КПІ ім. Ігоря Сікорського

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт до комп'ютерного практикуму з курсу «Основи програмування»

Прийняв
асистент кафедри ІПІ
Ахаладзе А. Е.
«13» грудня 2024 р.

Виконав
студент групи ІП-43
Дутов І. А.

Київ 2024

Комп'ютерний практикум №6

Тема: *Використання динамічних масивів.*

Завдання:

Написати програму розв'язання системи лінійних алгебраїчних рівнянь (СЛАР) методом простої ітерації з використанням динамічних масивів.

Текст програми

../src/main.c

```c
#include <math.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

#include "algorithm.h"
#include "common/constants.h"
#include "common/types.h"
#include "io/double.h"
#include "io/equations.h"
#include "io/int.h"
#include "io/utils.h"
#include "io/validators.h"

const IntRange EQUATION_COUNT_RANGE = {2, 10000, true, true};
const DoubleRange COEF_RANGE = {-1e12, 1e12, true, true};
const DoubleRange PRECISION_RANGE = {DBL_EPSILON, 1.0, true, false};

void print_demo();

int main() {
  print_demo();
  do {
    Equations eqs;
    eqs.count = read_int_within_range_with_validation(
        "Розмір СЛАР", &EQUATION_COUNT_RANGE, NULL);
    if (initialize_equations(&eqs) == FAILURE) {
      return FAILURE;
    }

    DoubleRange user_coef_range;
    user_coef_range.min = read_double_within_range_with_validation(
        "Мінімальне значення коефіцієнтів", &COEF_RANGE, NULL);

    user_coef_range.max = read_double_within_range_with_validation(
        "Максимальне значення коефієнтів", &COEF_RANGE, is_greater,
        user_coef_range.min);
    user_coef_range.is_max_included = user_coef_range.is_min_included = true;

    double precision = read_double_within_range_with_validation(
        "Точність", &PRECISION_RANGE, NULL);

    int decimal_places = (int)(-log10(precision));

    if (ask_question("Чи бажаєте Ви уввімкнути випадковий режим?")) {
      generate_equations(&eqs, &user_coef_range);
      print_equations(&eqs, decimal_places);
    } else {
```

```
    handle_equation_input(&eqs, &user_coef_range, decimal_places);
  }

  Solutions sol;
  sol.count = eqs.count;

  if (initialize_solutions(&sol) == FAILURE) {
    return FAILURE;
  }

  if (solve_equations(&eqs, &sol, precision) == SUCCESS) {
    print_solutions(&sol, decimal_places);
  } else {
    handle_error("Ця система не є збіжною для методу!");
  };

  free_program(&eqs, &sol);

 } while (ask_question("Чи хочете ви повторити?"));

 return 0;
}

void print_demo() {
 printf(
    "\Вас вітає програма NumberWorld, що вирішує задану \n систему "
    "алгебраїчних рівнянь СЛАР() методом ГауссаЗейделя- із заданою "
    "точністю\n");
 show_warning("Наші обмеження такі:");
 print_int_range("кількість рівнянь у системі", &EQUATION_COUNT_RANGE);
 print_double_range(
    "мінімальне та максимальне значення коефіцієнтів у рівняннях",
    &COEF_RANGE);
 printf(
    "мінімальне значення коефіцієнтів < максимальне значення коефіцієнтів\n");
 print_double_range("точність", &PRECISION_RANGE);

 print_double_character_count_range();
 print_int_character_count_range();
 print_double_precision_limit();

 printf("\n");
 show_warning(
    "Алгоритм ГауссаЗейделя- вимагає, щоб матриця була діагонально "
    "домінантною, тобто:");
 printf(
    "Модуль будьякого- діагонального елемента \n має бути більшим за суму "
    "модулів елементів того самого рядка\n");
 printf("Якщо Ви не дотримаєте цього правила, отримаєте помилку.\n");
 show_warning(
    "Випадкова генерація хоч і діє в проміжку, що зазначаєте Ви, \протен "
    "діагональні елементи підбираються для сходимості алгоритму.");
 printf("\n");
}
```

../src/common/constants.h

```
#ifndef CONSTANTS_H
#define CONSTANTS_H
```

```

#define SUCCESS 0
#define FAILURE 1

#define RED "\033[31m"
#define GREEN "\033[32m"
#define YELLOW "\033[33m"
#define RESET "\033[0m"

#endif
```

