgorithms::RadixSort(sorted, test\_max\_size);

wcout << L"Отсортированный массив:" << endl;

AllTestIterations(sorted);

wcout << endl << endl;

auto reversed = new int[test\_max\_size];

for (auto i = 0; i < test\_max\_size; ++i) reversed[i] = sorted[test\_max\_size - i - 1];

wcout << L"Отсортированный в обратном порядке массив:" << endl;

AllTestIterations(reversed);

wcout << endl << endl;

{

size\_t const integrity\_test\_size = 20;

wcout << L"Тестирование адекватности сортировок:" << endl;

Println(data, integrity\_test\_size);

auto temp = new int[integrity\_test\_size];

for (auto t : SortAlgorithms::AllTestsArr<int>)

{

for (size\_t i = 0; i < integrity\_test\_size; ++i) temp[i] = data[i];

t.proc(temp, integrity\_test\_size);

Println(temp, integrity\_test\_size);

}

delete[] temp;

}

delete[] data;

delete[] sorted;

delete[] reversed;

}