МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Московский Авиационный Институт
(Национальный исследовательский университет)

Институт №8
«Компьютерный науки и прикладная математика»
Кафедра 806
«Вычислительная математика и программирование»
Курсовой проект по дисциплине «Математический практикум»
На тему «Разработка прикладных программ на языке Си для решения практических задач»

Студент: Жбанков М.А.
Группа: М8О-213Б-22
Преподаватель: Романенков А.М.
Оценка:
Дата:

Москва, 2023

Содержание.

Лабораторная работа №1.

Задание 1.

“Через аргументы командной строки программе подаются число и флаг, определяющий действие с этим числом. Флаг начинается с символа ‘-’ или ‘/’.

Программа распознает следующие флаги:

● -h вывести в консоль натуральные числа в пределах 100 включительно, кратные указанному. Если таковых нету – вывести соответствующее сообщение;

● -p определить является ли введенное число простым; является ли оно составным;

● -s разделить число на отдельные цифры и вывести отдельно каждую цифру числа, разделяя их пробелом, от старших разрядов к младшим, без ведущих нулей в строковом представлении;

● -e вывести таблицу степеней (для всех показателей в диапазоне от 1 до заданного числа) оснований от 1 до 10; для этого флага работает ограничение на вводимое число: оно должно быть не больше 10;

● -a вычислить сумму всех натуральных чисел от 1 до указанного числа включительно и вывести полученное значение в консоль;

● -f вычислить факториал указанного числа и вывести полученное значение в консоль.”

Из точки входа в приложение вызывается my_func, обеспечивающая обработку ввода.

Листинг 1. Функция обработки ввода.

```
HANDLER my_func(int argc, char* argv[]) {
  if (argc != 3) {
    return ARGUMENTS_COUNT_IR;
  }
  if (!if_flag(argv[1]) || strlen(argv[1]) >= BUFF_SIZE) {
    return FLAG_ISSUE;
  }
  char flag[BUFF_SIZE];
  strcpy_without_first(argv[1], flag);
  if (!if_ll(argv[2])) {
    return NOT_NUMBER;
  }
  long long number = strtoll(argv[2], NULL, 10);
  int code = flag_sw_case(flag, number);
```

```
  return code;
}
```
Листинг 2. Функция вызова соответсвующей для флага операции
```
int flag_sw_case(const char* flag, const long long
number) {
  if (strlen(flag) != 1) {
    return FLAG_UNDEF;
  }

  switch (flag[0]) {
    case 'h':
      return h_fl_handle(number);
    case 'p':
      return p_fl_handle(number);
    case 's':
      return s_fl_handle(number);
    case 'e':
      return e_fl_handle(number);
    case 'a':
      return a_fl_handle(number);
    case 'f':
      return f_fl_handle(number);
    default:
      return FLAG_UNDEF;
  }
}
```

Далее в каждой из функций флага вызывается соответствующая ему функция.

Флагу h соответсвует функция multiples

Листинг 3. Функция multiples
```
int multiples(int* arr, int* len, const long long number)
{
  long long n_temp = number;
  for (long long i = n_temp; i <= BUFF_SIZE; i += number)
{
```

```
    arr[(*len)] = i;
    (*len)++;
  }
  return OK;
}
```

Флагу p соответствует функция is_prime

Листинг 4. Функция is_prime

```
int is_prime(const long long number) {
  long long temp_n = number < 0 ? -number : number;
  double limit = floor(sqrt(temp_n));
  if (number == 1 || number == 0) {
    return P_NOT_COMPLEX_OR_PRIME;
  }
  if (number == 2) {
    return true;
  }
  if (number % 2 == 0) {
    return false;
  }

  for (long long i = 3; i < limit; i += 2) {
    if (temp_n % i == 0) {
      return false;
    }
  }
  return true;
}
```

Флагу s соответствует функция string_separate

Листинг 5. функция string_separate

```
int string_separate(char* buff, const int number) {
  bool flag = false;
  int len = 0;
  int temp_n = number;
  if (number < 0) {
    temp_n *= -1;
    flag = true;
  }
```

```
  int ier = 0;
  do {
    if (ier == 1) {
      ier = 0;
      buff[len++] = ' ';
    } else if (ier == 0) {
      ier = 1;

      buff[len++] = temp_n % 10 + '0';
      temp_n /= 10;
    }
  } while (temp_n != 0);

  if (flag) {
    buff[len++] = ' ';
    buff[len++] = '-';
    buff[len++] = '\0';
  }

  strrev(buff);
  return OK;
}
```

Флагу e соответствует функция exp_table

Листинг 6. Функция exp_table

```
int exp_table(int buff[10][10], const long long number) {
  for (int i = 0; i < 10; i++) {
    int x = i + 1;
    for (int j = 0; j < number; j++) {
      buff[i][j] = x;
      x *= (i + 1);
    }
  }
  return OK;
}
```

Флагу a соответствует функция sum_to_n

Листинг 7. Функция sum_to_n

```
long long int sum_to_n(const long long number) {
  if (number <= 0) {
```

```
    return A_NEGATIVE;
  }
  long long sum = 0;
  for (int i = number; i >= 1; i--) {
    if (sum > LONG_LONG_MAX - i) {
      return A_MORE_THAN_LLMAX;
    }
    sum += i;
  }
  return sum;
}
```

Флагу f соответствует функция factorial

Листинг 8. Функция factorial

```
long long factorial(const long long number) {
  if (number <= 1) {
    return 1;
  }

  long long temp = factorial(number - 1);
  if(temp == F_MORE_THAN_LLMAX) {
    return F_MORE_THAN_LLMAX;
  }
  if (temp <= LONG_LONG_MAX / number) {
    return temp * number;
  }
  return F_MORE_THAN_LLMAX;
}
```

Задание 2.

Реализуйте функции, вычисляющие значения чисел e, π, $ln2$, 2, γ с заданной точностью. Для каждой константы реализуйте три способа вычисления: как сумму ряда, как решение специального уравнения, как значение предела.

Точность вычислений (значение эпсилон) подаётся программе в качестве аргумента командной строки. Продемонстрируйте выполнение реализованных функций.

Из точки входа в приложение вызывается input_hadle, обеспечивающая обработку ввода.

Листинг 9. Функция input_handle

```
st_code input_handle(int argc, char* argv[]) {
  if (argc != 2) {
    return INVALID_ARGC;
  }
  double eps;
  char* endptr = NULL;
  eps = strtod(argv[1], &endptr);
  if (endptr != (argv[1] + strlen(argv[1])) || eps < 
EPS_LOWER_BOUND) { // endptr указывает после строки? || 
tiny eps
    return INVALID_EPS;
  }
  print_all_e(eps);
  print_all_pi(eps);
  print_all_ln2(eps);
  print_all_sqrt2(eps);
  print_all_gamma(eps);
  return INPUT_OK;
}
```

Из функции input_handle вызываются функции для вывода значений каждой константы найденных разными путями. Из этих функций вызываются функции для вычисления констант.

Листинг 10. Функция print_all_e

```
calc_st_code print_all_e(double eps) {
  double lim_res, sum_res, equ_res;
  calc_st_code a = e_lim(eps, &lim_res);
  e_sum(eps, &sum_res);
  e_equation(eps, &equ_res);
  if (a != OK) {
    printf("lim is not ok\n");
    return a;
  }
  printf("%.10lf  %.10lf  %.10lf\n",  lim_res,  sum_res, 
equ_res);
```

```
  return OK;
}
```

Листинг 11. Функция printf_all_pi

```
calc_st_code print_all_pi(double eps) {
  double lim_res, sum_res, equ_res;
  pi_lim(eps, &lim_res);
  pi_sum(eps, &sum_res);
  pi_equation(eps, &equ_res);

  printf("%.10lf   %.10lf   %.10lf\n",   lim_res,   sum_res,
equ_res);
  return OK;
}
```

Листинг 12. Функция print_all_ln2

```
calc_st_code print_all_ln2(double eps) {
  double lim_res, sum_res, equ_res;
  calc_st_code a = ln2_lim(eps, &lim_res);
  ln2_sum(eps, &sum_res);
  ln2_equation(eps, &equ_res);
  if (a != OK) {
    printf("lim is not ok\n");
    return a;
  }
  printf("%.10lf   %.10lf   %.10lf\n",   lim_res,   sum_res,
equ_res);
  return OK;
}
```

Листинг 13. Функция printf_all_sqrt2

```
calc_st_code print_all_sqrt2(double eps) {
  double lim_res, sum_res, equ_res;
  calc_st_code a = sqrt2_lim(eps, &lim_res);
  sqrt2_multiplication(eps, &sum_res);
  sqrt2_equation(eps, &equ_res);
  if (a != OK) {
    printf("lim is not ok\n");
    return a;
```

```
  }
  printf("%.10lf   %.10lf   %.10lf\n",   lim_res,   sum_res,
equ_res);
  return OK;
}
```

Листинг 14. Функция printf_all_gamma

```
calc_st_code print_all_gamma(double eps) {
  double lim_res, sum_res, equ_res;
  gamma_lim(eps, &lim_res);
  gamma_sum(eps, &sum_res);
  gamma_equation_ver2(eps, &equ_res);
  printf("%.10lf   %.10lf   %.10lf\n",   lim_res,   sum_res,
equ_res);
  return OK;
}
```

Задание 3.

Через аргументы командной строки программе подается флаг, который определяет действие, и набор чисел. Флаг начинается с символа ‘-’ или ‘/’. Необходимо проверять соответствие количества параметров введённому флагу. Программа распознает следующие флаги:

- -q первый параметр (вещественное число) задаёт точность сравнения вещественных чисел (эпсилон), оставшиеся три (вещественные числа) являются коэффициентами квадратного уравнения; необходимо вывести в консоль решения этого уравнения при всевозможных уникальных перестановках значений коэффициентов при степенях переменной;
- -m необходимо задать два ненулевых целых числа, после чего определить, кратно ли первое число второму;
- -t первый параметр (вещественное число) задаёт точность сравнения вещественных чисел (эпсилон); необходимо проверить, могут ли оставшиеся три (вещественные числа) параметра являться длинами сторон прямоугольного треугольника.

Из точки входа в приложение вызывается input, производящая обработку ввода.

Листинг 15. Функция input.

```
st_code input(int argc, char* argv[]) {
```

```
  if (argc < 2) {
    return TOO_FEW_ARGS;
  }
  if (!if_flag(argv[1])) {
    return FLAG_ERROR;
  }
  char flag[BUFF_SIZE];
  if (strlen(argv[1]) >= BUFF_SIZE) {
    return FLAG_TOO_LONG;
  }
  strcpy_without_first(argv[1], flag);
  return flags_handling(flag, argc, argv);
}
```

Из нее, в свою очередь, вызывается функция flags_handling, вызывающая соответствующую каждому флагу функцию. Для флага q вызывается функция q_fl_print, для флага m вызывается функция m_fl_print и для флага t вызывается функция t_fl_print.

Листинг 16. Функция q_fl_print.

```
st_code q_fl_print(const int argc, char* argv[]) {
  if (argc != 6) {
    return ARGC_ERROR;
  }
  if (!if_lf(argv[2]) || !if_lf(argv[3]) ||
!if_lf(argv[4]) || !if_lf(argv[5])) {
    return INVALID_NUMBER;
  }
  double ans[6][2];
  quadr_st_codes ans_errs[6];
  double first, second, third, epsilon;
  epsilon = strtod(argv[2], NULL);
  epsilon = fabs(epsilon);
  first = strtod(argv[3], NULL);
  second = strtod(argv[4], NULL);
  third = strtod(argv[5], NULL);
  int code = quadratic_eq_all_solves(ans, ans_errs, first,
second, third, epsilon);
  return code;
```

```
}
```
В матрице ans хранятся решения уравнений при перестановках переменных. В массиве ans_errs хранятся данные об уникальности конкретного решения.