../src/common/types.h

```
#ifndef TYPES_H
#define TYPES_H

#include <stdarg.h>
#include <stdbool.h>

typedef enum {
  LESS,
  LESS_EQUAL,
  GREATER,
  GREATER_EQUAL,
  WITHIN_RANGE
} RangeCheckResult;

typedef struct {
  double min, max;
  bool is_min_included, is_max_included;
} DoubleRange;

typedef struct {
  int min, max;
  bool is_min_included, is_max_included;
} IntRange;

typedef bool (*DoubleValidation)(double value, va_list args);
typedef bool (*IntValidation)(int value, va_list args);

#endif
```

../src/algorithm.h

```
#ifndef ALGORITHM_H
#define ALGORITHM_H

#include <float.h>
#include <stdbool.h>

#include "common/types.h"

typedef struct {
  int count;
  double** coefs;
  double* const_coefs;
} Equations;
```

```c

typedef struct {
  int count;
  double* corrs; // corrections
  double* roots;
} Solutions;

int initialize_equations(Equations* eqs);
int initialize_solutions(Solutions* sol);
void free_program(Equations* eqs, Solutions* sol);

void generate_equations(Equations* eqs, DoubleRange* coef_range);

bool is_convergent_row(Equations* eqs, int row);

int solve_equations(Equations* eqs, Solutions* sol, double precision);

#endif
```

../src/algorithm.c

```c
#include "algorithm.h"

#include <math.h>
#include <stdbool.h>
#include <stdlib.h>
#include <time.h>

#include "common/constants.h"
#include "io/utils.h"

const DoubleRange DIAGONAL_RANGE = {0.0, 100.0};

void initialize_generator() { srand(time(NULL)); }

int initialize_equations(Equations* eqs) {
  eqs->coefs = (double**)malloc(eqs->count * sizeof(double*));

  if (eqs->coefs == NULL) {
    free(eqs->coefs);
    handle_error_memory_allocation("вказівників на коефіцієнти рівнянь");
    return FAILURE;
  };

  for (int i = 0; i < eqs->count; i++) {
    eqs->coefs[i] = (double*)malloc(eqs->count * sizeof(double));
    if (eqs->coefs[i] == NULL) {
      for (int j = i; j >= 0; j--) {
        free(eqs->coefs[j]);
      }
      free(eqs->coefs);
      handle_error_memory_allocation("коефіцієнтів рівнянь");
    }
  }

  eqs->const_coefs = (double*)malloc(eqs->count * sizeof(double));
  if (eqs->const_coefs == NULL) {
    free(eqs->const_coefs);
    free(eqs->coefs);
```

```
    handle_error_memory_allocation("вільних членів рівнянь");
    return FAILURE;
  }
  return SUCCESS;
}

int initialize_solutions(Solutions* sol) {
  sol->roots = (double*)malloc(sol->count * sizeof(double));
  if (sol->roots == NULL) {
    free(sol->roots);
    handle_error_memory_allocation("коренів рівнянь");
    return FAILURE;
  }

  sol->corrs = (double*)malloc(sol->count * sizeof(double));
  if (sol->corrs == NULL) {
    free(sol->roots);
    free(sol->corrs);
    handle_error_memory_allocation(
        "коррекцій коренів використовуються( всередині алгоритму)");
    return FAILURE;
  }
  return SUCCESS;
}

void free_program(Equations* eqs, Solutions* sol) {
  for (int i = 0; i < eqs->count; i++) {
    free(eqs->coefs[i]);
  }
  free(eqs->coefs);
  free(eqs->const_coefs);

  free(sol->roots);
  free(sol->corrs);
}

double generate_double(const DoubleRange* range) {
  double random_0_1 = (double)rand() / ((double)RAND_MAX + 1.0);
  return (range->min + (range->max - range->min) * random_0_1);
}

void generate_equations(Equations* eqs, DoubleRange* coef_range) {
  initialize_generator();

  for (int i = 0; i < eqs->count; i++) {
    double compound_sum = 0.0;

    for (int j = 0; j < eqs->count; j++) {
      if (j == i) continue;
      eqs->coefs[i][j] = fabs(generate_double(coef_range));
      compound_sum += eqs->coefs[i][j];
    }

    // There would be no overflow for our constraints
    double diagonal_item = compound_sum + generate_double(&DIAGONAL_RANGE);
    eqs->coefs[i][i] = diagonal_item;

    eqs->const_coefs[i] = generate_double(coef_range);
  }
}
```

```c

bool is_convergent_row(Equations* eqs, int row) {
  double non_diagonal_sum = 0;

  for (int i = 0; i < eqs->count; i++) {
    if (i == row) continue;
    non_diagonal_sum += fabs(eqs->coefs[row][i]);
  }

  if (fabs(eqs->coefs[row][row]) <= non_diagonal_sum) {
    return false;
  }
  return true;
}

bool is_convergent_equations(Equations* eqs) {
  for (int i = 0; i < eqs->count; i++) {
    if (!is_convergent_row(eqs, i)) {
      return false;
    }
  }
  return true;
}

int solve_equations(Equations* eqs, Solutions* sol, double precision) {
  // Ensure diagonal dominance for convergence
  if (!is_convergent_equations(eqs)) {
    return FAILURE;
  }

  for (int i = 0; i < eqs->count; i++) {
    sol->roots[i] = eqs->const_coefs[i] / eqs->coefs[i][i]; // Initial guess
  }

  double max_correction;
  do {
    max_correction = 0.0;
    for (int i = 0; i < eqs->count; i++) {
      double compound_sum = 0.0;

      for (int j = 0; j < eqs->count; j++) {
        if (j == i) continue;
        compound_sum += eqs->coefs[i][j] * sol->roots[j];
      }

      double new_value =
         (eqs->const_coefs[i] - compound_sum) / eqs->coefs[i][i];
      double delta = fabs(new_value - sol->roots[i]);

      if (delta > max_correction) {
        max_correction = delta;
      }

      sol->roots[i] = new_value;
    }
  } while (max_correction >= precision);

  return SUCCESS;
}
```

