ГУАП

КАФЕДРА № 42

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

доцент		В. А. Кузнецов
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ № 3.2

АЛГОРИТМ С ИСПОЛЬЗОВАНИЕМ РЕКУРСИИ И ЦИКЛОВ

по курсу:

ОСНОВЫ ПРОГРАММИРОВАНИЯ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ гр. № 4326		Г. С. Томчук
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2024

СОДЕРЖАНИЕ

1 Постановка задачи

Задача: реализовать программную функцию на языке C/C++, выполняющую поставленную задачу с использованием рекурсии. Вариант задания, пример входных и выходных данных представлен в таблице 1.

- Написать код функции, принимающей в качестве аргументов и возвращающей все необходимые параметры, без использования глобальных переменных. Допустимо использование дополнительных функций.
- Протестировать функцию для всех возможных исключительных ситуаций, особое значение придается тестам на возникновение ошибок в ходе работы программы.
- Из наименования функции и принимаемых аргументов должно быть ясно их назначение.
- В ходе тестирования функции при каждом вызове рекурсивной функции необходимо вывести отладочную информацию: порядковый номер вызова рекурсивной функции, значения изменяющегося аргумента и возвращаемого значения, если они присутствуют. Привести глубину рекурсии для каждого тестового примера.

Таблица 1 – Вариант

N	Текст задания	Вход	Выход
2	Дан одномерный массив A размерностью N. Сохранить бинарно в файл все возможные размещения элементов массива из K элементов.	N: 3 A: [1,2,3] K: 2	[1,2] [1,3] [2,1] [2,3] [3,1] [3,2]