Листинг 17. Функция m_fl_print.
```
st_code m_fl_print(int argc, char* argv[]) {
  if (argc != 4) {
    return ARGC_ERROR;
  }
  if (!if_ll(argv[2]) || !if_ll(argv[3])) {
    return INVALID_NUMBER;
  }
  long long first = strtoll(argv[2], NULL, 10), second =
strtoll(argv[3], NULL, 10);
  if (multiplicity(first, second)) {
    printf("OK, divisible\n");
  } else {
    printf("Not OK, not divisible\n");
  }
  return OK;
}
```

Листинг 18. Функция t_fl_print.
```
st_code t_fl_print(const int argc, char* argv[]) {
  if (argc != 6) {
    return ARGC_ERROR;
  }
  if (!if_lf(argv[2]) || !if_lf(argv[3]) ||
!if_lf(argv[4]) || !if_lf(argv[5])) {
    return INVALID_NUMBER;
  }
  double first, second, third, epsilon;
  epsilon = strtod(argv[2], NULL);
  first = strtod(argv[3], NULL);
  second = strtod(argv[4], NULL);
  third = strtod(argv[5], NULL);
  if (triangle(first, second, third, epsilon)) {
    printf("Triangle - OK\n");
```

```
  } else {
    printf("Triangle - not OK\n");
  }
  return OK;
}
```

Задание 4.

На вход программе, через аргументы командной строки, подается флаг и путь к файлу. Флаг определяет действие с входным файлом. Флаг начинается с символа ‘-’ или ‘/’. Если флаг содержит в качестве второго символа опциональный символ ‘n’ (то есть “-nd”, “/nd”, “-ni”, “/ni”, “-ns”, “/ns”, “-na”, “/na”), то путь к выходному файлу является третьим аргументом командной строки; иначе имя выходного файла генерируется приписыванием к имени входного файла префикса “out_”. Вывод программы должен транслироваться в выходной файл. Программа распознает следующие флаги:

- -d необходимо исключить символы арабских цифр из входного файла;
- -i для каждой строки входного файла в выходной файл необходимо записать сколько раз в этой строке встречаются символы букв латинского алфавита;
- -s для каждой строки входного файла в выходной файл необходимо записать сколько раз в этой строке встречаются символы, отличные от символов букв латинского алфавита, символов арабских цифр и символа пробела;
- -a необходимо заменить символы, отличные от символов цифр, ASCII кодом, записанным в системе счисления с основанием 16.

Сначала провалидируем аргументы командной строки и определим какой флаг был подан на обработку.

Из точки входа в приложение вызывается input, обеспечивающая обработку ввода. Из нее, в свою очередь, вызывается функция flags_handling, вызывающая соответствующую каждому флагу функцию. Для флага d - digit_exclude, для флага i - alpha_count, для флага s - non_digit_non_space_non_alpha_count, для флага a – non_digit_ascii_replace.

Листинг 19. Функция digit_exclude.

```
int digit_exclude(FILE* input, FILE* output) {
  if (input == NULL || output == NULL) {
    return FILE_IS_NULL;
```

```
  }
  char a = '\0';
  while (!feof(input)) {
    fscanf(input, "%c", &a);
    if (!isdigit(a)) {
      fputc(a, output);
    }
  }
  return OK;
}
```

Листинг 20. Функция alpha_count.

```
int alpha_count(FILE* input, FILE* output) {
  if (input == NULL || output == NULL) {
    return FILE_IS_NULL;
  }

  while (!feof(input)) {
    char a;
    int cntr = 0;
    while ((a = fgetc(input)) != '\n' && !feof(input)) {
      if (isalpha(a)) {
        cntr++;
      }
    }
    fprintf(output, "%d\n", cntr);
  }
  return OK;
}
```

Листинг 21. Функция non_digit_non_space_non_alpha_count.

```
int non_digit_non_space_non_alpha_count(FILE* input, FILE*
output) {
  if (input == NULL || output == NULL) {
    return FILE_IS_NULL;
  }
  while (!feof(input)) {
    char a;
    int cntr = 0;
```

```
    while ((a = fgetc(input)) != '\n' && !feof(input)) {
      if (!isalnum(a) || a != ' ') {
        cntr++;
      }
    }
    fprintf(output, "%d\n", cntr);
  }
  return OK;
}
```

Листинг 22. Функция non_digit_ascii_replace.

```
int non_digit_ascii_replace(FILE* input, FILE* output) {
  if (input == NULL || output == NULL) {
    return FILE_IS_NULL;
  }

  while (!feof(input)) {
    char a;
    while ((a = fgetc(input)) != '\n' && !feof(input)) {
      if (isdigit(a)) {
        fprintf(output, "%c\n", a);
      } else {
        fprintf(output, "%X", a);
      }
    }
    fprintf(output, "\n");
  }
  return OK;
}
```

Задание 5.

Вычислить значения сумм с точностью ε, где ε (вещественное число) подаётся программе в виде аргумента командной строки:

a. $\sum_{n=0}^{\infty} \frac{x^n}{n!}$;

b. $\sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!}$;

c. $\sum_{n=0}^{\infty} \frac{3^{3n}(n!)^3 x^{2n}}{(3n)!}$;

d. $\sum_{n=1}^{\infty} \frac{(-1)^n (2n-1)!! x^{2n}}{(2n)!!}$.

Рисунок 1. Задание 5

Из точки входа в приложение вызывается функция input, производящая обработку аргументов командой строки и вызов функций подсчета суммы.

Листинг 23. Функция input.

```
fsc input(int argc, char* argv[]) {
  if (argc != 3) {
    return ARGC_ERROR;
  }
  if (!if_lf(argv[1])) {
    return INCORRECT_ARG;
  }
  double x = strtod(argv[1], NULL);
  if (!if_lf(argv[2])) {
    return INCORRECT_ARG;
  }
  double eps = strtod(argv[2], NULL);
  if(eps <= 0)
    return INCORRECT_ARG;
  double res;
  sum_a(x, eps, &res);
  printf("a. %.10lf\n", res);

  sum_b(x, eps, &res);
  printf("b. %.10lf\n", res);

  printf("c. ");
  fsc status_code = sum_c(x, eps, &res);
  if (status_code == INCORRECT_ARG)
```

```
    printf("Enter x which absolute value is less than
1\n");
  else
    printf("%.10lf\n", res);

  printf("d. ");
  status_code = sum_d(x, eps, &res);
  if (status_code == INCORRECT_ARG)
    printf("Enter x which absolute value is less than
1\n");
  else
    printf("%.10lf\n", res);
  return OK;
}
```

Важно заметить, что для пунктов c и d невозможно вычислить значение в точках, чье абсолютное значение больше единицы.

Листинг 24. Функция подсчета суммы C.

```
fsc sum_c(double x, double eps, double* res) {
  if (fabs(x) > 1)
    return INCORRECT_ARG;
  long n = 0;
  double cur = 1;
  double sum = cur;
  double prev;
  do {
    prev = cur;
    cur = prev * pow(3 * (n + 1), 3) / (3 * n + 1) / (3 *
n + 2) * x / (3 * n + 3) * x;
    sum += cur;
    n++;
  } while (fabs(cur) > eps);
  *res = sum;
 return OK;
}
```

Листинг 25. Функция подсчета суммы D.

```
fsc sum_d(double x, double eps, double* res) {
  if (fabs(x) >= 1) {
```

```
    return INCORRECT_ARG;
  }
  long n = 1;
  double cur = -1.0 * 1 * pow(x, 2) / 2;
  double sum = cur;
  double prev;
  do {
    prev = cur;
    cur = -1.0 * prev * (2 * n + 1) / (2 * n + 2) * pow(x,
2);
    sum += cur;
    n++;
  } while (fabs(cur) > eps);
  *res = sum;
  return OK;
}
```

Задание 6.

Вычислить значения интегралов с точностью ε, где ε (вещественное число) подаётся программе в виде аргумента командной строки:

a. $\int_0^1 \frac{ln(1+x)}{x} dx$

b. $\int_0^1 e^{-\frac{x^2}{2}} dx$

c. $\int_0^1 ln\,\frac{1}{1-x}\, dx$

d. $\int_0^1 x^x dx$

Рисунок 2. Задание 6

Все интегралы, кроме интеграла C можно посчитать методом правых прямоугольников. Шаг алгоритма сперва равен 0.5; с каждой итерацией этот шаг уменьшается в двое и таким образом достигается нужная точность.

Листинг 26. Функции для вычисления интегралов.

```
int integral_a(double eps, double* res) {
  double left = 0;
```

```
  double right = 1;
  double step = 1;
  double cur = 0;
  double prev = 0;

  do {
    step /= 2;
    double sum = 0;
    double point = left + step;
    double rectangle;
    prev = cur;
    double m_eps = machine_eps();

    while (fabs(point - right) >= m_eps) {
      rectangle = step * (log(1 + point) / point);
      sum += rectangle;
      point += step;
    }
    cur = sum;
  } while (fabs(cur - prev) > eps);
  *res = cur;
  return 0;
}

int integral_b(double eps, double* res) {
  double left = 0;
  double right = 1;
  double step = 1;
  double cur = 0;
  double prev = 0;
  double m_eps = machine_eps();
  do {
    step /= 2;
    double sum = 0;
    double point = left + step;
    double rectangle;
    prev = cur;
    while (fabs(point - right) >= m_eps) {
```

```
      rectangle = step * (exp(-point * point / 2));
      sum += rectangle;
      point += step;
    }
    cur = sum;
  } while (fabs(cur - prev) > eps);
  *res = cur;
  return 0;
}

int integral_c(double eps, double* res) {
  double left = 0;
  double right = 1;
  double step = 1;
  double cur = 0;
  double prev = 0;
  double m_eps = machine_eps();
  do {
    step /= 2;
    double sum = 0;
    double point = left;
    double right_lim = right;
    double rectangle;
    prev = cur;
    while (fabs(right_lim - point) >= m_eps) {
      rectangle = step * (log(1.0 / (1 - point)));
      sum += rectangle;
      point += step;
    }
    cur = sum;
  } while (fabs(cur - prev) > eps);

  *res = cur;
  return 0;
}

int integral_d(double eps, double* res) {
  double left = 0;
```

```
  double right = 1;
  double step = 1;
  double cur = 0;
  double prev = 0;
  double m_eps = machine_eps();

  do {
    step /= 2;
    double sum = 0;
    double point = left;
    double rectangle;
    prev = cur;
    while (fabs(point - right) >= m_eps) {
      rectangle = step * (pow(point, point));
      sum += rectangle;
      point += step;
    }
    cur = sum;
  } while (fabs(cur - prev) > eps);
  *res = cur;
  return 0;
}
```

Задание 7.

На вход программе подаётся флаг и пути к файлам. Флаг начинается с символа ‘-’ или “/”. Необходимо проверять соответствие количества параметров введённому флагу. Программа распознает следующие флаги:

● -r записать в файл, путь к которому подаётся последним аргументом командной строки, лексемы из файлов *file1* и *file2*, пути к которым подаются как третий и четвёртый аргументы командной строки соответственно, где на нечётных позициях находятся лексемы из file1, а на чётных – из file2. Лексемы во входных файлах разделяются произвольным количеством символов пробела, табуляций и переносов строк. Если количество лексем в файлах различается, после достижения конца одного из входных файлов, необходимо переписать в выходной файл оставшиеся лексемы из другого из входных файлов. Лексемы в выходном файле должны быть разделены одним символом пробела.

