МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
Кафедра вычислительной техники

ОТЧЁТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ
ПО ДИСЦИПЛИНЕ «ИНФОРМАТИКА»

«Анализ программ, функции**»**

Факультет: АВТ
Группа: ДТ-460а
Студент: Пантюхин Артём Евгеньевич

Преподаватель:
Копылова Оксана Андреевна

Новосибирск, 2025 г.

Содержание

Задание

Из заданий 13-58 выдается 5 номеров фрагментов кода. Понять, что происходит в программе, что является результатом ее работы, прокомментировать код, указав все стандартные решения (стандартные решения см. в разделе 2.4 cprog).

Каждый фрагмент оформить в виде трех функций. В первой функции – передача параметров по значению, во второй – по ссылке, в третьей – по указателю. Рядом с функцией в комментариях указать, каким способом передаются переменные и для чего эти переменные нужны. Передать в функции массив. В main() каждую функцию вызвать по 2 раза с разными входными фактическими параметрами.

В main одновременный вывод всех результатов в виде “F13_val()” (по значению), “F13_ref() ” (по ссылке), “F13_pointer()”(по указателю) – номер функции_номер вызова – результат.

```
//-----------------------------------------------------20
for (i=1,k=0; i<n; i++)
        if (A[i]>A[k]) k=i;
for (j=k; j<n-1;j++) A[j]=A[j+1];
        n--;
//-----------------------------------------------------29
for (i=0; i<n; i++)
        if (A[i]%v!=0) { v--; i=-1; }
//-----------------------------------------------------32
for (s=0,i=0; i<n; i++){
        for (k=0,j=0; j<n; j++)
                if (c[i]==c[j]) k++;
        if (k>s) s=k,b=i;
        }
```

```
//-----------------------------------------------------50
for (i=0,s=0,k=0; i<10; i++)
    if (A[i]<0) k=1;
    else {
          if (k==1) s++;
          k=0;
          }
//-----------------------------------------------------56
for (i=1,s=0; i<10; i++)
    if (A[i]>0 && A[i-1]<0) s++;
```

Теоретические сведения

Указатель (pointer) - переменная, предназначенная для хранения адреса любого другого значения в памяти. Указатель можно разыменовывать, получая значение по данному адресу. Тип указателя известен во время компиляции и используется для ограничения битности значения при разыменовании (конвертации области памяти, объёмом, соответствующим типу указателя, в переменную того же типа), а так же при адресном сдвиге. Адрес, как правило, представляет собой целое беззнаковое число. Диапазон адресов зависит от архитектуры проекта при сборке.

Ссылка (reference) — синтаксический сахар, предоставляющий функционал указателя, разыменованного во время компиляции. Ссылка может быть использована как для указания альтернативного имени переменной, так и для передачи переменной в функцию не по значению, а по адресу в памяти (все изменения переменной, переданной по ссылке, сохранятся по выходу из функции). Ссылка должна быть инициализирована в момент объявления. Ссылка не может изменить значение (переменную, на которую ссылается) после инициализации. Ссылка может быть инициализирована другой ссылкой, ссылаясь на ту же переменную, что и первая.

1. Передача по значению

При передаче параметра по значению создается копия исходной переменной, которую функция получает. Любые изменения этой копии внутри функции никак не влияют на оригинальную переменную вне функции.

При передаче массива (не структуры), в случае C и C++, будет передан указатель на его первый элемент.

2. Передача по ссылке

Передача по ссылке позволяет функции получить доступ непосредственно к оригиналу переменной, передавая её адрес. Таким образом, любые изменения параметра внутри функции будут отражены и снаружи функции.

3. Передача по указателю

Указатели позволяют передавать адрес переменной в функцию. Далее указатель может быть как разыменован для получения доступа к значению переменной (изменения данного значения сохранятся при выходе из функции), так и использован в целях итерации по элементам массива, если память под значения массива выделяется в соседних блоках (небезопасно, в случае неосторожного обращения и/или несоблюдения границ известных значений приводит к ошибке сегментации).

Программа №20

i, j – счётчики.
k – индекс элемента с максимальным значением.
A – входной массив.
n – длина входного массива.
out – значение элемента с индексом k (до сдвига, в конце программы - удалённое из входного массива)

Исходный код с комментариями: см. приложение 1.1.
Код вызова функций в main(): см. приложение 2.1.
Пример работы программы: см. приложение 3.1.

Программа №29

i, v – счётчики.
A – входной массив.
n – длина входного массива.

Исходный код с комментариями: см. приложение 1.2.
Код вызова функций в main(): см. приложение 2.2.
Пример работы программы: см. приложение 3.2.

Программа №32

i, j, k, s – счётчики.
b – индекс элемента с наибольшим количеством вхождений.
c – входной массив.
n – длина входного массива.

Исходный код с комментариями: см. приложение 1.3.
Код вызова функций в main(): см. приложение 2.3.
Пример работы программы: см. приложение 3.3.

Программа №50

i, s – счётчики.
k – флаговая переменная.
A – входной массив.

Исходный код с комментариями: см. приложение 1.4.
Код вызова функций в main(): см. приложение 2.4.
Пример работы программы: см. приложение 3.4.

Ошибки и неточности:

Учитываются только чередования с отрицательного на положительное значение, но не наоборот. Учитываются только первые 10 элементов.

В случае, если во входном массиве будет менее 10 элементов, произойдёт ошибка сегментации. Программа завершит работу с кодом 11.

Программа №56

i,s – счётчики.
A – входной массив.

Исходный код с комментариями: см. приложение 1.5.
Код вызова функций в main(): см. приложение 2.5.
Пример работы программы: см. приложение 3.5.

Ошибки и неточности:

Учитываются только чередования с отрицательного на положительное значение, но не наоборот. Учитываются только первые 10 элементов.

В случае, если во входном массиве будет менее 10 элементов, произойдёт ошибка сегментации. Программа завершит работу с кодом 11.

Выводы

В ходе выполнения лабораторной работы был проведён анализ работы пяти функций, а так же исследованы особенности различных способов передачи переменных функцию:

По значению:

Пожалуй, наиболее безопасный метод передачи, изменения внутри функции никак не затрагивают переменные вне её.

По ссылке:

Удобное сокращение передачи по указателю, в рамках данной лабораторной работы функциональной разницы с указателем нет.

По указателю:

Преимущества указателя над ссылкой не были реализованы в рамках данной лабораторной работы. Функции для работы с указателями имеют небольшие отличия в синтаксисе в виду необходимости разыменования. Самый небезопасный метод передачи, позволяющий напрямую менять значения переменной по хранимому адресу. Изменения сохраняются по выходу из функции.

В ходе выполнения лабораторной работы были реализованы функции, выполняющие задачи:

1) Удаления наибольшего элемента из входного массива.

2) Нахождения наибольшего общего делителя для элементов входного массива.

3) Нахождения элемента, наиболее часто встречающегося в массиве.

4) Подсчёта количества чередований отрицательных и положительных элементов во входном массиве в двух реализациях.

Наиболее комплексной частью проекта оказался форматированный вывод результатов тестирования работы функций.

Список литературы

1. Подбельский В.В., Фомин С.С. Программирование на языке Си: Учеб.пособие. – 2-е доп. Изд. – М.: Финансы и статистика, 2004. – 600 с.

2. Романов Е. Л. Си/Си++. От дилетанта до профессионала. Электронное учебное пособие по дисциплинам "Информатика", "Программирование", "Технология программирования" для студентов 1–2 курсов направления 230100 : учеб. пособие / Е. Л. Романов. – Новосибирский государственный технический университет, № гос регистрации 0321000528, 2010. - 581 с.

3. Си/Си++ от дилетанта до профессионала [Электронный ресурс]. URL: http://ermak.cs.nstu.ru/cprog/HTML/index.htm (дата обращения 01.10.2022).

Приложение 1. Исходный код программы с комментариями.