2 Схема алгоритма решения

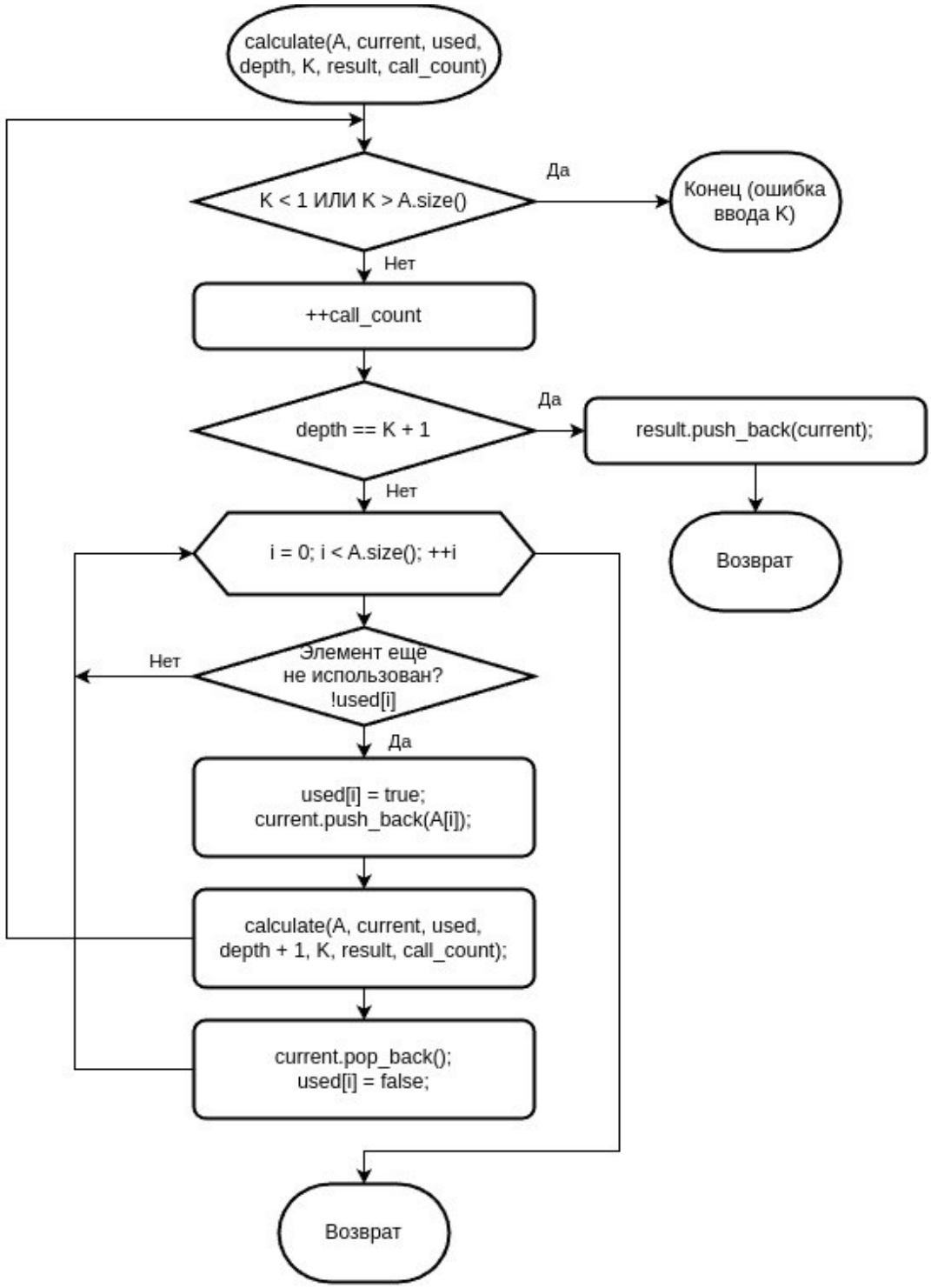

Рисунок 1 – Блок-схема алгоритма

3 Полное описание реализованных функций

3.1 get_partial_permutations_with_debug

Функция get_partial_permutations_with_debug генерирует все возможные размещения длины K из элементов массива A и сохраняет их в вектор result. В процессе работы функция выводит отладочную информацию о каждом вызове рекурсии, глубине вызова и состоянии текущего размещения. Принимает следующие аргументы:

1. const std::vector<int> &A: Исходный массив элементов.
2. std::vector<int> &current: Вектор для хранения текущего частичного размещения.
3. std::vector<bool> &used: Вектор флагов для отслеживания использования элементов.
4. int depth: Текущая глубина рекурсивного вызова.
5. int K: Длина размещения.
6. std::vector<std::vector<int>> &result: Вектор для хранения всех найденных размещений.
7. int &call_count: Счётчик вызовов функции для отладки.

Ничего не возвращает, результат записывается в result. Работа функции происходит следующим образом:

1. Проверяет корректность значений K.
2. Увеличивает счётчик вызовов и выводит отладочную информацию.
3. Если текущая глубина равна K+1, сохраняет текущее размещение и возвращается из рекурсии.
4. В противном случае, перебирает все элементы массива A и, если элемент ещё не использован, добавляет его в текущее размещение и вызывает рекурсивную функцию с увеличенной глубиной.
5. После возвращения из рекурсивного вызова удаляет последний элемент из текущего размещения и отмечает элемент как неиспользованный.

3.2 write_to_file

Функция write_to_file записывает в бинарный файл все размещения,

переданные в виде вектора векторов целых чисел. Принимает следующие аргументы:

1. const std::string &filename: Имя файла для записи.
2. const std::vector<std::vector<int>> &permutations: Вектор векторов, содержащий все размещения.

Ничего не возвращает. Работа функции происходит следующим образом:

1. Открывает файл для записи в бинарном режиме.
2. Проверяет успешность открытия файла, если неудачно — выводит сообщение об ошибке и завершает программу.
3. Пишет каждое размещение в файл, используя метод write для побайтовой записи данных.
4. Закрывает файл.

3.3 read_from_file

Функция read_from_file читает из бинарного файла все размещения и возвращает их в виде вектора векторов целых чисел. Принимает следующие аргументы:

1. const std::string &filename: Имя файла для чтения.
2. int K: Длина размещения.

Возвращает вектор векторов, содержащий все прочитанные размещения. Работа функции происходит следующим образом:

1. Открывает файл для чтения в бинарном режиме.
2. Проверяет успешность открытия файла, если неудачно — выводит сообщение об ошибке и завершает программу.
3. Читает данные из файла в буфер фиксированного размера (размера K), пока файл не закончится.
4. Сохраняет каждое прочитанное размещение в результирующий вектор.
5. Закрывает файл и возвращает результирующий вектор.

3.4 print_permutations

Функция print_permutations выводит все размещения, переданные в виде вектора векторов целых чисел. Принимает следующие аргументы:

1. const std::vector<std::vector<int>> &permutations: Вектор векторов, содержащий все размещения.

   Ничего не возвращает. Работа функции происходит следующим образом:

1. Перебирает каждое размещение из переданного вектора.
2. Для каждого размещения выводит его элементы в консоль.

3.5 main

1. Вводит массив элементов и значение K.
2. Генерирует все возможные размещения длины K из элементов массива A с помощью функции get_partial_permutations_with_debug.
3. Записывает найденные размещения в бинарный файл permutations.bin.
4. Читает размещения из файла и выводит их на экран.
5. Выводит глубину рекурсии для данной задачи.