● -a записать в файл, путь к которому подаётся последним аргументом командной строки, файл, путь к которому подаётся третьим аргументом командной строки, таким образом, чтобы:
○ в каждой десятой лексеме сначала все символы букв латинского алфавита были преобразованы в эквивалентные символы строчных букв латинского алфавита, а затем все символы были преобразованы в эквивалентные им ASCII-коды, записанные в системе счисления с основанием 4;
○ в каждой второй (и одновременно не десятой) лексеме все символы букв латинского алфавита были преобразованы в эквивалентные символы строчных букв латинского алфавита;
○ в каждой пятой (и одновременно не десятой) лексеме все символы были преобразованы в эквивалентные им ASCII-коды, записанные в системе счисления с основанием 8.
Лексемы во входном файле разделяются произвольным количеством символов пробела, табуляций и переносов строк. Лексемы в выходном файле должны быть разделены одним символом пробела.
В функции input происходит обработка ввода, из функции input вызывается функция flags_handling, вызывающая соответствующую каждому из флагов функцию.

Листинг 27. Функции input и flags_handling.
```
st_code input(int argc, char* argv[]) {
  if (argc < 2) {
    return INVALID_ARGC;
  }
  if (!if_flag(argv[1])) {
    return NOT_FLAG;
  }
  char* flag = argv[1];
  flag++;

  st_code code = flag_handling(flag, argc, argv);
  return code;
}

st_code flag_handling(char* flag, int argc, char** argv) {
  if (strlen(flag) != 1) {
    return UNKNOWN_FLAG;
```

```
  }
  switch (*flag) {
    case 'a':
      return a_fl(argc, argv);
    case 'r':
      return r_fl(argc, argv);
    default:
      return UNKNOWN_FLAG;
  }
}
```

Для нахождения лексем используется следующий алгоритм: сначала пропускаются все символы разделители, после этого все значащие символы считываются и записываются в буфер до первого разделителя, этот буфер хранит в себе лексему.

Листинг 28. функция считывающая лексему.

```
int alloc_and_get_lexema(char** str, FILE* stream) {
  int a;
  while (isspace(a = fgetc(stream)));
  if (feof(stream)) {
    return -1;
  }
  int count = 0;
  size_t size = 8;
  *str = (char*) malloc(sizeof(char) * size);

  if (*str == NULL) {
    return -1;
  }
  (*str)[count++] = a;

  while (!isspace(a = fgetc(stream)) && a != EOF) {
    (*str)[count++] = a;

    if (count >= size) {
      size *= 2;
      char* new_str = (char*) realloc(*str, sizeof(char) *
size);
```

```
      if (new_str == NULL) {
        return -1;
      }
      *str = new_str;
    }
  }
  (*str)[count] = 0;
  return 0;
}
```

Листинг 29. Функция для флага r.

```
int print_formated_str(size_t count, FILE* output, char*
str) {
  if (count % 10 == 0) {
    char buff[5];
    for (int i = 0; str[i] != '\0'; i++) {
      ascii_base_4(buff, tolower(str[i]));
      fprintf(output, "%s|", buff);
    }
  } else if (count % 2 == 0) {
    for (int i = 0; str[i] != '\0'; i++) {
      fprintf(output, "%c", tolower(str[i]));
    }
  } else if (count % 5 == 0) {
    for (int i = 0; str[i] != '\0'; i++) {
      fprintf(output, "%o|", str[i]);
    }
  } else {
    fprintf(output, "%s", str);
  }
  fprintf(output, " ");
  return 0;
}

st_code r_strange_cat(FILE* input1, FILE* input2, FILE*
output) {
  if (!input1 || !input2 || !output) {
    return FILE_IS_NULL;
```

```
  }
  while (!feof(input1) && !feof(input2)) {
    char* str = NULL;
    if (alloc_and_get_lexema(&str, input1) != -1) {
      fprintf(output, "%s ", str);
      free(str);
      str = NULL;
    }
    if (alloc_and_get_lexema(&str, input2) != -1) {
      fprintf(output, "%s ", str);
      free(str);
      str = NULL;
    }
  }
  put_rest_of_f1_to_f2(input1, output);
  put_rest_of_f1_to_f2(input2, output);
  return OK;
}
```

Листинг 30. Функции для флага a.

```
st_code a_strange_transform(FILE* input, FILE* output) {
  if (!input || !output) {
    return FILE_IS_NULL;
  }
  size_t count = 0;
  while (!feof(input)) {
    count++;
    char* str;
    if (alloc_and_get_lexema(&str, input) != -1) {
      print_formated_str(count, output, str);
      free(str);
    }
  }
  return OK;
}
```

Задание 8.

В текстовом файле, путь к которому подаётся как второй аргумент командной строки, находятся числа записанные в разных системах счисления, при этом информация о конкретной системе счисления для каждого числа утеряна. В файле числа разделены произвольным количеством разделителей (символов пробела, табуляций и переносов строки). Для каждого числа из входного файла программа должна определить минимальное основание системы счисления (в диапазоне [2..36]), в которой представление этого числа корректно, и в выходной файл, путь к которому подаётся третьим аргументом командной строки, построчно выводит: входное число без ведущих нулей, определенное для него минимальное основание системы счисления и представление этого числа в системе счисления с основанием 10. Прописные и строчные символы букв латинского алфавита отождествляются.
Функция file_handle обрабатывает файл, считывая из него лексемы, находя для них минимальное основание и записывает эту информацию в другой файл. Эта функция вызывается из функции input, которая проверяет входные данные на валидность.

Листинг 31. Функция file_handle

```
st_code file_handle(FILE* input, FILE* output) {
  if (input == NULL || output == NULL) {
    return INVALID_FILE;
  }

  do {
    char str[BUFSIZ];
    if (get_str_between_spaces(input, str, BUFSIZ) ==
STR_OK && is_str_ok_to_convert(str)) {
      int base = find_min_base(str);
      print_num_to_file(str, base, output);
    }
  } while (!feof(input));
  return OK;
}
```

Листинг 32. Функция find_min_base.

```
int find_min_base(const char* str) {
  int base = 0;
  if(*str == '-'){
```

```
    str++;

  }
  for (int i = 0; str[i] != '\0'; i++) {
    if (!isalnum(str[i])) {
      return 0;
    }
    int a = tolower(str[i]);
    a = 1 + (isdigit(a) ? (a - '0') : (a + 10 - 'a'));
    base = a > base ? a : base;
  }
  if (base == 1 || base == 0) {
    return 0;
  }
  return base;
}
```

Функция get_str_between_spaces использует схожий с алгоритмом из 1.7 алгоритм для нахождения лексемы.

Задание 9.

1. Заполнить массив фиксированного размера псевдослучайными числами в диапазоне [a..b], где a, b задаются в качестве аргументов командной строки. Реализовать функцию, выполняющую поиск максимального и минимального значений элементов массива и меняющую местами максимальный и минимальный элементы в исходном массиве за один проход по нему.
2. Заполнить динамические массивы A и B псевдослучайного размера в диапазоне [10..10000] псевдослучайными числами в диапазоне [-1000..1000]. Сформировать из них динамический массив C, где i-й элемент массива C есть i-й элемент массива A, к которому добавлено значение ближайшего к A[i] по значению элемента из массива B.

Функция get_rand_int(left, right) возвращает случайное значение из диапазона left, right. Она использует формулу Линейного Конгруэнтного Генератора. Чтобы задать начальное значение используется функция set_rand_seed.

Листинг 33. Функции для генерации случайного числа.

```
int get_random_int(int left, int right) {
  if (left > right) {
    return 0;
  }

  static const int A_rand = 1103515245;
  static const int C_rand = 12345;
  const int mod = right - left + 1;
  __rand_seed = (A_rand * __rand_seed + C_rand);
  return __rand_seed % mod + left;
}

void set_random_seed(int seed) {
  __rand_seed = seed;
}
```

Листинг 34. Функция для слияния двух массивов.

```
void merge_arrays(const int arr_a[], const int arr_b[],
int arr_c[], int size) {
  for (int i = 0; i < size; i++) {
    arr_c[i] = arr_a[i] + find_closest_to_elem(arr_b,
size, arr_a[i]);
  }
}
```

Листинг 35. Функция find_closest_to_elem.

```
int find_closest_to_elem(const int arr[], const int size,
const int elem) {
  int closest = arr[0];
  for (int i = 1; i < size; i++) {
    if (abs(arr[i] - elem) < abs(closest - arr[i])) {
      closest = arr[i];
    }
  }
  return closest;
}
```

Листинг 36. Функция start, вызывающая функции заполнения и слияния массивов

```
void start(int left, int right) {
```

```
  set_random_seed((int) time(NULL));

  int arr[ARR_SIZE];
  fill_array_with_rands(arr, ARR_SIZE, left, right);
  printf("1. Unchanged:\t");
  print_array(arr, ARR_SIZE);
  swap_max_and_min(arr, ARR_SIZE);
  printf("Changed:\t");
  print_array(arr, ARR_SIZE);
  putchar('\n');
  int size = get_random_int(10, 10000);
  int* arr_A = (int*) malloc(sizeof(int) * size);
  int* arr_B = (int*) malloc(sizeof(int) * size);
  int* arr_C = (int*) malloc(sizeof(int) * size);
  fill_array_with_rands(arr_A, size, -1000, 1000);
  fill_array_with_rands(arr_B, size, -1000, 1000);
  merge_arrays(arr_A, arr_B, arr_C, size);
  print_array(arr_C, size);
  free(arr_A);
  free(arr_B);
  free(arr_C);
}
```

Задание 10.

Пользователь вводит в консоль основание системы счисления (в диапазоне [2..36]) и затем строковые представления целых чисел в системе счисления с введённым основанием (цифры со значением больше 9 должны вводиться как прописные буквы латинского алфавита). Окончанием ввода является ввод строки “Stop”. Найти среди введённых чисел максимальное по модулю. Напечатать его без ведущих нулей, а также его строковые представления в системах счисления с основаниями 9, 18, 27 и 36.

Из функции main вызывается функция find_max_from_input, которая запрашивает у пользователя основание системы счисления и считывает числа из стандартного потока ввода. Из всех считанных чисел она находит наибольшее и записывает его в параметр result.

Листинг 37. Функция find_max_from_input.

```
myf_sc find_max_from_input(char** string, int* cap, long*
result) {
  printf("Enter base: ");
  get_string_safely_realloc(string, cap, stdin);
  char* endptr;
  long base = strtol(*string, &endptr, 10);
  if (endptr != *string + strlen(*string)) {
    return INCORRECT_STRING;
  }
  if (base < 2 || base > 36) {
    return INVALID_BASE;
  }

  long max_ = 0;
  input_numbers(string, cap, &max_, base);
  *result = max_;
  return OK;
}
```

Она использует функцию get_string_safely_realloc, которая считывает строку из стандартного потока ввода и, при необходимости, довыделяет место.
Далее в функции main это число выводится в разных системах счисления, для этого используется функция show_number_with_base.

Листинг 38. Функция show_number_with_base.

```
void show_number_with_base(long num, const int base) {
  if(num == 0){
    printf("0\n");
    return;
  }
  char buff[66];
  buff[65] = 0;
  char* ptr = buff + 64;
  bool if_negative = false;
  if (num < 0) {
    num *= -1;
    if_negative = true;
  }
```

```
  while (num > 0) {
    int rem = num % base;
    num /= base;
    if (rem >= 0 && rem <= 9) {
      *ptr = rem + '0';
    } else if (rem >= 10) {
      *ptr = rem + 'A' - 10;
    }
    ptr--;
  }
  if (if_negative) {
    *ptr = '-';
    ptr--;
  }
  ptr++;
  printf("%s\n", ptr);
}
```

Лабораторная работа №2.

Задание 1.

