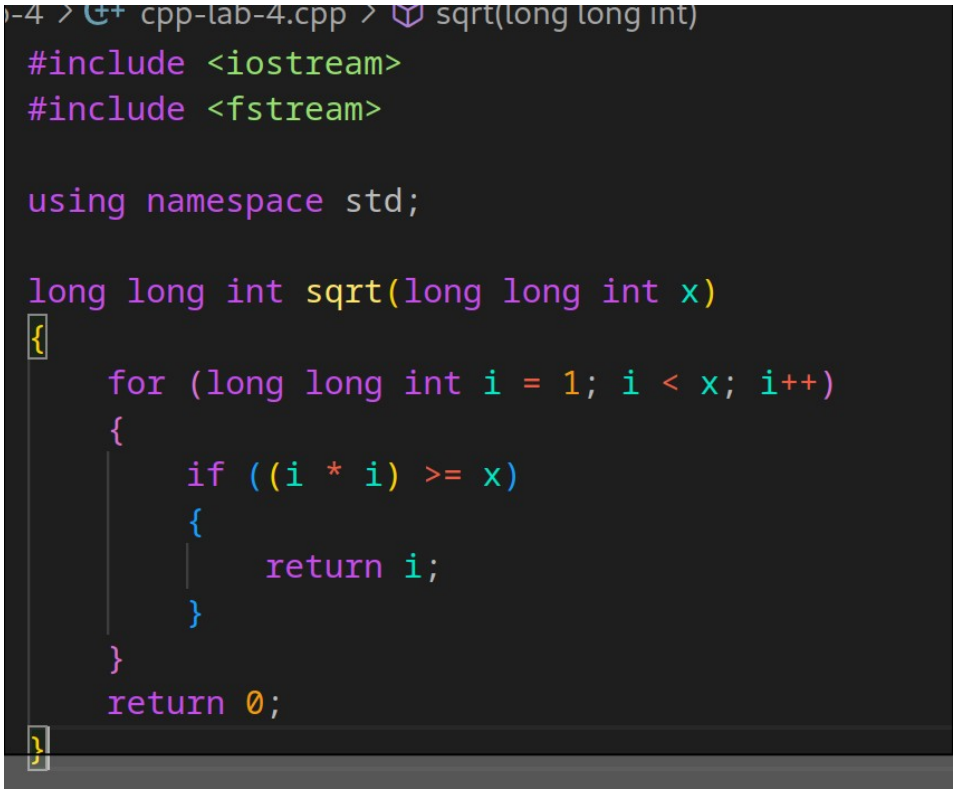
Сидоров Евгений СКБ241
Лабораторная работа 4

Задача: Маша и Петя договорились купить суп (включенный в комплексный обед за 180рублей) на двоих. Они поделили суп на nмисок, в каждой из которых было i грамм борща ($1 \le i \le n$) так, чтобы ни одна масса тарелочки не повторялась дважды. Помогите им понять, можно ли поесть поровну супа и, если всё-таки можно, то выведите массы мисок, которые задействует каждый из первокурсников.

Логика решения:
По формуле арифметической прогрессии S(сумма всех чашек) = (1 + n) * n / 2, соответственно половина прогрессии = (1 + n) * n / 4 ==> чтобы можно было получить половину, число n должно делиться на 4 или давать остаток 3 при деление на 4. Если n соответствует этому условию, то можно комбинацией чашек получить половину. Для её получения будем брать наибольшие чашки и складывать их, пока получающаяся сумма меньше чем половина суммы всех чашек. Когда мы не можем больше прибавить следующую наибольшую чашку, это будет означать, что где-то в оставшихся чашках есть та, которую можно добавить к нашей сумме и получить половину.
Можно ускорить программу, если сразу найти индекс i, такой что сумма последовательности от i до n будет меньше или равна половине суммы. Тогда i можно найти по формуле i = (2 + sqrt(1 + 2 * n + 2 * n * n)) / 2
А далее нам остаётся лишь понять, какой элемент добавить к этой сумме и вывести ответ в файл, т.к вывод в консоль занимает больше времени

Алгоритм:



```cpp
int main()
{
    long int n;
    cin >> n;
    long long int summa = (n + 1) * n / 4;

    ofstream file("output.txt");

    if (n % 4 == 0 || n % 4 == 3)
    {
        file << "YES" << endl;

        long int i = (2 + sqrt(1 + 2 * n + 2 * n * n)) / 2, l = n + 1 - i, extension = 0;

        if (((i + n) * (n + 1 - i) / 2) != summa)
        {
            l += 1;
            extension = summa - ((i + n) * (n + 1 - i) / 2);
        }
```

