Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

**Санкт-Петербургский национальный исследовательский университет**

**информационных технологий, механики и оптики**

## Факультет Фотоники и оптоинформатики

Кафедра Компьютерной фотоники и видеоинформатики

**Отчет по практике**

Выполнил: Ермаков Дмитрий

Группа: 3362

Преподаватель: Кудрявцев А. С.

Санкт-Петербург, 2015

# Лабораторная работа №3

### Задача

Напечатать все пары "дружественных" чисел, не превосходящих заданного натурального числа.

### Теория

Дружественные числа — это два различных натуральных числа, для которых сумма всех собственных делителей первого числа равна второму числу и наоборот, сумма всех собственных делителей второго числа равна первому числу.

Например, числа 220 и 284: делители числа 220 это 1, 2, 4, 5, 10, 11, 20, 22, 44, 55, 110; в сумме эти числа дают 284. И наоборот делители числа 284 в сумме дают число 220.

### Входные данные

Натурально число N, которое является границей поиска дружественных чисел.

### Результат

Дружественнее числа в диапазоне от 0 до N.

### Метод решения

Создадим массив, элементы которого равны 0 и количество которых равно N+1.

Заполняем этот массив суммами:

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

{

for (int j = i + i; j <= N; j += i)

arr[j] += i;

}

Далее из массива сумм выводим значения, которые нужны нам, а следовательно будут удовлетворять условиями: сумма входит в диапазон (меньше или равна N), число не должно быть равно своей же сумме делителей и числа должны быть парными:

if (arr[i] <= N && arr[i] != i && arr[arr[i]] == i)

{

cout << arr[i] << "\t " << i << std::endl;

arr[i] = 0;

}

### Пример

|  |  |
| --- | --- |
| Ввод | Вывод |
| 400 | 284 220 |
| 8000 | 284 220  1210 1184  2924 2620  5564 5020  6368 6232 |

### Приложение

Код программы на языке С++

#include <iostream>

using namespace std;

int main()

{

int N;

cin >> N;

int\* arr = new int[N + 1];

for (int i = 0; i <= N; i++)

{

arr[i] = 0;

}

int m = N / 2;

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

{

for (int j = i + i; j <= N; j += i)

arr[j] += i;

}

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

{

if (arr[i] <= N && arr[i] != i && arr[arr[i]] == i)

{

cout << arr[i] << "\t " << i << std::endl;

arr[i] = 0;

}

}

}

# Лабораторная работа №5

### Задача

Бандиты Гарри и Ларри отдыхали на природе. Решив пострелять, они выставили на бревно несколько банок из-под пива (не больше 10). Гарри начал простреливать банки по порядку, начиная с самой левой, Ларри — с самой правой. В какой-то момент получилось так, что они одновременно прострелили одну и ту же последнюю банку.

Гарри возмутился и сказал, что Ларри должен ему кучу денег за то, что тот лишил его удовольствия прострелить несколько банок. В ответ Ларри сказал, что Гарри должен ему еще больше денег по тем же причинам. Они стали спорить кто кому сколько должен, но никто из них не помнил сколько банок было в начале, а искать простреленные банки по всей округе было неохота. Каждый из них помнил только, сколько банок прострелил он сам.

Определите по этим данным, сколько банок не прострелил Гарри и сколько банок не прострелил Ларри.

### Входные данные

Количество банок, в которые попал Ларри и количество банок, в которые попал Гарри.

### Результат

Дружественнее числа в диапазоне от 0 до N.

### Метод решения

Нам известно, что последнюю банку подстелили оба бандита, а значит общее количество банок будет равно сумме подстреленных банок минус один:

int vse = larry + garry - 1;

Но эта сумма не больше 10 по условию задачи. В так случае, если общее количество банок получится больше 10, то кто-то из бандитов наврал и назвал число, которое не равно количеству им подстреленных банок. Эта проверка реализуется так:

if (vse > 10)

{

cout << endl << "kto-to vret" << endl;

}

else

{

cout << endl << vse - larry << " " << vse - garry << endl;

}

А количество неподстреленных банок каждым из бандитов будет являться разностью всех банок и банок, которые подстрелил этот бандит.