[Ссылка на репозиторий с исходным кодом и Makefile на GitHub]

main.cpp:

```
#include <clocale>
#include <vector>

#include "../core/output.hpp"

int main(void) {
  // Создаём два эталонных вектора и их длину для использования в функциях.
  const std::vector<int> rval_1 = {447, 691, 126, -836, 748,
                                   767, 301, -37, 345,  -164};
  const std::vector<int> rval_2 = {-420, 2,   -160, 536, -573, -429, -233,
                                   905,  558, -772, 445, 862,  -312, 2};
  const int rlen_1 = 10;
  const int rlen_2 = 14;
  // Устанавливаем локаль для корректного вывода мультибайтовой кириллицы.
  setlocale(LC_ALL, "Russian");
  // Форматированный вывод для каждой из наших функций
  S2L1::Output::print_f20(rval_1, rval_2, rlen_1, rlen_2);
  S2L1::Output::print_f29(rval_1, rval_2, rlen_1, rlen_2);
  S2L1::Output::print_f32(rval_1, rval_2, rlen_1, rlen_2);
  S2L1::Output::print_f50(rval_1, rval_2, rlen_1, rlen_2);
  S2L1::Output::print_f56(rval_1, rval_2, rlen_1, rlen_2);
  return 0;
}
```

output.hpp:

```
#ifndef S2L1_OUTPUT_H
#define S2L1_OUTPUT_H
#include <clocale>
#include <cstdio>
#include <cstring>
#include <vector>
#include "functions.hpp"
namespace S2L1::Output {
//Функция для форматированного вывода вектора в поток stdout;
void print_vector(std::vector<int> val, int length);
//Функция для проверки работы функции F20_*, и форматированного вывода результатов;
void print_f20(std::vector<int> rval_1, std::vector<int> rval_2, int rlen_1,
        int rlen_2);
//Функция для проверки работы функции F29_*, и форматированного вывода результатов;
void print_f29(std::vector<int> rval_1, std::vector<int> rval_2, int rlen_1,
        int rlen_2);
//Функция для проверки работы функции F32_*, и форматированного вывода результатов;
void print_f32(std::vector<int> rval_1, std::vector<int> rval_2, int rlen_1,
        int rlen_2);
//Функция для проверки работы функции F50_*, и форматированного вывода результатов;
void print_f50(std::vector<int> rval_1, std::vector<int> rval_2, int rlen_1,
        int rlen_2);
//Функция для проверки работы функции F56_*, и форматированного вывода результатов;
void print_f56(std::vector<int> rval_1, std::vector<int> rval_2, int rlen_1,
        int rlen_2);
}
#endif
```