4 Листинг программы

Листинг 1

```
#include <iostream>
#include <fstream>
#include <vector>
#include <sstream>

void get_partial_permutations_with_debug(const std::vector<int> &A,
std::vector<int> &current,
                                         std::vector<bool> &used, int depth,
int K,
                                         std::vector<std::vector<int>>
&result, int &call_count) {
    if (K < 1) {
        std::cout << "Ошибка: K < 1";
        exit(1);
    }
    if (K > A.size()) {
        std::cout << "Ошибка: элементов меньше, чем K";
        exit(1);
    }

    int current_call_count = ++call_count;
    std::cout << "Вызов #" << current_call_count << ": depth=" << depth << ",
current={ ";
    for (int i : current) std::cout << i << " ";
    std::cout << "}" << std::endl;

    // Базовый случай: прекращаем рекурсию
    if (depth == K + 1) {
        result.push_back(current);
        std::cout << "- Возврат из вызова #" << current_call_count <<
std::endl;
        return;
    }

    // Рекурсивный случай: продолжаем составление
    for (size_t i = 0; i < A.size(); ++i)
        if (!used[i]) {
            used[i] = true;
            current.push_back(A[i]);
            get_partial_permutations_with_debug(A, current, used, depth + 1,
K, result, call_count);
            current.pop_back();
            used[i] = false;
        }

    std::cout << "- Возврат из вызова #" << current_call_count << std::endl;
    return;
}

void write_to_file(const std::string &filename, const
std::vector<std::vector<int>> &permutations) {
    std::ofstream file(filename, std::ios::binary);
    if (!file) {
        std::cout << std::endl << "Ошибка открытия файла для записи";
```

```
        exit(1);
    }
    for (const auto &permutation : permutations)
        file.write(reinterpret_cast<const char *>(permutation.data()),
permutation.size() * sizeof(int));
    file.close();
}

std::vector<std::vector<int>> read_from_file(const std::string &filename, int
K) {
    std::ifstream file(filename, std::ios::binary);
    if (!file) {
        std::cout << std::endl << "Ошибка открытия файла для чтения";
        exit(1);
    }
    std::vector<std::vector<int>> permutations;
    std::vector<int> permutation(K);
    while (file.read(reinterpret_cast<char *>(permutation.data()), K *
sizeof(int)))
        permutations.push_back(permutation);
    file.close();
    return permutations;
}

void print_permutations(const std::vector<std::vector<int>> &permutations) {
    for (const auto &permutation : permutations) {
        std::cout << "{ ";
        for (int item : permutation) std::cout << item << " ";
        std::cout << "}" << std::endl;
    }
}

int main() {
    // Ввод массива
    std::string input;
    std::vector<int> A;
    std::cout << "Массив: ";
    std::getline(std::cin >> std::ws, input);
    std::stringstream ss(input);
    std::string word;
    while (ss >> word) A.push_back(std::stoi(word));

    int K;
    std::cout << "K: ";
    std::cin >> K;

    std::vector<std::vector<int>> permutations;
    std::vector<int> current;
    std::vector<bool> used(A.size(), false);
    int call_count = 0;

    get_partial_permutations_with_debug(A, current, used, 1, K, permutations,
call_count);

    // Запись размещений в файл
    std::cout << "Запись в файл permutations.bin ... ";
```

```
    std::string filename =
"/home/grigorijtomczuk/Desktop/suai/op/lab3.2/permutations.bin";
    write_to_file(filename, permutations);
    std::cout << " ✔" << std::endl;

    // Чтение и вывод размещений из файла
    std::cout << "Чтение из файла permutations.bin ... ";
    std::vector<std::vector<int>> readPermutations = read_from_file(filename,
K);
    std::cout << " ✔" << std::endl;
    print_permutations(readPermutations);

    // Глубина рекурсии для данной задачи всегда равна K+1
    std::cout << "Глубина: " << K + 1;

    return 0;
}
```

5 Результаты тестирования программы

```
Массив: 1 2 3
K: 2
Вызов #1: depth=1, current={ }
Вызов #2: depth=2, current={ 1 }
Вызов #3: depth=3, current={ 1 2 }
- Возврат из вызова #3
Вызов #4: depth=3, current={ 1 3 }
- Возврат из вызова #4
- Возврат из вызова #2
Вызов #5: depth=2, current={ 2 }
Вызов #6: depth=3, current={ 2 1 }
- Возврат из вызова #6
Вызов #7: depth=3, current={ 2 3 }
- Возврат из вызова #7
- Возврат из вызова #5
Вызов #8: depth=2, current={ 3 }
Вызов #9: depth=3, current={ 3 1 }
- Возврат из вызова #9
Вызов #10: depth=3, current={ 3 2 }
- Возврат из вызова #10
- Возврат из вызова #8
- Возврат из вызова #1
Запись в файл permutations.bin ... ✔
Чтение из файла permutations.bin ... ✔
{ 1 2 }
{ 1 3 }
{ 2 1 }
{ 2 3 }
{ 3 1 }
{ 3 2 }
Глубина: 3
Process finished with exit code 0
```