Считываем n, проверяем, можно ли вообще получить половину, записываем YES в файл, находим i по формуле и число, которое нужно добавить(extension) (если сумма от i до n уже равна половине, то extension = 0).

```cpp
        file << l << endl;

        if (extension != 0)
        {
            file << extension << " ";
        }

        for (long long int j = i; j <= n; j++)
        {
            file << j << " ";
        }

        file << endl;
        file << n - l << endl;

        for(long long int k = 1; k < i; k ++)
        {
            if(k != extension)
            {
                file << k << " ";
            }
        }

        file << endl;

    }
    else
    {
        file << "NO" << endl;
    }
    file.close();
    return 0;
```

Теперь просто записываем в файл нужные числа или NO, если для n невозможно получить половину

Тесты

```
int main()
{
    long int n;
    fstream input_file("input.txt");
    while (input_file >> n)
    {
        auto start = chrono::high_resolution_clock::now();
        result(n);
        auto end = chrono::high_resolution_clock::now();
        chrono::duration<double> duration = end - start;

        if(test(n))
        {
            cout << "Тест  пройден за " << duration.count() << " секунды" << endl;

        }
        else
            cout << "Тест не пройден" << endl;
    }
    return 0;
}
```

В тестах изначальный код программы помещён в функцию result, из входного файла (input.txt) считывается n с каждой строки. Далее выполняется сам алгоритм (функция result). И правильность алгоритма проверяется функцией test

```
int test(long int n)
{
    char word[4];

    fstream output_file("output.txt");
    output_file >> word;
    if (n % 4 == 0 || n % 4 == 3)
    {
        if (strcmp("YES", word))
        {
            long int target = (1 + n) * n / 4, count, digit, ans_sum = 0, *digits = new long[n], i;
            output_file >> count;

            for (i = 0; i < count; ++i)
            {
                output_file >> digit;

                if (is_in(digit, digits, n))
                {
                    delete[] digits;
                    return 0;
                }

                digits[i] = digit;
                ans_sum += digit;
            }

            if (ans_sum != target)
            {
                delete[] digits;
                return 0;
            }

            ans_sum = 0;
            output_file >> count;
```

Функция test поверяет правильность первого слова в выходном файле: для n, которые делятся на 4 без остатка или с остатком 3 должно быть YES

```
int strcmp(const char *s_1, const char *s_2)
{
    int i = 0;
    for (; s_1[i] != '\0' && s_2[i] != '\0'; ++i)
    {
        if (s_1[i] != s_2[i])
        {
            return 0;
        }
    }
    return (s_1[i] == '\0' && s_2[i] == '\0');
}
```

Функция strcmp сравнивает две C — строки. Далее если слово записано правильно, то в случае. если наше ответ — YES, нужно просто сложить все числа первой комбинации и второй и проверить, что они равны друг другу и половине всей возможной суммы. Подсчёт первой комбинации представлен на первом скриншоте функции test. Подсчёт второй комбинации представлен ниже

```
        for (; i < n; ++i)
        {
            output_file >> digit;

            if (is_in(digit, digits, n))
            {
                delete[] digits;
                return 0;
            }

            digits[i] = digit;
            ans_sum += digit;
        }

        delete[] digits;
        return ans_sum == target;
    }
    return 0;
}
return strcmp("NO", word);
```

Теперь осталось разобраться с динамическим массивом digits, в который мы будем записывать все числа, которые использовались и проверять, не повторяется ли какое-то число дважды. Для этого каждый раз перед добавлением цифры проверяем, не было ли её ранее (функция is_in). И главное — нужно не забыть освободить память перед выходом из функции.

```
// Не работает при больших n, так как долгий перебор
int is_in(long int number, long int *array, long int len)
{
    // for (long int index = 0; index < len; ++index)
    // {
    //     if (array[index] == number)
    //         return 1;
    // }
    return 0;
}
```

Разберёмся теперь с моим комментарием, при больших n получается большой массив, который мы постоянно перебираем при вызове функции is_in, поэтому при тестировании больших N данная проверка не работает, точнее работает, но долго

Вывод: в ходе выполнения этой лабораторной работы я в большинстве работал с выводом и вводом файл и использовал математику для решения задачи, также я использовал динамические массивы
с
С кодом всей программы можно ознакомится на https://github.com/EvgehS/labs