output.cpp:

```
#include "output.hpp"
namespace S2L1::Output {
#define NF_BREAK "----------------------------------------"

void print_vector(std::vector<int> val, int length) {
  printf("{");
  for (int i = 0; i < length; i++) {
    if (i != 0) {
      printf(", ");
    }
    //В рамках данных тестов, все входные значения по модулю меньше 1000.
    printf("%4d", val[i]);
  }
  printf("}\n");
}

void print_function_preexec(const char* name, const int iteration,
                            std::vector<int> array, int length) {
  //Устанавливаем локаль, если вдруг была другая.
  if (!strcmp(setlocale(LC_ALL, NULL), "Russian")) {
    setlocale(LC_ALL, "Russian");
  }
  char out_str[255] = "";
  //Выводим строку сначала в буфер, чтобы иметь возможность указать длину при выводе
в stdout.
  snprintf(out_str, 254, "%s-%d-%s", name, iteration, "входной массив");
  printf("%-43s: ", out_str);
  print_vector(array, length);
}

void print_function_postexec(const char* name, const int iteration,
                             std::vector<int> array, int length, int result) {
  if (!strcmp(setlocale(LC_ALL, NULL), "Russian")) {
    setlocale(LC_ALL, "Russian");
  }
  char out_str[255] = "";
```

```
  //Выводим строку сначала в буфер, чтобы иметь возможность указать длину при выводе
в stdout.
  sprintf(out_str, "%s-%d-%s", name, iteration, "итоговый массив");
  printf("%-44s: ", out_str);
  print_vector(array, length);
  //Выводим строку сначала в буфер, чтобы иметь возможность указать длину при выводе
в stdout.
  sprintf(out_str, "%s-%d-%s", name, iteration, "результат");
  printf("%-39s: %d\n", out_str, result);
}

void print_f20(std::vector<int> rval_1, std::vector<int> rval_2, int rlen_1,
               int rlen_2) {
  /*
  Здесь и далее:
  test_vector, n - копии эталонных значений для отслеживания изменений
  res - переменная под вывод функций
  */
  std::vector<int> test_vector = {};
  int n = 0;
  int res = 0;

  //Выводим ---------- для красоты
  printf("%s\n", NF_BREAK);

  /*
  Здесь и далее: логика вывода:
  выводим название функции, номер набора входных значений, входные значения.
  получаем результат работы функции в res.
  выводим название функции, номер набора входных значений, выходные значения.
  выводим название функции, номер набора входных значений, результат работы.
  */
  test_vector = rval_1;
  n = rlen_1;
  print_function_preexec("F20_val", 1, test_vector, n);
  res = S2L1::Functions::F20_val(test_vector, n);
  print_function_postexec("F20_val", 1, test_vector, n, res);
```

```
  test_vector = rval_1;
  n = rlen_1;
  print_function_preexec("F20_ref", 1, test_vector, n);
  res = S2L1::Functions::F20_ref(test_vector, n);
  print_function_postexec("F20_ref", 1, test_vector, n, res);

  test_vector = rval_1;
  n = rlen_1;
  print_function_preexec("F20_pointer", 1, test_vector, n);
  res = S2L1::Functions::F20_pointer(&test_vector, &n);
  print_function_postexec("F20_pointer", 1, test_vector, n, res);

  printf("\n");

  test_vector = rval_2;
  n = rlen_2;
  print_function_preexec("F20_val", 2, test_vector, n);
  res = S2L1::Functions::F20_val(test_vector, n);
  print_function_postexec("F20_val", 2, test_vector, n, res);

  test_vector = rval_2;
  n = rlen_2;
  print_function_preexec("F20_ref", 2, test_vector, n);
  res = S2L1::Functions::F20_ref(test_vector, n);
  print_function_postexec("F20_ref", 2, test_vector, n, res);

  test_vector = rval_2;
  n = rlen_2;
  print_function_preexec("F20_pointer", 2, test_vector, n);
  res = S2L1::Functions::F20_pointer(&test_vector, &n);
  print_function_postexec("F20_pointer", 2, test_vector, n, res);

  printf("%s\n", NF_BREAK);
}
```

```
void print_f29(std::vector<int> rval_1, std::vector<int> rval_2, int rlen_1,
               int rlen_2) {
  std::vector<int> test_vector = {};
  int n = 0;
  int res = 0;

  printf("%s\n", NF_BREAK);

  test_vector = rval_1;
  n = rlen_1;
  print_function_preexec("F29_val", 1, test_vector, n);
  res = S2L1::Functions::F29_val(test_vector, n);
  print_function_postexec("F29_val", 1, test_vector, n, res);

  test_vector = rval_1;
  n = rlen_1;
  print_function_preexec("F29_ref", 1, test_vector, n);
  res = S2L1::Functions::F29_ref(test_vector, n);
  print_function_postexec("F29_ref", 1, test_vector, n, res);

  test_vector = rval_1;
  n = rlen_1;
  print_function_preexec("F29_pointer", 1, test_vector, n);
  res = S2L1::Functions::F29_pointer(&test_vector, &n);
  print_function_postexec("F29_pointer", 1, test_vector, n, res);

  printf("\n");

  test_vector = rval_2;
  n = rlen_2;
  print_function_preexec("F29_val", 2, test_vector, n);
  res = S2L1::Functions::F29_val(test_vector, n);
  print_function_postexec("F29_val", 2, test_vector, n, res);

  test_vector = rval_2;
  n = rlen_2;
```

```
  print_function_preexec("F29_ref", 2, test_vector, n);
  res = S2L1::Functions::F29_ref(test_vector, n);
  print_function_postexec("F29_ref", 2, test_vector, n, res);

  test_vector = rval_2;
  n = rlen_2;
  print_function_preexec("F29_pointer", 2, test_vector, n);
  res = S2L1::Functions::F29_pointer(&test_vector, &n);
  print_function_postexec("F29_pointer", 2, test_vector, n, res);

  printf("%s\n", NF_BREAK);
}

void print_f32(std::vector<int> rval_1, std::vector<int> rval_2, int rlen_1,
               int rlen_2) {
  std::vector<int> test_vector = {};
  int n = 0, res = 0;

  printf("%s\n", NF_BREAK);

  test_vector = rval_1;
  n = rlen_1;
  print_function_preexec("F32_val", 1, test_vector, n);
  res = S2L1::Functions::F32_val(test_vector, n);
  print_function_postexec("F32_val", 1, test_vector, n, res);

  test_vector = rval_1;
  n = rlen_1;
  print_function_preexec("F32_ref", 1, test_vector, n);
  res = S2L1::Functions::F32_ref(test_vector, n);
  print_function_postexec("F32_ref", 1, test_vector, n, res);

  test_vector = rval_1;
  n = rlen_1;
  print_function_preexec("F32_pointer", 1, test_vector, n);
  res = S2L1::Functions::F32_pointer(&test_vector, &n);
```

```
  print_function_postexec("F32_pointer", 1, test_vector, n, res);

  printf("\n");

  test_vector = rval_2;
  n = rlen_2;
  print_function_preexec("F32_val", 2, test_vector, n);
  res = S2L1::Functions::F32_val(test_vector, n);
  print_function_postexec("F32_val", 2, test_vector, n, res);

  test_vector = rval_2;
  n = rlen_2;
  print_function_preexec("F32_ref", 2, test_vector, n);
  res = S2L1::Functions::F32_ref(test_vector, n);
  print_function_postexec("F32_ref", 2, test_vector, n, res);

  test_vector = rval_2;
  n = rlen_2;
  print_function_preexec("F32_pointer", 2, test_vector, n);
  res = S2L1::Functions::F32_pointer(&test_vector, &n);
  print_function_postexec("F32_pointer", 2, test_vector, n, res);

  printf("%s\n", NF_BREAK);
}

void print_f50(std::vector<int> rval_1, std::vector<int> rval_2, int rlen_1, int rlen_2) {
  std::vector<int> test_vector = {};
  int n = 0;
  int res = 0;

  printf("%s\n", NF_BREAK);

  test_vector = rval_1;
  n = rlen_1;
  print_function_preexec("F50_val", 1, test_vector, n);
  res = S2L1::Functions::F50_val(test_vector);
```

```
  print_function_postexec("F50_val", 1, test_vector, n, res);

  test_vector = rval_1;
  n = rlen_1;
  print_function_preexec("F50_ref", 1, test_vector, n);
  res = S2L1::Functions::F50_ref(test_vector);
  print_function_postexec("F50_ref", 1, test_vector, n, res);

  test_vector = rval_1;
  n = rlen_1;
  print_function_preexec("F50_pointer", 1, test_vector, n);
  res = S2L1::Functions::F50_pointer(&test_vector);
  print_function_postexec("F50_pointer", 1, test_vector, n, res);

  printf("\n");

  test_vector = rval_2;
  n = rlen_2;
  print_function_preexec("F50_val", 2, test_vector, n);
  res = S2L1::Functions::F50_val(test_vector);
  print_function_postexec("F50_val", 2, test_vector, n, res);

  test_vector = rval_2;
  n = rlen_2;
  print_function_preexec("F50_ref", 2, test_vector, n);
  res = S2L1::Functions::F50_ref(test_vector);
  print_function_postexec("F50_ref", 2, test_vector, n, res);

  test_vector = rval_2;
  n = rlen_2;
  print_function_preexec("F50_pointer", 2, test_vector, n);
  res = S2L1::Functions::F50_pointer(&test_vector);
  print_function_postexec("F50_pointer", 2, test_vector, n, res);

  printf("%s\n", NF_BREAK);
}
```

```
void print_f56(std::vector<int> rval_1, std::vector<int> rval_2, int rlen_1, int rlen_2) {
  std::vector<int> test_vector = {};
  int n = 0, res = 0;

  printf("%s\n", NF_BREAK);

  test_vector = rval_1;
  n = rlen_1;
  print_function_preexec("F56_val", 1, test_vector, n);
  res = S2L1::Functions::F56_val(test_vector);
  print_function_postexec("F56_val", 1, test_vector, n, res);

  test_vector = rval_1;
  n = rlen_1;
  print_function_preexec("F56_ref", 1, test_vector, n);
  res = S2L1::Functions::F56_ref(test_vector);
  print_function_postexec("F56_ref", 1, test_vector, n, res);

  test_vector = rval_1;
  n = rlen_1;
  print_function_preexec("F56_pointer", 1, test_vector, n);
  res = S2L1::Functions::F56_pointer(&test_vector);
  print_function_postexec("F56_pointer", 1, test_vector, n, res);

  printf("\n");

  test_vector = rval_2;
  n = rlen_2;
  print_function_preexec("F56_val", 2, test_vector, n);
  res = S2L1::Functions::F56_val(test_vector);
  print_function_postexec("F56_val", 2, test_vector, n, res);

  test_vector = rval_2;
  n = rlen_2;
  print_function_preexec("F56_ref", 2, test_vector, n);
```

```
  res = S2L1::Functions::F56_ref(test_vector);
  print_function_postexec("F56_ref", 2, test_vector, n, res);

  test_vector = rval_2;
  n = rlen_2;
  print_function_preexec("F56_pointer", 2, test_vector, n);
  res = S2L1::Functions::F56_pointer(&test_vector);
  print_function_postexec("F56_pointer", 2, test_vector, n, res);

  printf("%s\n", NF_BREAK);
}
}
```