Через аргументы командной строки программе подаются флаг (первым аргументом), строка (вторым аргументом); и (только для флага -c) целое число типа *unsigned int* и далее произвольное количество строк. Флаг определяет действие, которое необходимо выполнить над переданными строками:

●-l подсчёт длины переданной строки, переданной вторым аргументом;

●-r получить новую строку, являющуюся перевёрнутой (reversed) переданной вторым
аргументом строкой;

●-u получить новую строку, идентичную переданной вторым аргументом, при этом
каждый символ, стоящий на нечётной позиции (первый символ строки находится на
позиции 0), должен быть преобразован в верхний регистр;

●-n получить из символов переданной вторым аргументом строки новую строку так,
чтобы в начале строки находились символы цифр в исходном порядке, затем символы букв в исходном порядке, а в самом конце – все остальные символы, также в исходном порядке;

●-c получить новую строку, являющуюся конкатенацией второй, четвёртой, пятой и т. д. переданных в командную строку строк; строки конкатенируются в псевдослучайном порядке; для засеивания генератора псевдослучайных чисел функцией *srand* используйте *seed* равный числу, переданному в командную строку третьим аргументом.

Для каждого флага необходимо реализовать соответствующую ему собственную функцию, выполняющую действие. Созданные функциями строки должны располагаться в выделенной из динамической кучи памяти. При реализации функций запрещено использование функций из заголовочного файла *string.h*. Продемонстрируйте работу реализованных функций.

Из точки входа в программу вызывается функция input, обрабатывающая аргументы переданные в программу; функция input вызывает функцию flag_compare, которая вызывает соответствующую определенному флагу функцию. Для флага l - l_fl, для флага r - r_fl, для флага u - u_fl, для флага n - n_fl, для флага c – c_fl.

Листинг 39. Функция l_fl.

```
st_code l_fl(int argc, char* argv[]) {
  if (argc != 3) {
    return inv_argc;
  }

  printf("length: %lld\n", str_length(argv[2]));
  return ok;
}
```

Листинг 40. Функция r_fl.

```
st_code r_fl(int argc, char* argv[]) {
  if (argc != 3) {
    return inv_argc;
  }
  char* string = (char*) malloc(sizeof(char) *
(str_length(argv[2]) + 1));
  if (string == NULL) {
    return not_enough_space;
  }
  str_reverse(argv[2], string);
  printf("reversed string: %s\n", string);
  free(string);
  return ok;
}
```

Листинг 41. Функция u_fl.

```
st_code u_fl(int argc, char* argv[]) {
  if (argc != 3) {
    return inv_argc;
  }
  char* string = (char*) malloc(sizeof(char) *
(str_length(argv[2]) + 1));
  if (string == NULL) {
    return not_enough_space;
  }
  str_odd_element_toupper(argv[2], string);
  printf("formatted string: %s\n", string);
```

```
  free(string);
  return ok;
}
```

Листинг 42. Функция n_fl.

```
st_code n_fl(int argc, char* argv[]) {
  if (argc != 3) {
    return inv_argc;
  }
  char* string = (char*) malloc(sizeof(char) *
(str_length(argv[2]) + 1));
  if (string == NULL) {
    return not_enough_space;
  }
  str__num_alph_other__order(argv[2], string);
  printf("formatted string: %s\n", string);
  free(string);
  return ok;
}
```

Листинг 43. Функция c_fl.

```
st_code c_fl(int argc, char* argv[]) {
  if (argc <= 3) {
    return inv_argc;
  }

  if (!if_i(argv[2])) {
    return inv_num;
  }
  int seed = strtol(argv[2], NULL, 10);
  ull size = 0;
  for (int i = 3; i < argc; i++) {
    size += str_length(argv[i]);
  }
  char* string = (char*) malloc(sizeof(char) * (size +
1));
  if (string == NULL) {
    return not_enough_space;
  }
```

```
  if(cat_string_rand_order(seed, argv + 3, argc - 3,
string) == 0) {
    printf("random ordered string: %s\n", string);
  }

  free(string);
  return ok;
}
```

Каждая из этих функций вызывает одну или более реализованных функций (str_length, str_reverse, str_odd_element_toupper, str__num_alph_other__order, cat_string_rand_order)

Листинг 44. Функция str_length.

```
ull str_length(const char* str) {
  if (str == NULL) {
    return 0;
  }
  ull size = 0;
  for (; str[size] != 0; size++);
  return size;
}
```

Листинг 45. Функция str_reverse.

```
int str_reverse(const char* src, char* dest) {
  if (src == NULL || dest == NULL) {
    return -1;
  }

  ull len = str_length(src);
  for (int i = 0; i <= len / 2; i++) {
    dest[i] = src[len - 1 - i];
    dest[len - 1 - i] = src[i];
  }
  dest[len] = '\0';
  return 0;
}
```

Листинг 46. Функция str_odd_element_toupper.

```
int str_odd_element_toupper(const char* src, char* dest) {
  if (src == NULL || dest == NULL) {
    return -1;
  }
  ull i = 0;
  for (; src[i] != '\0'; i++) {
    if (i & 1) {
      dest[i] = toupper(src[i]);
    } else {
      dest[i] = src[i];
    }
  }
  dest[i] = '\0';
  return 0;
}
```

Листинг 47. Функция str__num_alph_other__order.

```
int str__num_alph_other__order(const char* src, char*
dest) {
  if (src == NULL || dest == NULL) {
    return -1;
  }
  int ind = 0;
  for (int i = 0; src[i] != '\0'; i++) {
    if (isdigit(src[i])) {
      dest[ind++] = src[i];
    }
  }
  for (int i = 0; src[i] != '\0'; i++) {
    if (isalpha(src[i])) {
      dest[ind++] = src[i];
    }
  }
  for (int i = 0; src[i] != '\0'; i++) {
    if (!isalnum(src[i])) {
      dest[ind++] = src[i];
    }
  }
```

```
  dest[ind] = '\0';
  return 0;
}
```

Листинг 48. Функция cat_string_rand_order.

```
char* str_cpy(const char* src, char* dest) {
  int i = 0;
  for (; src[i] != '\0'; i++) {
    dest[i] = src[i];
  }
  dest[i] = '\0';
  return dest + i;
}

int rand_comp(const void* i, const void* j) {
  return rand() % 2 ? 1 : -1;
}

int cat_string_rand_order(int seed, char* strings_arr[],
int str_cnt, char* dest) {
  srand(seed);
  char** arr = (char**) malloc(sizeof(char*) * str_cnt);
  if (arr == NULL) {
    return -1;
  }
  for (int i = 0; i < str_cnt; i++) {
    arr[i] = strings_arr[i];
  }

  qsort(arr, str_cnt, sizeof(char*), rand_comp);
  *dest = '\0';
  char* ptr = dest;
  for (int i = 0; i < str_cnt; i++) {
    ptr = str_cpy(arr[i], ptr);
  }
  free(arr);
  return 0;
}
```

Эта функция использует qsort и псевдослучайный компаратор чтобы составить случайную последовательность индексов строк. В этом порядке строки будут сконкатенированы.

Задание 2.

1. Реализуйте функцию с переменным числом аргументов, вычисляющую среднее геометрическое переданных ей чисел вещественного типа. Количество (значение типа *int*) переданных вещественных чисел задаётся в качестве последнего обязательного параметра функции.
2. Реализуйте рекурсивную функцию возведения вещественного числа в целую степень. При реализации используйте алгоритм быстрого возведения в степень. Продемонстрируйте работу реализованных функций.

В этом задании используются функции с переменным числом аргументов. Для работы с ними сушествует набор макросов va_.

Листинг 49. Функция для подсчета среднего геометрического.

```
int geometric_mean(double* res, int count, ...) {
  va_list ap;
  va_start(ap, count);
  double product = 1;
  for (int i = 0; i < count; ++i) {
    double temp = va_arg(ap, double);
    product *= temp;
  }
  va_end(ap);
  *res = pow(product, 1.0 / count);
  return 0;
}
```

Листинг 50. Функция res_power рекурсивного возведения в степень и ее обертка — power.

```
double rec_power(double exp, int power_it) {
  if (power_it == 1) {
    return exp;
  }
  if (power_it == 0) {
    return 1;
```

```
  }
  if ((power_it & 1) == 1) {
    return exp * rec_power(exp * exp, power_it >> 1);
  } else {
    return rec_power(exp * exp, power_it >> 1);
  }
}

int power(double num, int power, double* res) {
  int flag = 0;
  if (power < 0) {
    flag = 1;
    power = -power;
  }

  *res = rec_power(num, power);
  if (flag) {
      *res = 1 / (*res);
  }

  return 0;
}
```

Задание 3.

Реализуйте функцию с переменным числом аргументов, принимающую в качестве входных параметров подстроку и пути к файлам. Необходимо чтобы для каждого файла функция производила поиск всех вхождений переданной подстроки в содержимом этого файла. При реализации запрещается использование функций strstr и strncmp из заголовочного файла *string.h*. Продемонстрируйте работу функции, также организуйте наглядный вывод результатов работы функции: для каждого файла, путь к которому передан как параметр Вашей функции, для каждого вхождения подстроки в содержимое файла необходимо вывести номер строки (индексируется с 1) и номер символа в строке файла, начиная с которого найдено вхождение подстроки. В подстроку могут входить любые символы (обратите внимание, что в подстроку также могут входить символы пробела, табуляций, переноса строки, в неограниченном количестве).

В этом задании также используются функции с переменным числом аргументов. Функция find_all_substr_in_multiple_files искомую строку, указатель на массив массивов структур, где будут храниться данные о найденных подстроках для каждого файла, количество этих файлов и имена этих фалов. Функция find_all_substr принимает искомую строку, имя файла, а также указатель на массив структур где будт храниться данные о найденных подстроках для этого файла. Функция find_substr_in_file принимает искомую строку, указатель на структуру где будет храниться информация о найденной подстроке, сам файл, а также положение указателя (текущую строку а также колонку).

Листинг 51. Функция find_all_substr_in_multiple_files.

```
find_substr_st
find_all_substr_in_multiple_files(char_info***
indexes_arr, int count, const char* substr, ...) {
  if(substr[0] == 0) {
    return find_str_inv;
  }
  va_list a;
  if (count == 0) {
    return find_cnt_inv;
  }
  *indexes_arr = (char_info**) malloc(sizeof(char_info*) *
count);

  va_start(a, substr);
  for (int i = 0; i < count; i++) {
    char* filename = va_arg(a, char*);
    char_info* indexes;
    int rv = find_all_substr(substr, filename, &indexes);
    if (rv == find_ok) {
      (*indexes_arr)[i] = indexes;
    } else{
      (*indexes_arr)[i] = NULL;
    }
  }
  va_end(a);
  return 0;
```

```
}
```

Листинг 52. Функция find_all_substr.

```
find_substr_st find_all_substr(const char* str, const
char* filename, char_info** substr_indexes) {
  FILE* stream = fopen(filename, "r");
  if (stream == NULL) {
    return find_stream_null;
  }
  *substr_indexes = (char_info*)
malloc(sizeof(char_info));
  if (*substr_indexes == NULL) {
    return find_bad_alloc;
  }

  size_t size = 1;
  char_info res_index, cur_index = {.index = -1, .line =
1};
  int rv;
  while ((rv = find_substr_in_file(&res_index, str,
stream, &cur_index)) == find_ok) {
    char_info* temp;
    (*substr_indexes)[size - 1] = res_index;
    temp = (char_info*) realloc(*substr_indexes, (++size)
* sizeof(char_info));
    if (temp == NULL) {
      free(*substr_indexes);
      substr_indexes = NULL;
      return find_bad_alloc;
    }
    *substr_indexes = temp;
  }
  if (rv == find_not_ok) {
    free(*substr_indexes);
    *substr_indexes = NULL;
    return find_bad_alloc;
  }
  (*substr_indexes)[size - 1].line = -1;
```

```
  fclose(stream);
  return find_ok;
}
```

Листинг 53. Функция find_substr_in_file.

```
find_substr_st find_substr_in_file(char_info* info, const
char* str, FILE* stream, char_info* ptr_pos) {
  size_t len = strlen(str);
  char* strbuf = (char*) malloc(sizeof(char) * (len + 1));
  if (strbuf == NULL) {
    return find_not_ok;
  }
  strbuf[len] = '\0';
  while (fread(strbuf, sizeof(char), len, stream) == len)
{
    fseek(stream, 1 - len, SEEK_CUR);
    ptr_pos->index++;
    if (is_equal_s(str, strbuf)) {
      info->index = ptr_pos->index;
      info->line = ptr_pos->line;
      if (*strbuf == '\n') {
        ptr_pos->line++;
        ptr_pos->index = -1;
      }
      return find_ok;
    }
    if (*strbuf == '\n') {
      ptr_pos->line++;
      ptr_pos->index = -1;
    }
  }
  return find_eof;
}
```

Задание 4.