### Пример

|  |  |
| --- | --- |
| Ввод | Вывод |
| 4 7 | 6 3 |
| 2 2 | 1 1 |
| 6 7 | kto-to vret |

### Приложение

Код программы на языке С++

#include "iostream"

using namespace std;

int main()

{

int larry;

cin >> larry;

int garry;

cin >> garry;

int vse = larry + garry - 1;

if (vse > 10)

{

cout << endl << "kto-to vret" << endl;

}

else

{

cout << endl << vse - larry << " " << vse - garry << endl;

}

}

# Лабораторная работа №6

### Задача

У сороконожки 40 левых ножек и 40 правых ножек. Под кроватью у сороконожки a левых тапочек и b правых тапочек. Сороконожка, просыпаясь, надевает тапочки. Для этого она засовывает под кровать первую левую ножку и надевает первый попавшийся тапочек, тратя на это одну секунду. Если тапочек оказывается левым, то она переходит ко второй левой ножке. Если же он оказывается правым, она переодевает его на какую-нибудь необутую правую ножку, тратя ещё одну секунду, то есть всего на такой тапочек уходит две секунды. Если все правые ножки уже обуты, то она снимает тапочек и кидает его в угол комнаты, тратя на это одну секунду, то есть на такой тапочек сороконожка тратит также две секунды. Процесс продолжается до тех пор, пока все левые ножки не окажутся в левых тапочках. Затем сороконожка аналогичным образом начинает надевать правые тапочки, продолжая до тех пор, пока не будут обуты все правые ножки.

Сегодня сороконожка встала не с той ножки, поэтому она готовится к худшему. Несмотря на это, она, как обычно, начинает обуваться с левой ножки. Сколько секунд понадобится сороконожке на утреннее обувание?

### Входные данные

Натуральные числа a и b, которые являются количеством левых и правых тапочек соответственно и находятся в диапазоне от 40 до 100 включительно.

### Результат

Количество секунд sec, потраченных на одевание тапочек в худшем случае.

### Метод решения

Рассмотрим два варианта событий и выберем из них наихудшей (где результат равен наибольшему числу):

1. Сороконожка натыкается на первых 40 правых тапочек под кроватью, тратя на это 2 секунды на тапочек (40 \* 2). После обувания правых ножек ее ситуация не улучшается и ей продолжают попадать правые тапочки, которые она бросает в угол и тратит 2 секунды на тапочек (2 \* (b - 40)) . И только после того, как в кучке кончаются все правые тапочки, она обувает все левые ножки тратя на это 40 секунд: 2 \* 40 + 2 \* (b - 40) + 40 = 80 + 2b - 80 + 40 = 2b + 40.

2. Сороконожке не везет и первые 39 тапочек попадаются правыми и каждый требует по 2 секунды на надевание на правую ножку (39 \* 2). Затем ей начинает везти и следующие 40 тапочек она надевает на левые ножки за 40 секунд. И вот когда осталась одна правая ножка, ей попадаются только левые тапочки, которые она выкидывает в угол (2 \* (a - 40)). Ну а после окончания в кучке левых тапочек, ей попадается последний правый: 39 \* 2 + 40 + 2 \* (a - 40) + 1 = 119 + 2a - 80 = 39 + 2a.

### Пример

|  |  |
| --- | --- |
| Ввод | Вывод |
| 40 40 | 120 |
| 80 60 | 199 |
| 60 80 | 200 |
| 60 60 | 160 |

### Приложение

Код программы на языке С++

#include <iostream>

using namespace std;

int main()

{

int a, b, sec, sec1, sec2;

cin >> a >> b;

sec1 = 2 \* b + 40;

sec2 = 39 + 2 \* a;

if (sec1 > sec2)

{

sec = sec1;

}

else

{

sec = sec2;

}

cout << sec << endl;

}

# Лабораторная работа №7

Я работал в компании Merkeleon, которая занимается разработкой интернет-аукционов, разработка высоконагруженного проекта с realtime составляющей, т.е. сайта, на котором в зависимости от действий пользователей будут моментально подгружаться данные без обновления страницы.

Программирование происходило на языке JavaScript на программной платформе Node.js.

Я был одним из ведущих разработчиков backend и занимался созданием нового модуля аукциона, работа которого заключалась в том, чтобы несколько параллельно-идущих и независимых друг от друга аукционов ежесекундно проверяли заявку на изменение данных от пользователя с минимальной погрешностью.

Разработка велась на двух типах баз данных: реляционной (PostgreSQL) и нереляционной (Redis). Нереляционная база обеспечивает более быстрый доступ к данным, чем реляционная, поэтому она служит для хранения часто изменяемых значений. После окончания аукциона данные перекидываются в реляционную бд, где им обеспечивается надежное хранение.

Были использованы такие библиотеки, как Async, Sequelize.js, Express.js.

Шаблонизатор: Handlebars.

Для обеспечения быстрой связи сервера и клиента использовались Socket.io.

Ниже представлен код с пояснениями, который отвечает за ежесекундную проверку изменения ставок в скандинавском аукционе.