functions.hpp:

```
#ifndef S2L1_FUNCTIONS_H
#define S2L1_FUNCTIONS_H

#include <vector>

namespace S2L1::Functions {
/*
Здесь и далее:
То, что мы привыкли понимать под "массивом" - в сущности, лишь указатель на его первый элемент. Для наглядности разницы в передаче по значению и по ссылке/указателю будет использоваться std::vector - последовательный контейнер, обладающий в наших условиях схожим синтаксисом, но являющийся структурой данных, вследствие чего мы можем передать его значение, а не адрес в памяти.

В дальнейших комментариях к задаче под "массивом" будет подразумеваться
любая структура элементов одного типа, с последовательным доступом к ним.
*/

// Вспомогательная функция для поиска наибольшего элемента массива.
// Используется в функции 29 при инициализации v;
int find_max(std::vector<int> A, int n);

/*
Функция 20:
Функция однократно убирает наибольший элемент массива.
В случае, если таких элементов несколько, убран будет лишь элемент с наибольшим индексом.

Стандартные решения:
    Переменная - максимум.
    Поиск - полный перебор
    Индексы как независимые перемещения в цикле.
    Предыдущий, текущий, следующий (в рамках следующего решения)
    Удаление с сохранением порядка
```

```
*/
int F20_val(std::vector<int> A, int n);
int F20_ref(std::vector<int>& A, int& n);
int F20_pointer(std::vector<int>* A, int* n);

/*
Функция 29:
Функция предназначена для нахождения наибольшего общего делителя для
элементов массива. Предположим, что будет логично инициализировать v значением
наибольшего элемента массива.

Стандартные решения:
   Переменная - максимум(ведь ищем максимальное значение) - счётчик(ведь
уменьшаем его на единицу с каждой реитерацией)
   Свойство всеобщности (алгоритмы из
cprog 2.4 модифицированы под реитерацию)
   Неравномерное движение (сразу прыгаем в
начало если первый найденный не соответствует условию)
*/
int F29_val(std::vector<int> A, int n);
int F29_ref(std::vector<int>& A, int& n);
int F29_pointer(std::vector<int>* A, int* n);

/*
Функция 32:
Функция предназначена для нахождения индекса элемента с наибольшим
количеством вхождений в массив И/или количества вхождений этого элемента.

Стандартные решения:
   Переменная - счётчик
   Переменная - максимум
   Вложенные циклы
*/
int F32_val(std::vector<int> c, int n);
int F32_ref(std::vector<int>& c, int& n);
int F32_pointer(std::vector<int>* c, int* n);
```

```
/*
Функция 50:
Функция предназначена для подсчёта чередований -,+ в рамках первых 10 элементов массива. В случае, если массив состоит менее, чем из 10 элементов, функция вызовет исключение с кодом ошибки 11 (SEGFAULT). Функционально - абсолютно идентична F56.

Стандартные решения:
   Переменная - счётчик
   Переменная - признак
   Предыдущий, текущий, следующий (в варианте с флаговой переменной, но логика в целом та же)
*/
int F50_val(std::vector<int> A);
int F50_ref(std::vector<int>& A);
int F50_pointer(std::vector<int>* A);

/*
Функция 56:
Функция предназначена для подсчёта чередований -,+ в рамках первых 10 элементов массива. В случае, если массив состоит менее, чем из 10 элементов, функция вызовет исключение с кодом ошибки 11 (SEGFAULT). Функционально - абсолютно идентична F50.

Стандартные решения:
   Переменная - счётчик
   Предыдущий, текущий, следующий
*/
int F56_val(std::vector<int> A);
int F56_ref(std::vector<int>& A);
int F56_pointer(std::vector<int>* A);
}
#endif
```

functions.cpp:

```
#include "functions.hpp"

namespace S2L1::Functions {
int find_max(std::vector<int> A, int n) {
  int out = 0x80000000;  // == -2147483648, минимально возможный integer
//Инициализировали i, проходим весь массив, меняя значение out если встречаем элемент со значением больше.
  for (int i = 0; i < n; i++) {
    if (A[i] > out) {
      out = A[i];
    }
  }
  return out;
}

int F20_val(std::vector<int> A, int n) {
  //Инициализруем переменные-счётчики i,k,j;
  int i = 0, k = 0, j = 0;
  /*
  Движемся по массиву A от 0 до n-1 включительно, сравнивая между собой i-й (каждый раз новый) и k-й (эталонный) элементы
массива.
  */
  for (i = 1, k = 0; i < n; i++)
    //По результатам сравнения в переменной k получаем индекс первого наибольшего элемента массива A.
    if (A[i] > A[k]) k = i;
  // Инициализируем out значением k-го элемента сейчас, ведь далее он будет удалён.
  int out = A[k];
  //Наибольший (k-й) элемент убирается из массива, а все элементы правее него - однократно смещаются влево.
  for (j = k; j < n - 1; j++) A[j] = A[j + 1];
  /*
  Уменьшаем длину на 1, поскольку один элемент был убран.
  (Впрочем, память-то под него никуда не делась...)
```

```
  */
  n--;
  return out;
}

int F20_ref(std::vector<int>& A, int& n) {
  int i = 0, k = 0, j = 0;
  for (i = 1, k = 0; i < n; i++)
    if (A[i] > A[k]) k = i;
  int out = A[k];
  for (j = k; j < n - 1; j++) A[j] = A[j + 1];
  n--;
  return out;
}

//Для работы со значением по адресу, нам необходимо разименовать указатель (ранее A[i], теперь (*A)[i]).
int F20_pointer(std::vector<int>* A, int* n) {
  int i = 0, k = 0, j = 0;
  for (i = 1, k = 0; i < *n; i++)
    if ((*A)[i] > (*A)[k]) k = i;
  int out = (*A)[k];
  for (j = k; j < (*n) - 1; j++) (*A)[j] = (*A)[j + 1];
  (*n)--;
  return out;
}

int F29_val(std::vector<int> A, int n) {
  //Инициализируем v максимальным значением среди элементов массива A.
  int v = find_max(A, n);
  int i = 0;
  //Инициализируем i, далее движемся по массиву A от 0 до n-1 включительно
  for (i = 0; i < n; i++)
    /*
    В случае, если i-й элемент массива не делится нацело на v, уменьшаем v на 1 и
    начинаем заново
    */
```

```
    if (A[i] % v != 0) {
      v--;
      /*
      i = -1; ->
      после выполнится i++ из шапки цикла ->
      i==0 на следующей итерации.
      */
      i = -1;
    }
  return v;
}

int F29_ref(std::vector<int>& A, int& n) {
  int v = find_max(A, n);
  int i = 0;
  for (i = 0; i < n; i++)
    if (A[i] % v != 0) {
      v--;
      i = -1;
    }
  return v;
}

//Для работы со значением по адресу, нам необходимо разименовать указатель (ранее
A[i], теперь (*A)[i]).
int F29_pointer(std::vector<int>* A, int* n) {
  int v = find_max(*A, *n);
  int i = 0;
  for (i = 0; i < *n; i++)
    if ((*A)[i] % v != 0) {
      v--;
      i = -1;
    }
  return v;
}

int F32_val(std::vector<int> c, int n) {
```

```
  int s, i, k, j, b;
  //Инициализируем i, s, далее движемся по массиву c от 0 до n-1;
  for (s = 0, i = 0; i < n; i++) {
    /*
    Для каждого i инициализируем k,j, после чего заново
    проходим весь массив, сравнивая i-й и j-й элементы.
    */
    for (k = 0, j = 0; j < n; j++)
      if (c[i] == c[j]) k++;
    /*
    После прохождения, получаем k == количество вхождений i-го элемента в массив c
    Если данное количество больше эталонного (начиная с s == 0),
    то количество вхождений i-го элемента становится эталонным (s = k),
    а индекс элемента сохраняется в b.
    */
    if (k > s) s = k, b = i;
  }
  return b;
}

int F32_ref(std::vector<int>& c, int& n) {
  int s, i, k, j, b;
  for (s = 0, i = 0; i < n; i++) {
    for (k = 0, j = 0; j < n; j++)
      if (c[i] == c[j]) k++;
    if (k > s) s = k, b = i;
  }
  return b;
}

//Для работы со значением по адресу, нам необходимо разименовать указатель (ранее
A[i], теперь (*A)[i]).
int F32_pointer(std::vector<int>* c, int* n) {
  int s, i, k, j, b;
  for (s = 0, i = 0; i < (*n); i++) {
    for (k = 0, j = 0; j < (*n); j++)
      if ((*c)[i] == (*c)[j]) k++;
```

```
    if (k > s) s = k, b = i;
  }
  return b;
}


int F50_val(std::vector<int> A) {
  int i, s, k;
  //Инициализируем i,s,k, далее движемся по массиву A от 0 до 9 включительно.
  for (i = 0, s = 0, k = 0; i < 10; i++) {
    /*
    Скажем, что k в рамках итерации - знак прошлого элемента массива.
    k = 0 если элемент был положительным,
    k = 1 в противном случае.
    */
    if (A[i] < 0)
      k = 1;
    else {
      /*
      Если прошлый элемент был отрицательным, а текущий - положителен,
      то увеличиваем счётчик (s) на 1.
      */
      if (k == 1) s++;
      k = 0;
    }
  }
  return s;
}

int F50_ref(std::vector<int>& A) {
  int i, s, k;
  for (i = 0, s = 0, k = 0; i < 10; i++) {
    if (A[i] < 0)
      k = 1;
    else {
      if (k == 1) s++;
      k = 0;
```

```
    }
  }
  return s;
}

//Для работы со значением по адресу, нам необходимо разименовать указатель (ранее
A[i], теперь (*A)[i]).
int F50_pointer(std::vector<int>* A) {
  int i, s, k;
  for (i = 0, s = 0, k = 0; i < 10; i++) {
    if ((*A)[i] < 0)
      k = 1;
    else {
      if (k == 1) s++;
      k = 0;
    }
  }
  return s;
}

int F56_val(std::vector<int> A) {
  //Инициализируем i,s, движемся по массиву A от 1 до 9 включительно.
  int i, s = 0;
  for (i = 1, s = 0; i < 10; i++)
    /*
    Если среди элементов встречено чередование -,+ (как и в прошлой функции),
    увеличиваем счётчик на 1.
    */
    if (A[i] > 0 && A[i - 1] < 0) s++;
  return s;
}

int F56_ref(std::vector<int>& A) {
  int i, s = 0;
  for (i = 1, s = 0; i < 10; i++)
    if (A[i] > 0 && A[i - 1] < 0) s++;
  return s;
```

```
}

//Для работы со значением по адресу, нам необходимо разыменовать указатель (ранее A[i], теперь (*A)[i]).
int F56_pointer(std::vector<int>* A) {
  int i, s = 0;
  for (i = 1, s = 0; i < 10; i++)
    if ((*A)[i] > 0 && (*A)[i - 1] < 0) s++;
  return s;
}
}
```