1. Реализуйте функцию с переменным числом аргументов, принимающую координаты (вещественного типа, пространство двумерное) вершин

многоугольника и определяющую, является ли этот многоугольник выпуклым.
2. Реализуйте функцию с переменным числом аргументов, находящую значение многочлена степени *n* в заданной точке. Входными параметрами являются точка (вещественного типа), в которой определяется значение многочлена, степень многочлена (целочисленного типа), и его коэффициенты (вещественного типа, от старшей степени до свободного коэффициента в порядке передачи параметров функции слева направо).
Продемонстрируйте работу реализованных функций.

Для проверки многоугольника на выпуклость я использовал тот факт, что при проходе вершин поворот должен происходить всегда в одну сторону. Нельзя также забывать соединить последнюю точку с первой. Для этого я использовал структуру точки и структуру вектора.

Листинг 54. Структуры точки и вектора.
```
typedef struct point2__ {
  double x;
  double y;
} point2;

typedef struct vector2__ {
  double x;
  double y;
} vector2;
```

Листинг 55. Функция проверки на выпуклость.
```
int is_convex_polygon(unsigned int n, ...) {
  if (n < 3) {
    return -1;
  }
  if (n == 3) {
    return 1;
  }
  va_list ap;
  va_start(ap, n);
  point2 x1 = va_arg(ap, point2), x2 = va_arg(ap, point2);
  vector2 prev_vec, cur_vec;
  make_vector(x1, x2, &prev_vec);
```

```
  point2 start_point = x1;
  x1 = x2;
  x2 = va_arg(ap, point2);
  make_vector(x1, x2, &cur_vec);
  double        rotation_prev,        rotation_cur        =
rotation_val(prev_vec, cur_vec);
  for (int i = 3; i < n; i++) {
    x1 = x2;
    x2 = va_arg(ap, point2);

    prev_vec = cur_vec;
    make_vector(x1, x2, &cur_vec);

    rotation_prev = rotation_cur;
    rotation_cur = rotation_val(prev_vec, cur_vec);

    if (rotation_cur * rotation_prev <= 0) {
      va_end(ap);
      return 0;
    }
  }
  va_end(ap);
  x1 = x2;
  x2 = start_point;

  prev_vec = cur_vec;
  make_vector(x1, x2, &cur_vec);

  rotation_prev = rotation_cur;
  rotation_cur = rotation_val(prev_vec, cur_vec);

  if (rotation_cur * rotation_prev <= 0) {
    return 0;
  }

  return 1;
}
```

Задание 7.

Реализуйте функцию, которая находит корень уравнения одной переменной методом дихотомии. Аргументами функции являются границы интервала, на котором находится корень; точность (эпсилон), с которой корень необходимо найти, а также указатель на функцию, связанной с уравнением от одной переменной. Продемонстрируйте работу функции на разных значениях интервалов и точности, для различных уравнений.

Листинг 56. Функция выполняющая дихотомический поиск.

```
int dichotomy(double* res, double left_bound, double
right_bound, double eps, double (*func)(double)) {
  if (func(left_bound) * func(right_bound) > 0) {
    *res = 0;
    return 1;
  }
  if(eps <= 0){
    return 2;
  }
  double mid;
  double f_mid;
  do {
    mid = (left_bound + right_bound) / 2;
    f_mid = func(mid);
    if (f_mid * (func(left_bound)) > 0) {
      left_bound = mid;
    }
    if (f_mid * (func(right_bound)) > 0) {
      right_bound = mid;
    }
  } while (fabs(f_mid) > eps);
  *res = mid;

  return 0;
}
```

Для корректного исполнения функции необходимо гарантировать единственность решения на интервале.

Задание 8.

Реализуйте функцию с переменным числом аргументов, вычисляющую сумму переданных целых неотрицательных чисел в заданной системе счисления с основанием [2..36]. Параметрами функции являются основание системы счисления, в которой производится суммирование, количество переданных чисел, строковые представления чисел в заданной системе счисления. Десятичное представление переданных чисел может быть слишком велико и не поместиться во встроенные типы данных; для решения возникшей проблемы также реализуйте функцию «сложения в столбик» двух чисел в заданной системе счисления, для её использования при реализации основной функции. Результирующее число не должно содержать ведущих нулей. Продемонстрируйте работу функции.

Для поиска суммы реализуем функцию сложения двух чисел в строковом представлении с заданным основанием.

Листинг 57. Функция, добавляющая к res число num в системе счисления с основанием base.

```
sum_st_code sum_up(int base, char** res, size_t* res_size,
const char* num) {
  if (base < 2 || base > 36) {
    return sum_base_inv;
  }
  if (!is_num_valid(num, base)) {
    return sum_num_inv;
  }

  size_t len1 = strlen(*res);
  num = skip_leading_zeros(num);
  size_t len2 = strlen(num);
  //num по размеру больше чем res

  if (len2 > len1) {
    while (len2 > *res_size) {
      size_t new_sz = len2 * 2;
      char* temp_p = realloc(*res, new_sz);
      if (temp_p == NULL) {
        return sum_bad_alloc;
```

```
    }
    *res = temp_p;
    *res_size = new_sz;
  }
  //добавили нулей, сровняли по размеру
  shift_str_for_i(*res, len2 - len1);
  len1 = strlen(*res);
}

bool if_plus_one = false;
long long cntr = 0;
//суммируем с конца

while (cntr < len2) {
  int n1 = ascii_to_int((*res)[(len1 - 1) - cntr]);
  int n2 = ascii_to_int(num[(len2 - 1) - cntr]);
  int sum = n1 + n2;
  if (if_plus_one) {
    sum += 1;
    if_plus_one = false;
  }
  if (sum >= base) {
    if_plus_one = true;
    sum -= base;
  }
  (*res)[(len1 - 1) - cntr] = int_to_ascii(sum);

  cntr++;
}

//num закончился, добиваем переносимую еденичку
while (if_plus_one) {
  if (cntr == len1) {
    if (len1 >= *res_size) {
      size_t new_sz = *res_size + 1;
      char* temp_p = realloc(*res, new_sz);
      if (temp_p == NULL) {
        return sum_bad_alloc;
```

```
        }

        *res = temp_p;
        *res_size = new_sz;
      }
      shift_str_for_i(*res, 1);
      len1 += 1;
    }

    int sum = ascii_to_int((*res)[(len1 - 1) - cntr]) + 1;

    if_plus_one = false;
    if (sum >= base) {
      if_plus_one = true;
      sum -= base;
    }
    (*res)[(len1 - 1) - cntr] = int_to_ascii(sum);
    cntr++;
  }

  return sum_ok;
}
```

Листинг 58. Функция суммы с переменным числом аргументов.

```
sum_st_code sum_base_n(size_t cnt, int base, char** res,
...) {
  if (cnt == 0) {
    return sum_cnt_inv;
  }
  if (base < 2 || base > 36) {
    return sum_base_inv;
  }

  va_list a;
  size_t size = 1024;
  char* sum_res = malloc(size * sizeof(char));
  if (sum_res == NULL) {
    return sum_bad_alloc;
  }
```

```
  strcpy(sum_res, "0");
  va_start(a, res);
  for (int i = 0; i < cnt; i++) {
    char* t_num = va_arg(a, char*);
    switch (sum_up(base, &sum_res, &size, t_num)) {
      case sum_bad_alloc:
        free(sum_res);
        return sum_bad_alloc;
      case sum_num_inv:
        printf("%s is invalid\n", t_num);
        break;
      default:
        break;
    }
  }
  res = &sum_res;
  printf("sum result is - %s\n", sum_res);
  va_end(a);
  return sum_ok;
}
```

Задание 9.

Реализуйте функцию с переменным числом аргументов, определяющую для каждой из переданных десятичных дробей (в виде значения вещественного типа в диапазоне (0; 1)), имеет ли она в системе счисления с переданным как параметр функции основанием конечное представление. Продемонстрируйте работу функции.

Функция check_base_multiple_nums для проверки нескольких чисел для каждого числа вызывает check_base

Листинг 59. Функции check_base и check_base_multiple_nums

```
bool check_base(double num, unsigned int base, double eps)
{
  if (base < 2) {
    return 0;
  }
  if (num <= 0 || num >= 1) {
```

```
    return 0;
  }

  eps = m_eps();
  unsigned long long denom = 1, numer = 1;
  while (fabs(num - floor(num)) >= eps) {
    denom *= 10;
    num *= 10;
    numer = floor(num);
  }
  denom = denom / gcd(denom, numer);

  unsigned int t_base = base;
  while (t_base < denom) {
    t_base *= base;
  }

  if (t_base == denom) {
    return true;
  }

  return false;
}

int  check_base_multiple_nums(int**  result,  int  count,
unsigned int base, ...) {
  *result = (int*) malloc(sizeof(int) * count);
  if(*result == NULL){
    return -1;
  }
  va_list a;
  va_start(a, base);
  double eps = m_eps();
  for (int i = 0; i < count; i++) {
    double temp = va_arg(a, double);
    (*result)[i] = check_base(temp, base, eps);
  }
  va_end(a);
```

```
  return 0;
}
```

Для проверки числа на возможность представления в виде простой дроби необходимо максимально сократить простую дробь, для этого пригодится функция для нахождения наибольшего общего делителя

Листинг 60. Функция нахождения НОД

```
unsigned long long gcd(unsigned long long a, unsigned long
long b) {
  if (a % b == 0) {
    return b;
  }
  if (b % a == 0) {
    return a;
  }
  if (a > b) {
    return gcd(a % b, b);
  }
  return gcd(a, b % a);
}
```

Если base делится на знаменатель этой простой дроби, значит число можно представить в системе счисления с основанием base.

Задание 10.

Реализуйте функцию с переменным числом аргументов, находящую переразложение многочлена со степенями значения x по степеням значения (x - a). Входными аргументами этой функции являются точность ε, значение a (вещественного типа), указатель на место в памяти, по которому должен быть записан указатель на динамически выделенную в функции коллекцию коэффициентов результирующего многочлена, степень многочлена (целочисленного типа) и его коэффициенты (вещественного типа), передаваемые как список аргументов переменной длины. То есть, если задан многочлен $f(x)$, то необходимо найти коэффициенты $g\ 0$, $g\ 1$, …, $g\ n$ такие что:

$$f(x) = f_0 + f_1 x + f_2 x^2 + \ldots + f_n x^n = g_0 + g_1 (x - a) + g_2 (x - a)^2 + \ldots + g_n (x - a)^n.$$

Рисунок 3. Задание 10.

Продемонстрируйте работу функции.

Для выполнения задания использую Ряд Тейлора.