Рисунок 2

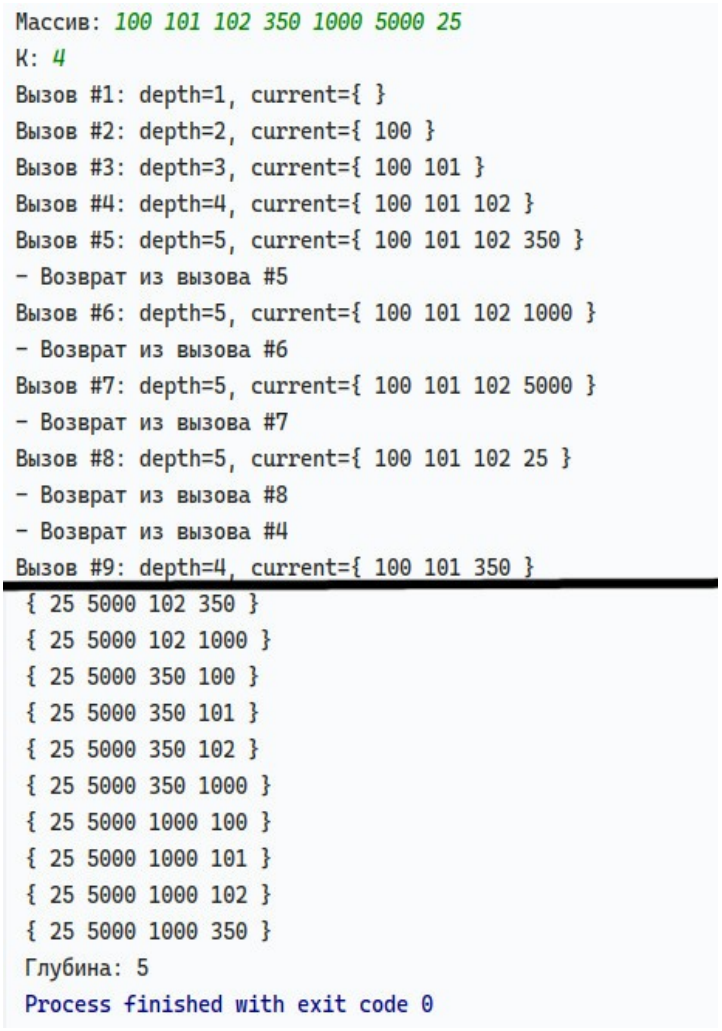

Рисунок 3

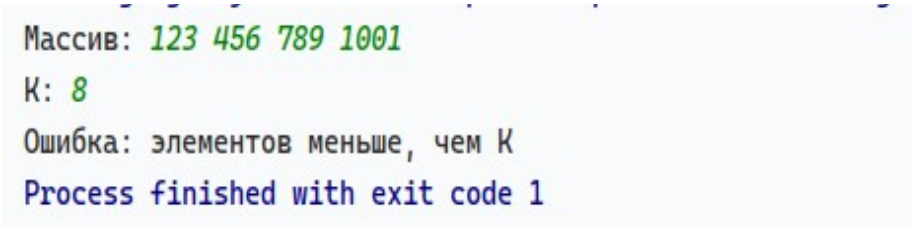

Рисунок 4

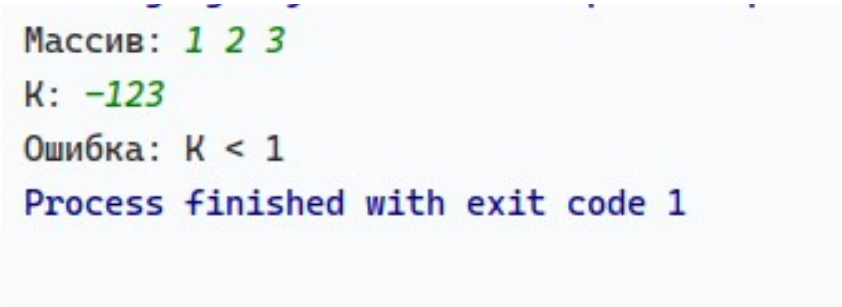

Рисунок 5

```
Массив: 1 2 3
К: 2
Вызов #1: depth=1, current={ }
Вызов #2: depth=2, current={ 1 }
Вызов #3: depth=3, current={ 1 2 }
- Возврат из вызова #3
Вызов #4: depth=3, current={ 1 3 }
Запись в файл permutations.bin ...
Ошибка открытия файла для записи
Process finished with exit code 1
```