Приложение 1.1. Исходный код программы №20.

```
/*
Функция 20:
Функция однократно убирает наибольший элемент массива.
В случае, если таких элементов несколько, убран будет лишь элемент с
наибольшим
индексом.

Стандартные решения:
    Переменная - максимум.
    Поиск - полный перебор
    Индексы как независимые перемещения в цикле.
    Предыдущий, текущий, следующий (в рамках следующего решения)
    Удаление с сохранением порядка
*/
int F20_val(std::vector<int> A, int n);
int F20_ref(std::vector<int>& A, int& n);
int F20_pointer(std::vector<int>* A, int* n);

int F20_val(std::vector<int> A, int n) {
  //Инициализируем переменные-счётчики i,k,j;
  int i = 0, k = 0, j = 0;
  /*
  Движемся по массиву A от 0 до n-1 включительно, сравнивая между собой i-й
(каждый раз новый) и k-й (эталонный) элементы
массива.
  */
  for (i = 1, k = 0; i < n; i++)
    //По результатам сравнения в переменной k получаем индекс первого
наибольшего элемента массива A.
    if (A[i] > A[k]) k = i;
  // Инициализируем out значением k-го элемента сейчас, ведь далее он будет
удалён.
  int out = A[k];
  //Наибольший (k-й) элемент убирается из массива, а все элементы правее
него - однократно смещаются влево.
  for (j = k; j < n - 1; j++) A[j] = A[j + 1];
  /*
  Уменьшаем длину на 1, поскольку один элемент был убран.
  (Впрочем, память-то под него никуда не делась...)
  */
  n--;
  return out;
```

```
}

int F20_ref(std::vector<int>& A, int& n) {
  int i = 0, k = 0, j = 0;
  for (i = 1, k = 0; i < n; i++)
    if (A[i] > A[k]) k = i;
  int out = A[k];
  for (j = k; j < n - 1; j++) A[j] = A[j + 1];
  n--;
  return out;
}

//Для работы со значением по адресу, нам необходимо разименовать указатель
(ранее A[i], теперь (*A)[i]).
int F20_pointer(std::vector<int>* A, int* n) {
  int i = 0, k = 0, j = 0;
  for (i = 1, k = 0; i < *n; i++)
    if ((*A)[i] > (*A)[k]) k = i;
  int out = (*A)[k];
  for (j = k; j < (*n) - 1; j++) (*A)[j] = (*A)[j + 1];
  (*n)--;
  return out;
}
```

Приложение 1.2. Исходный код программы №29.

```
/*

Функция 29:
Функция предназначена для нахождения наибольшего общего делителя для
элементов массива. Предположим, что будет логично инициализировать v
значением
наибольшего элемента массива.

Стандартные решения:
    Переменная - максимум(ведь ищем максимальное значение) - счётчик(ведь
уменьшаем его на единицу с каждой реитерацией)
    Свойство всеобщности (алгоритмы из
cprog 2.4 модифицированы под реитерацию)
    Неравномерное движение (сразу прыгаем в
начало если первый найденный не соответствует условию)
*/
int F29_val(std::vector<int> A, int n);
int F29_ref(std::vector<int>& A, int& n);
int F29_pointer(std::vector<int>* A, int* n);
int F29_val(std::vector<int> A, int n) {
  //Инициализируем v максимальным значением среди элементов массива A.
  int v = find_max(A, n);
  int i = 0;
  //Инициализируем i, далее движемся по массиву A от 0 до n-1 включительно
  for (i = 0; i < n; i++)
    /*
    В случае, если i-й элемент массива не делится нацело на v, уменьшаем v
на 1 и
    начинаем заново
    */
    if (A[i] % v =/= 0) {
      v--;
      /*
      i = -1; ->
      после выполнится i++ из шапки цикла ->
      i==0 на следующей итерации.
      */
      i = -1;
    }
  return v;
}
```

```
int F29_ref(std::vector<int>& A, int& n) {
  int v = find_max(A, n);
  int i = 0;
  for (i = 0; i < n; i++)
    if (A[i] % v != 0) {
      v--;
      i = -1;
    }
  return v;
}

//Для работы со значением по адресу, нам необходимо разименовать указатель
(ранее A[i], теперь (*A)[i]).
int F29_pointer(std::vector<int>* A, int* n) {
  int v = find_max(*A, *n);
  int i = 0;
  for (i = 0; i < *n; i++)
    if ((*A)[i] % v != 0) {
      v--;
      i = -1;
    }
  return v;
}
```

Приложение 1.3. Исходный код программы №32

```
/*
Функция 32:
Функция предназначена для нахождения индекса элемента с наибольшим
количеством вхождений в массив И/или количества вхождений этого элемента.

Стандартные решения:
    Переменная - счётчик
    Переменная - максимум
    Вложенные циклы
*/
int F32_val(std::vector<int> c, int n);
int F32_ref(std::vector<int>& c, int& n);
int F32_pointer(std::vector<int>* c, int* n);
int F32_val(std::vector<int> c, int n) {
  int s, i, k, j, b;
  //Инициализируем i, s, далее движемся по массиву c от 0 до n-1;
  for (s = 0, i = 0; i < n; i++) {
    /*
    Для каждого i инициализируем k,j, после чего заново
    проходим весь массив, сравнивая i-й и j-й элементы.
    */
    for (k = 0, j = 0; j < n; j++)
      if (c[i] == c[j]) k++;
    /*
    После прохождения, получаем k == количество вхождений i-го элемента в
массив c
    Если данное количество больше эталонного (начиная с s == 0),
    то количество вхождений i-го элемента становится эталонным (s = k),
    а индекс элемента сохраняется в b.
    */
    if (k > s) s = k, b = i;
  }
  return b;
}

int F32_ref(std::vector<int>& c, int& n) {
  int s, i, k, j, b;
  for (s = 0, i = 0; i < n; i++) {
    for (k = 0, j = 0; j < n; j++)
      if (c[i] == c[j]) k++;
    if (k > s) s = k, b = i;
  }
```

```
  return b;
}

//Для работы со значением по адресу, нам необходимо разименовать указатель
(ранее A[i], теперь (*A)[i]).
int F32_pointer(std::vector<int>* c, int* n) {
  int s, i, k, j, b;
  for (s = 0, i = 0; i < (*n); i++) {
    for (k = 0, j = 0; j < (*n); j++)
      if ((*c)[i] == (*c)[j]) k++;
    if (k > s) s = k, b = i;
  }
  return b;
}
```