Листинг 61. Функция для переразложения многочлена.

```
st_code tr_polynomial(size_t n, double x0, double eps,
double** result_k, ...) {
  if (n == 0) {
    return inv_n;
  }

  if (eps <= 0) {
    return inv_eps;
  }

  double* inp_k = (double*) malloc(sizeof(double) * n);
  if (inp_k == NULL) {
    return bad_alloc;
  }
  *result_k = (double*) malloc(sizeof(double) * n);
  if (result_k == NULL) {
    free(inp_k);
    return bad_alloc;
  }
  va_list a;
  va_start(a, result_k);
  for (int i = 0; i < n; i++) {
    inp_k[i] = va_arg(a, double);
  }
  va_end(a);

  for (int i = 0; i < n; i++) {

    calc_polynomial(x0, inp_k, n - i, &((*result_k)[i]));

    for (int j = 2; j < i + 1; j++) {
      (*result_k)[i] /= j;
```

```
    }

    calc_differ(inp_k, n, i);
  }

  free(inp_k);
  return ok;
}
```

Листинг 62. Функция для дифференцирования.

```
st_code calc_differ(double* coefs, size_t n, int iter) {
  for (int j = 0; j < n - iter; j++) {
    coefs[j] = (j + 1) * coefs[j + 1];
  }
  coefs[n - iter] = 0;
  return ok;
}
```

Листинг 63. Функция для вычисления значения многочлена в точке x0.

```
st_code calc_polynomial(double x0, double* coefs, size_t
n, double* res) {
  *res = 0;
  for (int i = 0; i <= n; ++i) {
    *res *= x0;
    *res += coefs[n - i];
  }
  return ok;
}
```

Лабораторная работа №3.

Задание 1.

Реализуйте функцию перевода числа из десятичной системы счисления в систему счисления с основанием 2 . При реализации функции разрешается r , r = 1, …, 5 использовать битовые операции и операции обращения к памяти, запрещается использовать стандартные арифметические операции. Продемонстрируйте работу реализованной функции.

Для решения этой задачи очевидным решением будет использовать битовые операции. Функция base10_to_base_2pow_r переводит число num в нужную степень и записывает значения каждой цифры слева направо в вектор result, барьерным элементом выступает -1.

Листинг 64. Функция base10_to_base_2pow_r.

```
int base10_to_base_2pow_r(unsigned long long num, unsigned
int r, int** result) {
  if (r < 1 || r > 5) {
    return 1;
  }
  int cnt = 0;
  if (num == 0) {
    cnt = 1;
  }

  unsigned long long t_n = num;
  while (t_n) {
    cnt++;
    t_n >>= r;
  }

  *result = (int*) malloc(sizeof(int) * (cnt + 1));
  if(*result == NULL){
    return -1;
  }

  for (int i = cnt - 1; i >= 0; i--) {
    unsigned int t = num >> r;
    t <<= r;
```

```
    (*result)[i] = t ^ num;
    num >>= r;
  }
  (*result)[cnt] = -1;

  return 0;
}
```

Задание 2.

Напишите функцию с переменным числом аргументов, на вход которой передаются: размерность пространства n; экземпляры структур, содержащие в себе координаты векторов из n-мерного пространства; указатели на функции, вычисляющие нормы векторов. Ваша функция должна для каждой переданной нормы вернуть самый длинный переданный вектор (если таковых векторов несколько, необходимо вернуть их все). Замечание. Реализуйте возможность вычислять следующие нормы:

$$\|x\|_\infty = max_j |x_j|,$$

$$\|x\|_p = \left(\sum_{j=1}^{n} |x_j|^p \right)^{\frac{1}{p}}, \; p \geq 1,$$

$$\|x\|_A = \sqrt{(Ax,\ x)},$$

Рисунок 4. Задание 2.

Функции вычисления норм векторов заданы как norm (для первой), norm_p (для второй), norm_m (для третьей).

Листинг 65. Функция norm.

```
st_code norm(vector* vec, double* nrm) {
    double mx = 0.0;
    for (unsigned int i = 0; i < vec->n; i++) {
        mx = fmax(mx, fabs(vec->elems[i]));
    }
```

```
    *nrm = mx;

    return OK;
}
```

Листинг 66. Функция norm_p.

```
st_code norm_p(vector* vec, unsigned int p, double* nrm) {
    if (p < 1) {
        return INVALID_ARG;
    }

    double sum = 0.0;
    for (unsigned int i = 0; i < vec->n; i++) {
        sum += pow(fabs(vec->elems[i]), p);
    }
    *nrm = pow(sum, 1.0 / p);

    return OK;
}
```

Листинг 67. Функция norm_m.

```
st_code norm_m(vector* vec, double* nrm) {
    double** A = (double**) malloc(sizeof(double*) * vec-
>n);
    if (A == NULL) {
        return BAD_ALLOC;
    }
    for (int i = 0; i < vec->n; i++) {
        A[i] = (double*) malloc(sizeof(double) * vec->n);
        if (A[i] == NULL) {
            for (int j = 0; j < i; j++) {
                free(A[j]);
            }
            free(A);
            return BAD_ALLOC;
        }
        // Единичная матрица - положительно определенная
        for (int j = 0; j < vec->n; ++j) {
            if (i == j) {
```

```
                A[i][j] = 1;
            }
            else {
                A[i][j] = 0;
            }
        }
    }

    double* mult_res = NULL;
    if (matrix_multipl(A, vec->elems, &mult_res, vec->n)
== BAD_ALLOC) {
        for (int i = 0; i < vec->n; ++i) {
            free(A[i]);
        }
        free(A);
        return BAD_ALLOC;
    }

    *nrm = 0;
    for (int i = 0; i < vec->n; ++i) {
        *nrm += mult_res[i] * vec->elems[i];
    }

    for (int i = 0; i < vec->n; ++i) {
        free(A[i]);
    }
    free(A);
    free(mult_res);
    *nrm = sqrt(*nrm);
    return OK;
}
```

Каждая из функций принимает в качестве параметра вектор, норму которого требуется посчитать, а также записывает найденную норму в параметр nrm.

Эти функции передаются в функцию с переменным числом аргументов longest_norm, которая вычисляет вектор с самой длинной нормой из

переданных. Эта функция имеет параметры res_1, res_2 и res_3 в каждый из которых записывает вектор с самой длинной нормой для каждой функции, если таких векторов несколько, заносит все.

Задание 3.

На вход программе через аргументы командной строки подается путь ко входному файлу, флаг (флаг начинается с символа ‘-’ или ‘/’, второй символ - ‘a’ или ‘d’) и путь к выходному файлу. В файле в каждой строчке содержится информация о сотруднике (для этой информации определите тип структуры Employee): id (целое неотрицательное число), имя (непустая строка только из букв латинского алфавита), фамилия (непустая строка только из букв латинского алфавита), заработная плата (неотрицательное вещественное число). Программа должна считать записи из файла в динамический массив структур и в выходной файл вывести данные, отсортированные (с флагом ‘-a’/‘/a’ - по возрастанию, с флагом ‘-d’/‘/d’ - по убыванию) первично - по зарплате, далее (если зарплаты равны) - по фамилии, далее (если зарплаты и фамилии равны) - по именам, наконец, по id. Для сортировки коллекции экземпляров структур используйте стандартную функцию qsort, своя реализация каких-либо алгоритмов сортировки не допускается.

Листинг 68. Структуры Employee и Employee_vec; список их методов и связанные функции.

```
typedef struct _employee {
  unsigned int id;
  char name[BUFSIZ];
  char surname[BUFSIZ];
  double salary;
} Employee;

typedef struct {
  Employee** empv;
  size_t capacity;
  size_t size;
} Employee_vec;

int empv_constr(Employee_vec* empv);
int empv_destr(Employee_vec* vec);
```

```
void show_employee(Employee const* emp);
enum get_employee_st get_employee(FILE* stream, Employee**
res);
enum get_employee_st get_info_from_file(Employee_vec*
empv, FILE* stream);
int print_emp_to_file(FILE* stream, Employee const* emp);
int print_empv_to_file(FILE* stream, Employee_vec const*
empv);

int comp_asc(void const* emp1, void const* emp2);
int comp_desc(void const* emp1, void const* emp2);

void empv_sort(Employee_vec* empv, enum sort_flag flag);
```

В функции handle выполняется основная логика приложения (ввод данных из файла, идентификация флага и вывод данных)

Листинг 69. Функция сортировки а также компараторы (восходящий и нисходящий).

```
int comp_asc(void const* emp1, void const* emp2) {
  Employee* e1 = *(Employee**) emp1;
  Employee* e2 = *(Employee**) emp2;

  if (e1->salary != e2->salary) {
    return e1->salary > e2->salary;
  }
  if (strcmp(e1->surname, e2->surname) != 0) {
    return e1->surname > e2->surname;
  }
  if (strcmp(e1->name, e2->name) != 0) {
    return e1->name > e2->name;
  }
  return e1->salary < e2->id;
}

int comp_desc(void const* emp1, void const* emp2) {
  return comp_asc(emp2, emp1);
}
```

```
void empv_sort(Employee_vec* db, enum sort_flag flag) {
  if (flag == ASCEND) {
    qsort(db->empv,      db->size,      sizeof(Employee*),
comp_asc);
  }

  if (flag == DESCEND) {
    qsort(db->empv,      db->size,      sizeof(Employee*),
comp_desc);
  }
}
```

Задание 4.

1. Опишите тип структуры String, содержащую в себе поля для указателя на динамический массив символов типа char и количества символов (длины строки) типа int. Для описанного типа структуры реализуйте функции:

● создания экземпляра типа String на основе значения типа char *

● удаления внутреннего содержимого экземпляра типа String

● отношения порядка между двумя экземплярами типа String (первично по длине строки, вторично по лексографическому компаратору)

● отношения эквивалентности между двумя экземплярами типа String (лексографический компаратор)

● копирования содержимого экземпляра типа String в существующий экземпляр типа String

● копирования содержимого экземпляра типа String в новый экземпляр типа String, размещённый в динамической памяти

● конкатенации к содержимому первого экземпляра типа String содержимого второго экземпляра типа String.

Продемонстрируйте работу реализованного функционала.

2. Экземпляр структуры Mail содержит в себе экземпляр структуры Address получателя (город (непустая строка), улица (непустая строка), номер дома (натуральное число), корпус (строка), номер квартиры (натуральное число), индекс получателя (строка из шести символов цифр)), вес посылки (неотрицательное вещественное число), почтовый идентификатор (строка из 14 символов цифр), время создания (строка в формате “dd:MM:yyyy hh:mm:ss”), время вручения (строка в формате “dd:MM:yyyy hh:mm:ss”). Экземпляр структуры Post содержит указатель

на экземпляр структуры Address текущего почтового отделения и динамический массив экземпляров структур типа Mail. Реализуйте интерактивный диалог с пользователем, предоставляющий функционал для добавления и удаления объектов структур типа Mail в объект структуры типа Post, информативный вывод данных об отправлении при поиске объекта типа Mail по идентификатору. Объекты структуры Mail должны быть отсортированы по индексу получателя (первично) и идентификатору посылки (вторично) в произвольный момент времени. Также в интерактивном диалоге реализуйте опции поиска всех доставленных отправлений, а также всех отправлений, срок доставки которых на текущий момент времени (системное время) истёк. Информацию о доставленных/недоставленных отправлениях выводите в порядке времени создания по возрастанию (от старых к новым). Для хранения строковых данных используйте структуру String из п. 1.

Структура String имеет расширенный функционал в сравнение с требованиями, так как реализованный функционал оказался приемлемым, он использовался в последующих заданиях и получал новые расширения.