Рисунок 6

```
Массив: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
К: 20
- Возврат из вызова #8698617
Вызов #8698623: depth=20, current={ 1 2 3 4 5 6 7 8 9 10 11 12 17 15 19 1
Вызов #8698624: depth=21, current={ 1 2 3 4 5 6 7 8 9 10 11 12 17 15 19 1
- Возврат из вызова #8698624
Вызов #8698625: depth=21, current={ 1 2 3 4 5 6 7 8 9 10 11 12 17 15 19 1
- Возврат из вызова #8698625
Вызов #8698626: depth=21, current={ 1 2 3 4 5 6 7 8 9 10 11 12 17 15 19 1
- Возврат из вызова #8698626
Вызов #8698627: depth=21, current={ 1 2 3 4 5 6 7 8 9 10 11 12 17 15 19 1
- Возврат из вызова #8698627
Вызов #8698628: depth=21, current={ 1 2 3 4 5 6 7 8 9 10 11 12 17 15 19 1
- Возврат из вызова #8698628
- Возврат из вызова #8698623
- Возврат из вызова #8698592
Вызов #8698629: depth=19, current={ 1 2 3 4 5 6 7 8 9 10 11 12 17 15 19 1
Вызов #8698630: depth=20, current={ 1 2 3 4 5 6 7 8 9 10 11 12 17 15 19 1
Process finished with exit code 15
```

Рисунок 7

6 Вывод по результату тестирования

Нормальные значения. Функция корректно вычисляет результат для нормальных значений (рис. 2, 3).

Отрицательные и нулевые значения. При значениях K < 1 и K больше размера A работа программы прекращается, так как составить размещения при таком значении невозможно (рис. 4, 5).

Ошибка открытия файла. При ошибке открытия файла работа программы прекращается, так как невозможно записать результат (рис. 6).

Большие значения. За счет того, что глубина рекурсии никогда не превышает K+1, большие значения и объемные вычисления не прерывают работу программы, однако вычисление результата занимает слишком много времени (рис. 7). Для того, чтобы программа оборвалась переполнением стека, необходимо подать на вход большое значение K, а следовательно и очень большой массив, чтобы глубина рекурсии достигла слишком большого значения.

```
Массив: 1 2 3
К: 2
Вызов #1: depth=1, current={ }
Вызов #2: depth=2, current={ 1 }
Вызов #3: depth=3, current={ 1 2 }
- Возврат из вызова #3
Вызов #4: depth=3, current={ 1 3 }
- Возврат из вызова #4
- Возврат из вызова #2
Вызов #5: depth=2, current={ 2 }
Вызов #6: depth=3, current={ 2 1 }
- Возврат из вызова #6
Вызов #7: depth=3, current={ 2 3 }
- Возврат из вызова #7
- Возврат из вызова #5
Вызов #8: depth=2, current={ 3 }
Вызов #9: depth=3, current={ 3 1 }
- Возврат из вызова #9
Вызов #10: depth=3, current={ 3 2 }
- Возврат из вызова #10
- Возврат из вызова #8
- Возврат из вызова #1
Запись в файл permutations.bin ... ✔
Чтение из файла permutations.bin ... ✔
{ 1 2 }
{ 1 3 }
{ 2 1 }
{ 2 3 }
{ 3 1 }
{ 3 2 }
Глубина: 3
Process finished with exit code 0
```

$$A_3^2 = \frac{3!}{(3-2)!} = 6$$

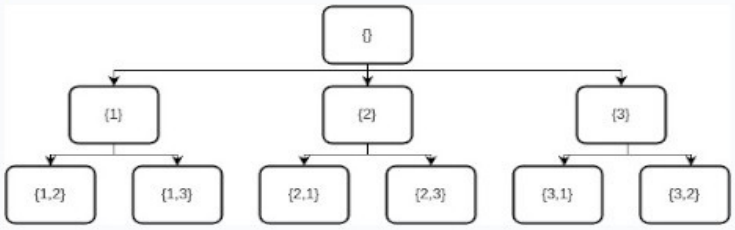

Рисунок 8

Из рис. 8 видно, что функция алгоритмически корректно решает задачу.