Приложение 1.4. Исходный код программы №50

```
/*
Функция 50:
Функция предназначена для подсчёта чередований -,+ в рамках первых 10
элементов массива. В случае, если массив состоит менее, чем из 10
элементов,
функция вызовет исключение с кодом ошибки 11 (SEGFAULT). Функционально -
абсолютно идентична F56.

Стандартные решения:
    Переменная - счётчик
    Переменная - признак
    Предыдущий, текущий, следующий (в варианте с флаговой переменной, но
логика
в целом та же)
*/
int F50_val(std::vector<int> A);
int F50_ref(std::vector<int>& A);
int F50_pointer(std::vector<int>* A);
int F50_val(std::vector<int> A) {
  int i, s, k;
  //Инициализируем i,s,k, далее движемся по массиву A от 0 до 9
включительно.
  for (i = 0, s = 0, k = 0; i < 10; i++) {
    /*
    Скажем, что k в рамках итерации - знак прошлого элемента массива.
    k = 0 если элемент был положительным,
    k = 1 в противном случае.
    */
    if (A[i] < 0)
      k = 1;
    else {
      /*
      Если прошлый элемент был отрицательным, а текущий - положителен,
      то увеличиваем счётчик (s) на 1.
      */
      if (k == 1) s++;
      k = 0;
    }
  }
  return s;
}
```

```
int F50_ref(std::vector<int>& A) {
  int i, s, k;
  for (i = 0, s = 0, k = 0; i < 10; i++) {
    if (A[i] < 0)
      k = 1;
    else {
      if (k == 1) s++;
      k = 0;
    }
  }
  return s;
}

//Для работы со значением по адресу, нам необходимо разименовать указатель
(ранее A[i], теперь (*A)[i]).
int F50_pointer(std::vector<int>* A) {
  int i, s, k;
  for (i = 0, s = 0, k = 0; i < 10; i++) {
    if ((*A)[i] < 0)
      k = 1;
    else {
      if (k == 1) s++;
      k = 0;
    }
  }
  return s;
}
```

Приложение 1.5. Исходный код программы №56.

```
/*
Функция 56:
Функция предназначена для подсчёта чередований -,+ в рамках первых 10
элементов массива. В случае, если массив состоит менее, чем из 10
элементов,
функция вызовет исключение с кодом ошибки 11 (SEGFAULT). Функционально -
абсолютно идентична F50.

Стандартные решения:
    Переменная - счётчик
    Предыдущий, текущий, следующий
*/
int F56_val(std::vector<int> A);
int F56_ref(std::vector<int>& A);
int F56_pointer(std::vector<int>* A);
int F56_val(std::vector<int> A) {
  //Инициализируем i,s, движемся по массиву A от 1 до 9 включительно.
  int i, s = 0;
  for (i = 1, s = 0; i < 10; i++)
    /*
    Если среди элементов встречено чередование -,+ (как и в прошлой
функции),
    увеличиваем счётчик на 1.
    */
    if (A[i] > 0 && A[i - 1] < 0) s++;
  return s;
}

int F56_ref(std::vector<int>& A) {
  int i, s = 0;
  for (i = 1, s = 0; i < 10; i++)
    if (A[i] > 0 && A[i - 1] < 0) s++;
  return s;
}

//Для работы со значением по адресу, нам необходимо разыменовать указатель
(ранее A[i], теперь (*A)[i]).
int F56_pointer(std::vector<int>* A) {
  int i, s = 0;
  for (i = 1, s = 0; i < 10; i++)
    if ((*A)[i] > 0 && (*A)[i - 1] < 0) s++;
  return s;
```

```
}
```

Приложение 2.1. Код вызова функций №20.

main.cpp:

```
S2L1::Output::print_f20(rval_1, rval_2, rlen_1, rlen_2);
```

output.cpp:

```
void print_f20(std::vector<int> rval_1, std::vector<int> rval_2, int
rlen_1,
               int rlen_2) {
  /*
  Здесь и далее:
  test_vector, n - копии эталонных значений для отслеживания изменений
  res - переменная под вывод функций
  */
  std::vector<int> test_vector = {};
  int n = 0;
  int res = 0;

  //Выводим ---------------------- для красоты
  printf("%s\n", NF_BREAK);

  /*
  Здесь и далее: логика вывода:
  выводим название функции, номер набора входных значений, входные
значения.
  получаем результат работы функции в res.
  выводим название функции, номер набора входных значений, выходные
значения.
  выводим название функции, номер набора входных значений, результат
работы.
  */
  test_vector = rval_1;
  n = rlen_1;
  print_function_preexec("F20_val", 1, test_vector, n);
  res = S2L1::Functions::F20_val(test_vector, n);
  print_function_postexec("F20_val", 1, test_vector, n, res);

  test_vector = rval_1;
  n = rlen_1;
  print_function_preexec("F20_ref", 1, test_vector, n);
  res = S2L1::Functions::F20_ref(test_vector, n);
```

```
  print_function_postexec("F20_ref", 1, test_vector, n, res);

  test_vector = rval_1;
  n = rlen_1;
  print_function_preexec("F20_pointer", 1, test_vector, n);
  res = S2L1::Functions::F20_pointer(&test_vector, &n);
  print_function_postexec("F20_pointer", 1, test_vector, n, res);

  printf("\n");

  test_vector = rval_2;
  n = rlen_2;
  print_function_preexec("F20_val", 2, test_vector, n);
  res = S2L1::Functions::F20_val(test_vector, n);
  print_function_postexec("F20_val", 2, test_vector, n, res);

  test_vector = rval_2;
  n = rlen_2;
  print_function_preexec("F20_ref", 2, test_vector, n);
  res = S2L1::Functions::F20_ref(test_vector, n);
  print_function_postexec("F20_ref", 2, test_vector, n, res);

  test_vector = rval_2;
  n = rlen_2;
  print_function_preexec("F20_pointer", 2, test_vector, n);
  res = S2L1::Functions::F20_pointer(&test_vector, &n);
  print_function_postexec("F20_pointer", 2, test_vector, n, res);

  printf("%s\n", NF_BREAK);
}
```

Приложение 2.2. Код вызова функций №29.

main.cpp:

```
S2L1::Output::print_f29(rval_1, rval_2, rlen_1, rlen_2);
```

output.cpp:

```
void print_f29(std::vector<int> rval_1, std::vector<int> rval_2, int
rlen_1,
               int rlen_2) {
  std::vector<int> test_vector = {};
  int n = 0;
  int res = 0;

  printf("%s\n", NF_BREAK);

  test_vector = rval_1;
  n = rlen_1;
  print_function_preexec("F29_val", 1, test_vector, n);
  res = S2L1::Functions::F29_val(test_vector, n);
  print_function_postexec("F29_val", 1, test_vector, n, res);

  test_vector = rval_1;
  n = rlen_1;
  print_function_preexec("F29_ref", 1, test_vector, n);
  res = S2L1::Functions::F29_ref(test_vector, n);
  print_function_postexec("F29_ref", 1, test_vector, n, res);

  test_vector = rval_1;
  n = rlen_1;
  print_function_preexec("F29_pointer", 1, test_vector, n);
  res = S2L1::Functions::F29_pointer(&test_vector, &n);
  print_function_postexec("F29_pointer", 1, test_vector, n, res);

  printf("\n");

  test_vector = rval_2;
  n = rlen_2;
  print_function_preexec("F29_val", 2, test_vector, n);
  res = S2L1::Functions::F29_val(test_vector, n);
  print_function_postexec("F29_val", 2, test_vector, n, res);

  test_vector = rval_2;
  n = rlen_2;
  print_function_preexec("F29_ref", 2, test_vector, n);
```

```
  res = S2L1::Functions::F29_ref(test_vector, n);
  print_function_postexec("F29_ref", 2, test_vector, n, res);

  test_vector = rval_2;
  n = rlen_2;
  print_function_preexec("F29_pointer", 2, test_vector, n);
  res = S2L1::Functions::F29_pointer(&test_vector, &n);
  print_function_postexec("F29_pointer", 2, test_vector, n, res);

  printf("%s\n", NF_BREAK);
}
```