Листинг 70. Структура String и список реализованных методов.

```
typedef struct _string {
  char* _buf;
  size_t _size;
  size_t _cap; // Terminating 0 is excluded
} String;

int string_resize(String* str, size_t n);


int string_init(String* str, size_t n);
int string_constr(char const* src, String* dest);
int string_destr(String* str);

int destr_strings(int cnt, ...);
int init_strings(int cnt, int n, ...);
int free_strings(int cnt, ...);

int string_clear(String* str);
void show_string(String const* str);
```

```
void str_fprint(String const* str, FILE* stream);
get_str_st getline(String* str, FILE* stream);
get_str_st get_lexema_or_empty(String* str, FILE* stream);
int string_is_empty(String const* str);

int string_is_equal(String const* str1, String const*
str2);
int str_is_equal_charp(String const* str1, char const*
str2);
int str1_cmp_str2(String const* str1, String const* str2);
int string_copy(String const* src, String* dest);
```

Листинг 71. Структуры второго пункта задания.

```
typedef struct _address {
  String city;
  String street;
  unsigned int house_n;
  String building;
  unsigned int apt_n;
  String index; // 6 chars
} Address;

typedef struct _mail {
  Address recieve_addr;
  double weight;
  String mail_id; // 14 chars
  String creation_time; // "dd:MM:yyyy hh:mm:ss"
  String recieve_time;
} Mail;

typedef struct node {
  Mail* data;
  struct node* left;
  struct node* right;
} mail_bst_node;

typedef struct {
  mail_bst_node* root;
```

```
  int (* comp)(Mail*, Mail*);
} MailBST;

typedef struct _post {
  Address* post_addr;
  MailBST* mails;
} Post;
```

Для хранения писем была выбрана структура бинарное дерево поиска, так как при добавлении элементов, и их поиске оно имеет оптимальную сложность (в среднем O(logN)) и не является трудным в реализации. Если , например, письма хранились бы в массиве, сложность составляла бы O(N).

Функция execute вызывает диалог с пользователем, принимая поток ввода как параметр (для удобства отладки). В процессе диалога у пользователя просят ввести данные Post, а также ему предлагается выбрать одну из опций.

Листинг 72. Функция execute.

```
int execute(FILE* in) {
  printf("It all starts here\n");
  Post post;
  Address* post_address = (Address*)
malloc(sizeof(Address));
  MailBST* bst = (MailBST*) malloc(sizeof(MailBST));
  post_constr(&post, post_address, bst);
  if (!post_address || !bst) {
    post_destruct(&post);
    return -1;
  }
  bst_constr(bst, comp);

  printf("Lets get address of the Post\n");
  if (get_address(post_address, in) != get_rv_ok) {
    free(post_address);
    return -1;
  }

  String temp_str;
```

```
  string_init(&temp_str, 16);

  command cmd = cm_help_msg;
  cmd = command_execute(cmd, &post, in);
  while (cmd != cm_exit && cmd != cm_eof) {
    int code = getline(&temp_str, in);
    if (code == get_str_ok) {
      cmd = get_command(&temp_str);
      cmd = command_execute(cmd, &post, in);
    } else if (code != get_str_empty) {
      post_destruct(&post);
      return -1;
    }
  }

  post_destruct(&post);
  return 0;
}
```

Задание 5.

Экземпляр структуры типа Student содержит поля: id студента (целое неотрицательное число), имя (непустая строка только из букв латинского алфавита), фамилия (непустая строка только из букв латинского алфавита), группа (непустая строка) и оценки за 5 экзаменов (динамический массив элементов типа unsigned char). Через аргументы командной строки программе на вход подаётся путь к файлу, содержащему записи о студентах. При старте программа считывает поданный файл в динамический массив структур типа Student. В программе должен быть реализован поиск всех студентов по:

- id;
- фамилии;
- имени;
- группе,

сортировка (для сортировки необходимо передавать компаратор для объектов структур) студента(-ов) по:

- id;

● фамилии;

● имени;

● группе.

Добавьте возможность вывода в трассировочный файл (путь к файлу передаётся как аргумент командной строки) вывести данные найденного по id студента: ФИО, группу и среднюю оценку за экзамены. Также добавьте возможность вывести в трассировочный файл фамилии и имена студентов, чей средний балл за все экзамены выше среднего балла за все экзамены по всем считанным из файла студентам. Все вышеописанные опции должны быть выполнимы из контекста интерактивного диалога с пользователем. Для сортировки коллекции экземпляров структур используйте стандартную функцию qsort, своя реализация каких-либо алгоритмов сортировки не допускается.

Для поиска и сортировки определен перечислимые тип sf_flag, каждому значению типа соответствует один компаратор.

Листинг 73. enum sf_flag и компараторы

```
typedef enum {
  sf_id,
  sf_name,
  sf_surname,
  sf_group,
  sf_unknown
} sf_flag;
int name_cmp(const void* s1, const void* s2);
int id_cmp(const void* s1, const void* s2);
int surname_cmp(const void* s1, const void* s2);
int group_cmp(const void* s1, const void* s2);
```

Листинг 74. Структуры Student и StudentVec.

```
typedef struct {
  unsigned int id;
  char name[BUFFSIZE + 1];
  char surname[BUFFSIZE + 1];
  char group[BUFFSIZE + 1];
  unsigned char* grades;
} Student;
```

```
typedef struct {
  Student* arr;
  size_t size;
  size_t cap;
} StudentVec;
```

Данные в StudentVec никак не упорядочены, поэтому поиск всегда будет за O(N), а сортировка в среднем O(N logN).
Макросы BUFFSIZE и GRADE_CNT определяют соответственно размер буфферов под имя и фамилию студента и количество оценок студента.

Функция start вызывает диалог с пользователем, в ходе диалога предлагается отсортировать массив, вывести данные в заранее определенный трассировочный файл или найти студента (если студент найден также предлагается вывести его данные в файл).

Листинг 75. Функция start

```
st_code start(int argc, char** argv) {
  if (argc != 3) {
    return INV_ARGC;
  }
  FILE* in = fopen(argv[1], "r");
  if (!in) {
    return INV_FILE;
  }
  StudentVec s_vec;
  if (st_vec_init(&s_vec, 1) != 0) {
    fclose(in);
    return BAD_ALLOC;
  }
  switch (st_vec_get_from_file(&s_vec, in)) {
    case get_bad_alloc:
      st_vec_dest(&s_vec);
      fclose(in);
      return BAD_ALLOC;
    case get_inv_input:
      fclose(in);
      st_vec_dest(&s_vec);
```

```
      return INV_INPUT;
  }
  fclose(in);
  FILE* out = fopen(argv[2], "w");
  if (!out) {
    return INV_FILE;
  }
  echo_help();
  user_cmd cmd = SORT;
  String temp;
  if (string_init(&temp, 10) != 0) {
    st_vec_dest(&s_vec);
    fclose(out);
    return BAD_ALLOC;
  }
  while (!feof(stdin) && cmd != EXIT) {
    switch (getline(&temp, stdin)) {
      case get_str_bad_alloc:
        st_vec_dest(&s_vec);
        string_destr(&temp);
        fclose(out);
        return BAD_ALLOC;
      case get_str_eof:
        st_vec_dest(&s_vec);
        string_destr(&temp);
        fclose(out);
        return OK;
    }
    cmd = get_cmd(temp._buf);
    cmd_exec(&s_vec, out, cmd);
  }

  st_vec_dest(&s_vec);
  string_destr(&temp);
  fclose(out);
  return OK;
}
```

Задание 10.

Реализуйте приложение для построения дерева скобочного выражения. На вход программе через аргументы командной строки подается путь к файлу, в котором содержится произвольное число строк. Каждая строка является корректным скобочным выражением. Ваша программа должна обработать каждое скобочное выражение из файла и вывести в текстовый файл (путь к файлу - аргумент командной строки) деревья скобочных записей в наглядной форме (форму вывода определите самостоятельно). Возникающие при работе программы деревья не обязаны быть бинарными. Например, запись A (B (E (G, T, R (W, Z)), F (L, M)), C) соответствует дереву

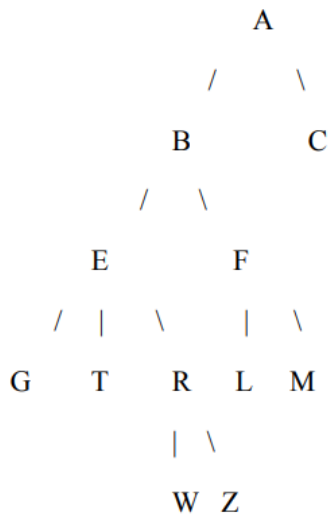

Рисунок 5. Задание 10.

Формат вывода не фиксирован и определяется удобством восприятия.

Определим структуры ноды дерева и самого дерева

Листинг 76. tree_node и Tree.

```
typedef struct tree_node {
  char value;
  struct tree_node* parent;
  struct tree_node* brother;
  struct tree_node* child;
} tree_node;

typedef struct {
  tree_node* root;
} Tree;
```

Определим функцию, которая создает дерево из строки, описывающей его.

Листинг 77. Функция tree_create.

```
int tree_create(Tree* _tree, String* _string) {
  if (_tree == NULL) {
    return -1;
  }
  if (_string == NULL) {
    return -2;
  }
  if (string_is_empty(_string)) {
    return -3;
  }
  char* ptr = _string->_buf;
  size_t size = _string->_size;
  tree_node* cur_node;
  bool if_next_child = false;
  bool if_first = true;
  for (int i = 0; i < size; i++) {
    switch (ptr[i]) {
      case ' ':
      case ',':
        break;
      case '(':
        if (if_next_child) {
          return -6;
        }
        if_next_child = true;
        break;
      case ')':
        if(cur_node->parent == NULL){
          return -7;
        }
        cur_node = cur_node->parent;
        break;
      default: {
        tree_node* new_node;
```

```
        if ((new_node = alloc_node(ptr[i])) == NULL) {
          return 1;
        }
        if (if_first) {
          _tree->root = new_node;
          cur_node = new_node;
          if_first = false;
          break;
        }
        if (if_next_child) {
          if (add_child(cur_node, new_node) != 0) {
            return -4;
          }
          cur_node = cur_node->child;
          if_next_child = false;
        } else {
          if (add_brother(cur_node, new_node)) {
            return -5;
          }
          cur_node = cur_node->brother;
        }
      }
    }
  }
  return 0;
}
```

Определим также функцию execute, которая будет обрабатывать входные данные и считывать построчно информацию о дереве из входного файла.

Листинг 78. Функция execute.

```
st_code execute(int argc, char* argv[]) {
  if (argc != 3) {
    return argc_invalid;
  }

  FILE* in = fopen(argv[1], "r");
  if (!in) {
```

```
  return file_invalid;
}
FILE* out = fopen(argv[2], "w");
if (!out) {
  fclose(in);
  return file_invalid;
}
String str;
if (string_init(&str, 10) != 0) {
  fclose(in);
  fclose(out);
  return bad_alloc;
}
Tree tree;
while (!feof(in)) {
  int code = getline(&str, in);
  if (code != get_str_ok) {
    fclose(in);
    fclose(out);
    if (code == get_str_bad_alloc) {
      return bad_alloc;
    }
    return input_invalid;
  }
  code = tree_create(&tree, &str);
  if (code != 0) {
    tree_destroy(&tree);
    fclose(in);
    fclose(out);
    if (code > 0) {
      return bad_alloc;
    }
    return input_invalid;
  }
  str_fprint(&str, out);
  fputc('\n', out);
  print_tree(&tree, out);
  tree_destroy(&tree);
```

```
  }
  fclose(in);
  fclose(out);
  return ok;
}
```

Лабораторная работа №4.

Задание 5.

На вход приложению через аргументы командной строки подаются пути к текстовым файлам, содержащим арифметические выражения (в каждой строке находится одно выражение). Выражения в файлах могут быть произвольной структуры: содержать произвольное количество арифметических операций (сложение, вычитание, умножение, целочисленное деление, взятие остатка от деления, возведение в целую неотрицательную степень), круглых скобок (задают приоритет вычисления подвыражений). В вашем приложении необходимо для каждого выражения из каждого входного файла:

● проверить баланс скобок;

● построить обратную польскую запись выражения;

● вычислить значение выражения с использованием алгоритма вычисления выражения, записанного в обратной польской записи.