Приложение 2.3. Код вызова функций №32.

main.cpp:

```
S2L1::Output::print_f32(rval_1, rval_2, rlen_1, rlen_2);
```

output.cpp:

```
void print_f32(std::vector<int> rval_1, std::vector<int> rval_2, int
rlen_1,
               int rlen_2) {
  std::vector<int> test_vector = {};
  int n = 0, res = 0;

  printf("%s\n", NF_BREAK);

  test_vector = rval_1;
  n = rlen_1;
  print_function_preexec("F32_val", 1, test_vector, n);
  res = S2L1::Functions::F32_val(test_vector, n);
  print_function_postexec("F32_val", 1, test_vector, n, res);

  test_vector = rval_1;
  n = rlen_1;
  print_function_preexec("F32_ref", 1, test_vector, n);
  res = S2L1::Functions::F32_ref(test_vector, n);
  print_function_postexec("F32_ref", 1, test_vector, n, res);

  test_vector = rval_1;
  n = rlen_1;
  print_function_preexec("F32_pointer", 1, test_vector, n);
  res = S2L1::Functions::F32_pointer(&test_vector, &n);
  print_function_postexec("F32_pointer", 1, test_vector, n, res);

  printf("\n");

  test_vector = rval_2;
  n = rlen_2;
  print_function_preexec("F32_val", 2, test_vector, n);
  res = S2L1::Functions::F32_val(test_vector, n);
  print_function_postexec("F32_val", 2, test_vector, n, res);

  test_vector = rval_2;
  n = rlen_2;
  print_function_preexec("F32_ref", 2, test_vector, n);
  res = S2L1::Functions::F32_ref(test_vector, n);
```

```
  print_function_postexec("F32_ref", 2, test_vector, n, res);

  test_vector = rval_2;
  n = rlen_2;
  print_function_preexec("F32_pointer", 2, test_vector, n);
  res = S2L1::Functions::F32_pointer(&test_vector, &n);
  print_function_postexec("F32_pointer", 2, test_vector, n, res);

  printf("%s\n", NF_BREAK);
}
```

Приложение 2.4. Код вызова функций №50.

main.cpp:

```
S2L1::Output::print_f50(rval_1, rval_2, rlen_1, rlen_2);
```

output.cpp:

```
void print_f50(std::vector<int> rval_1, std::vector<int> rval_2, int
rlen_1,
               int rlen_2) {
  std::vector<int> test_vector = {};
  int n = 0;
  int res = 0;

  printf("%s\n", NF_BREAK);

  test_vector = rval_1;
  n = rlen_1;
  print_function_preexec("F50_val", 1, test_vector, n);
  res = S2L1::Functions::F50_val(test_vector);
  print_function_postexec("F50_val", 1, test_vector, n, res);

  test_vector = rval_1;
  n = rlen_1;
  print_function_preexec("F50_ref", 1, test_vector, n);
  res = S2L1::Functions::F50_ref(test_vector);
  print_function_postexec("F50_ref", 1, test_vector, n, res);

  test_vector = rval_1;
  n = rlen_1;
  print_function_preexec("F50_pointer", 1, test_vector, n);
  res = S2L1::Functions::F50_pointer(&test_vector);
  print_function_postexec("F50_pointer", 1, test_vector, n, res);

  printf("\n");

  test_vector = rval_2;
  n = rlen_2;
  print_function_preexec("F50_val", 2, test_vector, n);
  res = S2L1::Functions::F50_val(test_vector);
  print_function_postexec("F50_val", 2, test_vector, n, res);

  test_vector = rval_2;
  n = rlen_2;
  print_function_preexec("F50_ref", 2, test_vector, n);
```

```
  res = S2L1::Functions::F50_ref(test_vector);
  print_function_postexec("F50_ref", 2, test_vector, n, res);

  test_vector = rval_2;
  n = rlen_2;
  print_function_preexec("F50_pointer", 2, test_vector, n);
  res = S2L1::Functions::F50_pointer(&test_vector);
  print_function_postexec("F50_pointer", 2, test_vector, n, res);

  printf("%s\n", NF_BREAK);
}
```

Приложение 2.5. Код вызова функций №56.

main.cpp:

```
S2L1::Output::print_f56(rval_1, rval_2, rlen_1, rlen_2);
```

output.cpp:

```
void print_f56(std::vector<int> rval_1, std::vector<int> rval_2, int
rlen_1,
               int rlen_2) {
  std::vector<int> test_vector = {};
  int n = 0, res = 0;

  printf("%s\n", NF_BREAK);

  test_vector = rval_1;
  n = rlen_1;
  print_function_preexec("F56_val", 1, test_vector, n);
  res = S2L1::Functions::F56_val(test_vector);
  print_function_postexec("F56_val", 1, test_vector, n, res);

  test_vector = rval_1;
  n = rlen_1;
  print_function_preexec("F56_ref", 1, test_vector, n);
  res = S2L1::Functions::F56_ref(test_vector);
  print_function_postexec("F56_ref", 1, test_vector, n, res);

  test_vector = rval_1;
  n = rlen_1;
  print_function_preexec("F56_pointer", 1, test_vector, n);
  res = S2L1::Functions::F56_pointer(&test_vector);
  print_function_postexec("F56_pointer", 1, test_vector, n, res);

  printf("\n");

  test_vector = rval_2;
  n = rlen_2;
  print_function_preexec("F56_val", 2, test_vector, n);
  res = S2L1::Functions::F56_val(test_vector);
  print_function_postexec("F56_val", 2, test_vector, n, res);

  test_vector = rval_2;
  n = rlen_2;
  print_function_preexec("F56_ref", 2, test_vector, n);
  res = S2L1::Functions::F56_ref(test_vector);
```

```
  print_function_postexec("F56_ref", 2, test_vector, n, res);

  test_vector = rval_2;
  n = rlen_2;
  print_function_preexec("F56_pointer", 2, test_vector, n);
  res = S2L1::Functions::F56_pointer(&test_vector);
  print_function_postexec("F56_pointer", 2, test_vector, n, res);

  printf("%s\n", NF_BREAK);
}
```

Приложение 3. Пример работы программы.

```
kuuk@ALuminum ~/OSS/NSTU_CPP/s2l1  inftech_s2l1 ± make test
Building s2l1.a
Building test
Running tests...
----------------------------------------
F20_val-1-входной массив      : { 447,  691,  126, -836,  748,  767,  301,  -37,  345, -164}
F20_val-1-итоговый массив     : { 447,  691,  126, -836,  748,  767,  301,  -37,  345, -164}
F20_val-1-результат           : 767
F20_ref-1-входной массив      : { 447,  691,  126, -836,  748,  767,  301,  -37,  345, -164}
F20_ref-1-итоговый массив     : { 447,  691,  126, -836,  748,  301,  -37,  345, -164}
F20_ref-1-результат           : 767
F20_pointer-1-входной массив  : { 447,  691,  126, -836,  748,  767,  301,  -37,  345, -164}
F20_pointer-1-итоговый массив : { 447,  691,  126, -836,  748,  301,  -37,  345, -164}
F20_pointer-1-результат       : 767

F20_val-2-входной массив      : {-420,    2, -160,  536, -573, -429, -233,  905,  558, -772,  445,  862, -312,    2}
F20_val-2-итоговый массив     : {-420,    2, -160,  536, -573, -429, -233,  905,  558, -772,  445,  862, -312,    2}
F20_val-2-результат           : 905
F20_ref-2-входной массив      : {-420,    2, -160,  536, -573, -429, -233,  905,  558, -772,  445,  862, -312,    2}
F20_ref-2-итоговый массив     : {-420,    2, -160,  536, -573, -429, -233,  558, -772,  445,  862, -312,    2}
F20_ref-2-результат           : 905
F20_pointer-2-входной массив  : {-420,    2, -160,  536, -573, -429, -233,  905,  558, -772,  445,  862, -312,    2}
F20_pointer-2-итоговый массив : {-420,    2, -160,  536, -573, -429, -233,  558, -772,  445,  862, -312,    2}
F20_pointer-2-результат       : 905
----------------------------------------
```