В результате работы приложения для каждого файла необходимо вывести в стандартный поток вывода путь к файлу, а также для каждого выражения из этого файла необходимо вывести:

● исходное выражение;

● обратную польскую запись для исходного выражения;

● значение выражения.

В случае обнаружения ошибки в расстановке скобок либо невозможности вычислить значение выражения, для каждого файла, где обнаружены вышеописанные ситуации, необходимо создать текстовый файл, в который выписать для каждого ошибочного выражения из исходного файла: само выражение, его порядковый номер в файле (индексация с 0) и причину невозможности вычисления. Для решения задачи используйте собственную реализацию структуры данных вида стек на базе структуры данных вида односвязный список. Предоставьте текстовый файл с выражениями для реализованного приложения и продемонстрируйте его работу. Обработайте ошибки времени выполнения инструкций.

Для начала реализуем стек.

Листинг 79. Нода стека и стек.

```
typedef struct stack_node {
    struct stack_node* next;
    char* value;
```

```
} stack_node;

typedef struct {
    stack_node* top;
} stack;

char* stack_pop(stack* st);
char* stack_top(stack* st);
int stack_push(stack* st, char* value);
int stack_destr(stack* st);
```

Стек не «владеет» данными которые лежат под указателем char* value, он просто ссылается на выделенную извне для другого места память, поэтому при добавлении строки на стек не происходит ее копирование, а также память не приходится очищать. Такой подход дает чуть меньше контроля, и взамен дает куда более высокую скорость и меньший объем занимаемой памяти. Если стек пуст, stack_top и stack_pop возвращают NULL.

Переходя к алгоритму, для удобства и для дальнейшего оперирования изначальное выражение бьется на отдельные части (операции, операнды, скобки) в функции split_expr. Далее эти данные можно пропустить через функцию convert_to_reversed, чтобы получить выражение в постфиксной форме, и уже у этой формы можно посчитать значение пропустив через функцию compute_exp. В функции же compute_file происходит обработка данных из файла (построчное считывание и вычисление значения при помощи функций выше).

Задание 6.

Реализуйте приложение, которое по заданной булевой формуле строит таблицу истинности, описывающую логическую функцию, заданную формулой. Через аргументы командной строки приложению подается путь к файлу, который содержит одну строку, в которой записана булева формула. В этой формуле могут присутствовать:
- односимвольные имена логических переменных;
- константы 0 (ложь) и 1 (истина);
- & - оператор логической конъюнкции;
- | - оператор логической дизъюнкции;
- ~ - оператор логической инверсии;

- -> - оператор логической импликации;
- +> - оператор логической коимпликации;
- <> - оператор логического сложения по модулю 2;
- = - оператор логической эквиваленции;
- ! - оператор логического штриха Шеффера;
- ? - оператор логической функции Вебба;
- операторы круглых скобок, задающие приоритет вычисления подвыражений. Вложенность скобок произвольна. Для вычисления булевой формулы постройте бинарное дерево выражения и вычисление значения булевой формулы на конкретном наборе переменных выполняйте с помощью этого дерева. Приоритеты операторов (от высшего к низшему): 3(~), 2(?,!,+>,&), 1(|, ->, <>, =). В результате работы приложения необходимо получить выходной файл (имя файла псевдослучайно составляется из букв латинского алфавита и символов арабских цифр, файл должен быть размещён в одном каталоге с исходным файлом) с таблицей истинности булевой функции, заданной входной булевой формулой.

Реализуем структуру дерева. Главная задача стоит в переборе значений для всех неконстантных операндов ради получения таблицы истинности. Реализовав дерево самым простым способом придется перебирать все ноды вручную, что, во-первых не очень эффективно, а во-вторых не совсем понятно как именно. Поэтому я предлагаю следующее решение:
Помимо обычных для дерева значения и ссылок на левое и правое поддерево будем хранить в каждой ноде ссылку на некий элемент таблицы, в которой будут храниться все встреченные операнды, если в ноде лежит оператор или константа указатель равен NULL. Для простоты предлагаю хранить таблицу в виде обычного массива пар имя-значение, причем значение будет задаваться только во время перебора.

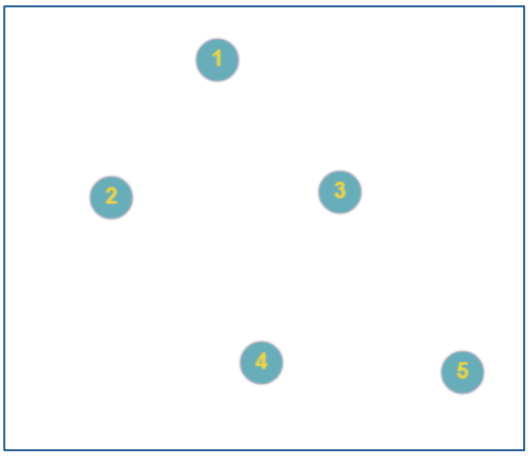

Рисунок 6. Пример структуры дерева

Например, если на месте нод 2 и 4 лежат операнды A, то они будут ссылаться на одно место в таблице и при переборе брать одно и то же значение из таблицы.

Листинг 80. Структура таблицы с операндами.
```
typedef struct {
    char* name;
    int value;
} LNode;
typedef struct {
    LNode* arr;
    int cap;
    int size;
} VarList;
```

Листинг 81. Структура дерева.
```
typedef struct TNode {
    char* value;
    LNode* cell;
    struct TNode* left;
    struct TNode* right;
} TNode;

typedef struct {
    TNode* root;
    VarList vars;
} Tree;
```

В классической реализации алгоритма используются два разных стека, стека операндов (хранятся ноды деревьев) и стек операций (строка или символ). Но так как в C нет ни шаблонов, ни дженериков реализовывать один и тот же функционал дважды не является лучшим решением. Можно использовать следующую идею — хранить на стеке операций такие же ноды деревьев, только всегда левое и правое поддеревья у них при создании будут пустые. Таким образом можно сделать лишь одну реализацию стека.

Листинг 82. Реализация стека.
```
typedef struct stack_node {
    TNode* value;
```

```
    struct stack_node* next;
} stack_node;
typedef struct {
    stack_node* top;
} stack;
TNode* stack_pop(stack* st);
TNode* stack_top(stack* st);
int stack_push(stack* st, TNode* value);
int stack_destr(stack* st);
```

Функция build_tree строит дерево выражения и вызывает функцию create_value_table, которая связывает каждую ноду-переменную с ее местом в таблице.

Функция print_table проходит по всем возможным перестановкам значений переменных, вычисляет их значения и выводит таблицу истинности в файл. Функция evaluate_expression делает подсчет значения. Важно, что обязательным условием работы является заполнение значений в таблице для каждой переменной (вручную или вызовом функции print_table), иначе поведение не определено.

Функция get_rand_name возвращает псевдослучайно сгенерированную строку из count символ (без учета нуль-терминатора).

Задание 7.

Опишите тип структуры MemoryCell, содержащей имя переменной и её целочисленное значение.
Через аргументы командной строки в программу подается файл с инструкциями вида

myvar=15;
bg=25;
ccc=bg+11;
print ccc;
myvar=ccc;
bg=ccc*myvar;
print;
Файл не содержит ошибок и все инструкции корректны. В рамках приложения реализуйте чтение данных из файла и выполнение всех

простых арифметических операций (сложение, вычитание, умножение, целочисленное деление, взятие остатка от деления), инициализации переменной и присваивания значения переменной (=) и операции print (вывод в стандартный поток вывода либо значения переменной, имя которой является параметром операции print, либо значений всех объявленных на текущий момент выполнения переменных с указанием их имён). В каждой инструкции может присутствовать только одна из вышеописанных операций; аргументами инструкций могут выступать значение переменной, подаваемое в виде имени этой переменной, а также целочисленные константы, записанные в системе счисления с основанием 10. При инициализации переменной по необходимости требуется довыделить память в динамическом массиве структур типа MemoryCell; инициализация переменной (по отношению к операции перевыделения памяти) должна выполняться за амортизированную константу. Для поиска переменной в массиве используйте алгоритм дихотомического поиска, для этого ваш массив в произвольный момент времени должен находиться в отсортированном состоянии по ключу имени переменной; для сортировки массива используйте стандартную функцию qsort, реализовывать непосредственно какие-либо алгоритмы сортировки запрещается. Имя переменной может иметь произвольную длину и содержать только символы латинских букв, прописные и строчные буквы при этом не отождествляются. В случае использования в вычислениях не объявленной переменной необходимо остановить работу интерпретатора и вывести сообщение об ошибке в стандартный поток вывода. Предоставьте текстовый файл с инструкциями для реализованного интерпретатора и продемонстрируйте его работу. Обработайте ошибки времени выполнения инструкций.

Из функции main вызывается функция parse_file, которая построчно обрабатывает входной файл.

Листинг 83. Функция parse_file

```
parse_rv parse_file(FILE* in, FILE* out) {
  MemoryVec mvec = {0};
  String str = {0};
  StringVec parsed_vec = {0};
  if (mem_vec_constr(&mvec, 1) != 0) {
    return parse_rv_bad_alloc;
  }
```

```
  if (string_init(&str, 16) != 0) {
    mem_vec_destr(&mvec);
    return parse_rv_bad_alloc;
  }
  if (str_vec_init(&parsed_vec, 5) != 0) {
    mem_vec_destr(&mvec);
    string_destr(&str);
    return parse_rv_bad_alloc;
  }

  while (!feof(in)) {
    switch (getline(&str, in)) {
      case get_str_bad_alloc: {
        mem_vec_destr(&mvec);
        string_destr(&str);
        str_vec_dest(&parsed_vec);
        return parse_rv_bad_alloc;
      }
      case get_str_eof:
      case get_str_empty:
        break;

      default: {
        parse_rv s_code = parse_string(&str, &parsed_vec);
        if (s_code) {
          mem_vec_destr(&mvec);
          string_destr(&str);
          str_vec_dest(&parsed_vec);
          return s_code;
        }

        s_code = parse_expression(&parsed_vec, &mvec,
out);
        if (s_code != parse_rv_ok) {
          mem_vec_destr(&mvec);
          str_vec_dest(&parsed_vec);
          string_destr(&str);
          return s_code;
```

```
        }
        str_vec_clear(&parsed_vec);
      }
    }
  }

  mem_vec_destr(&mvec);
  str_vec_dest(&parsed_vec);
  string_destr(&str);
  return 0;
}
```

Внутри этой функции вызывается функция parse_string, которая разбивает строку на раздельные части и заносит их в StringVec.

Листинг 84. структура StringVec.

```
#include "../../Lab-3/lab3-4/my_string.h"

typedef struct {
  String* buf;
  size_t cap;
  size_t cnt;
} StringVec;
```

Далее эти данные попадают в функцию parse_expression. В ней происходит исполнение команды print и создания/изменения значения переменной. Важно отметить, что запись var = var; инициализирует переменную var со значением 0.

Все ячейки MemoryCell хранятся в массиве MemoryVec, после каждого добавления нового элемента в этот массив он сортируется, что обеспечивает поиск за время O(N). Все ячейки сортируются по имени переменной, которая хранится в ней.

Вывод.

В ходе курсовой работы были изучены основы языка Си, его типы, стандартная библиотека языка. На практике применен функционал для работы с функциями с переменным числом аргументов, а также реализованы различные алгоритмы и структуры данных.

Список использованных источников.

1. Лекционный материал курса математический практикум.
2. Харви Дейтел, Пол Дейтел. Как программировать на С. / Электронное издание.

Приложение к работе.

Репозиторий на GitHub - MotyaSS/labs_3sem (github.com)