Рис.1. Работа функций №20.

```
----------------------------------------
F29_val-1-входной массив      : { 447,  691,  126, -836,  748,  767,  301,  -37,  345, -164}
F29_val-1-итоговый массив     : { 447,  691,  126, -836,  748,  767,  301,  -37,  345, -164}
F29_val-1-результат           : 1
F29_ref-1-входной массив      : { 447,  691,  126, -836,  748,  767,  301,  -37,  345, -164}
F29_ref-1-итоговый массив     : { 447,  691,  126, -836,  748,  767,  301,  -37,  345, -164}
F29_ref-1-результат           : 1
F29_pointer-1-входной массив  : { 447,  691,  126, -836,  748,  767,  301,  -37,  345, -164}
F29_pointer-1-итоговый массив : { 447,  691,  126, -836,  748,  767,  301,  -37,  345, -164}
F29_pointer-1-результат       : 1

F29_val-2-входной массив      : {-420,    2, -160,  536, -573, -429, -233,  905,  558, -772,  445,  862, -312,    2}
F29_val-2-итоговый массив     : {-420,    2, -160,  536, -573, -429, -233,  905,  558, -772,  445,  862, -312,    2}
F29_val-2-результат           : 1
F29_ref-2-входной массив      : {-420,    2, -160,  536, -573, -429, -233,  905,  558, -772,  445,  862, -312,    2}
F29_ref-2-итоговый массив     : {-420,    2, -160,  536, -573, -429, -233,  905,  558, -772,  445,  862, -312,    2}
F29_ref-2-результат           : 1
F29_pointer-2-входной массив  : {-420,    2, -160,  536, -573, -429, -233,  905,  558, -772,  445,  862, -312,    2}
F29_pointer-2-итоговый массив : {-420,    2, -160,  536, -573, -429, -233,  905,  558, -772,  445,  862, -312,    2}
F29_pointer-2-результат       : 1
----------------------------------------
```

Рис.2. Работа функций №29.

```
----------------------------------------
F32_val-1-входной массив      : { 447,  691,  126, -836,  748,  767,  301,  -37,  345, -164}
F32_val-1-итоговый массив     : { 447,  691,  126, -836,  748,  767,  301,  -37,  345, -164}
F32_val-1-результат           : 0
F32_ref-1-входной массив      : { 447,  691,  126, -836,  748,  767,  301,  -37,  345, -164}
F32_ref-1-итоговый массив     : { 447,  691,  126, -836,  748,  767,  301,  -37,  345, -164}
F32_ref-1-результат           : 0
F32_pointer-1-входной массив  : { 447,  691,  126, -836,  748,  767,  301,  -37,  345, -164}
F32_pointer-1-итоговый массив : { 447,  691,  126, -836,  748,  767,  301,  -37,  345, -164}
F32_pointer-1-результат       : 0

F32_val-2-входной массив      : {-420,    2, -160,  536, -573, -429, -233,  905,  558, -772,  445,  862, -312,    2}
F32_val-2-итоговый массив     : {-420,    2, -160,  536, -573, -429, -233,  905,  558, -772,  445,  862, -312,    2}
F32_val-2-результат           : 1
F32_ref-2-входной массив      : {-420,    2, -160,  536, -573, -429, -233,  905,  558, -772,  445,  862, -312,    2}
F32_ref-2-итоговый массив     : {-420,    2, -160,  536, -573, -429, -233,  905,  558, -772,  445,  862, -312,    2}
F32_ref-2-результат           : 1
F32_pointer-2-входной массив  : {-420,    2, -160,  536, -573, -429, -233,  905,  558, -772,  445,  862, -312,    2}
F32_pointer-2-итоговый массив : {-420,    2, -160,  536, -573, -429, -233,  905,  558, -772,  445,  862, -312,    2}
F32_pointer-2-результат       : 1
----------------------------------------
```

Рис.3. Работа функций №32.

```
----------------------------------------
F50_val-1-входной массив      : { 447,  691,  126, -836,  748,  767,  301,  -37,  345, -164}
F50_val-1-итоговый массив     : { 447,  691,  126, -836,  748,  767,  301,  -37,  345, -164}
F50_val-1-результат           : 2
F50_ref-1-входной массив      : { 447,  691,  126, -836,  748,  767,  301,  -37,  345, -164}
F50_ref-1-итоговый массив     : { 447,  691,  126, -836,  748,  767,  301,  -37,  345, -164}
F50_ref-1-результат           : 2
F50_pointer-1-входной массив  : { 447,  691,  126, -836,  748,  767,  301,  -37,  345, -164}
F50_pointer-1-итоговый массив : { 447,  691,  126, -836,  748,  767,  301,  -37,  345, -164}
F50_pointer-1-результат       : 2

F50_val-2-входной массив      : {-420,    2, -160,  536, -573, -429, -233,  905,  558, -772,  445,  862, -312,    2}
F50_val-2-итоговый массив     : {-420,    2, -160,  536, -573, -429, -233,  905,  558, -772,  445,  862, -312,    2}
F50_val-2-результат           : 3
F50_ref-2-входной массив      : {-420,    2, -160,  536, -573, -429, -233,  905,  558, -772,  445,  862, -312,    2}
F50_ref-2-итоговый массив     : {-420,    2, -160,  536, -573, -429, -233,  905,  558, -772,  445,  862, -312,    2}
F50_ref-2-результат           : 3
F50_pointer-2-входной массив  : {-420,    2, -160,  536, -573, -429, -233,  905,  558, -772,  445,  862, -312,    2}
F50_pointer-2-итоговый массив : {-420,    2, -160,  536, -573, -429, -233,  905,  558, -772,  445,  862, -312,    2}
F50_pointer-2-результат       : 3
----------------------------------------
```

Рис.4. Работа функций №50.

```
----------------------------------------
F56_val-1-входной массив      : { 447,  691,  126, -836,  748,  767,  301,  -37,  345, -164}
F56_val-1-итоговый массив     : { 447,  691,  126, -836,  748,  767,  301,  -37,  345, -164}
F56_val-1-результат           : 2
F56_ref-1-входной массив      : { 447,  691,  126, -836,  748,  767,  301,  -37,  345, -164}
F56_ref-1-итоговый массив     : { 447,  691,  126, -836,  748,  767,  301,  -37,  345, -164}
F56_ref-1-результат           : 2
F56_pointer-1-входной массив  : { 447,  691,  126, -836,  748,  767,  301,  -37,  345, -164}
F56_pointer-1-итоговый массив : { 447,  691,  126, -836,  748,  767,  301,  -37,  345, -164}
F56_pointer-1-результат       : 2

F56_val-2-входной массив      : {-420,    2, -160,  536, -573, -429, -233,  905,  558, -772,  445,  862, -312,    2}
F56_val-2-итоговый массив     : {-420,    2, -160,  536, -573, -429, -233,  905,  558, -772,  445,  862, -312,    2}
F56_val-2-результат           : 3
F56_ref-2-входной массив      : {-420,    2, -160,  536, -573, -429, -233,  905,  558, -772,  445,  862, -312,    2}
F56_ref-2-итоговый массив     : {-420,    2, -160,  536, -573, -429, -233,  905,  558, -772,  445,  862, -312,    2}
F56_ref-2-результат           : 3
F56_pointer-2-входной массив  : {-420,    2, -160,  536, -573, -429, -233,  905,  558, -772,  445,  862, -312,    2}
F56_pointer-2-итоговый массив : {-420,    2, -160,  536, -573, -429, -233,  905,  558, -772,  445,  862, -312,    2}
F56_pointer-2-результат       : 3
----------------------------------------
```

Рис.5. Работа функций №56.