../src/io/equations.c

```c
#include "equations.h"

#include <stdbool.h>
#include <stdio.h>

#include "../algorithm.h"
#include "common/constants.h"
#include "double.h"
#include "utils.h"

#define COEFS_PER_ROW 5
#define ROOTS_PER_ROW 5
#define CLEAR_LINE_COUNT 1

void print_equation_row(double* row_coefs, int start, int end,
                        int decimal_places) {
  for (int j = start; j <= end; j++) {
    print_truncated_double(row_coefs[j], decimal_places);
    printf("x_%d", j + 1);
    if (j < end) {
      printf(" + ");
    }
  }
}

void print_equations(Equations* eqs, int decimal_places) {
  for (int i = 0; i < eqs->count; i++) {
    printf(GREEN "Рівняння №%d:" RESET "\n", i + 1); // Print equation header

    for (int j = 0; j < eqs->count; j += COEFS_PER_ROW) {
      bool is_last = (j + COEFS_PER_ROW) >= eqs->count;
      int row_end = is_last ? (eqs->count - 1) : (j + COEFS_PER_ROW - 1);

      if (j > 0) {
        printf("\t");
      }

      print_equation_row(eqs->coefs[i], j, row_end, decimal_places);

      if (!is_last) {
        printf(" +\n");
      }
    }

    printf(" = ");
    print_truncated_double(eqs->const_coefs[i], decimal_places);
    printf("\n");
  }
}

void print_solutions(Solutions* sol, int decimal_places) {
  show_success("Корені знайдено успішно!");
  for (int i = 0; i < sol->count; i++) {
    printf("x_%d = ", i + 1);
    print_truncated_double(sol->roots[i], decimal_places);

    if (i != sol->count - 1) {
      printf(", ");
```

```
    }

    if (i % ROOTS_PER_ROW == (ROOTS_PER_ROW - 1) &&
        sol->count > ROOTS_PER_ROW) {
      printf("\n");
    }
  }
}

void ask_for_coef(double* row_coefs, int coef_index, int decimal_places) {
  int last_row_start = coef_index - (coef_index % COEFS_PER_ROW);

  print_equation_row(row_coefs, last_row_start, coef_index - 1, decimal_places);

  if (coef_index != 0) {
    printf(" + ");
  }

  printf("? • x_%d ", coef_index + 1);
}

void handle_equation_input(Equations* eqs, DoubleRange* coef_range,
                           int decimal_places) {
  for (int i = 0; i < eqs->count; i++) {
    printf(GREEN "Введіть рівняння №%d:\n" RESET, i + 1);

    for (int coef_index = 0; coef_index < eqs->count; coef_index++) {
      ask_for_coef(eqs->coefs[i], coef_index, decimal_places);
      while (read_double_within_range(&eqs->coefs[i][coef_index], "",
                                      coef_range) == FAILURE) {
        ask_for_coef(eqs->coefs[i], coef_index, decimal_places);
      };

      bool is_row_end = (coef_index % COEFS_PER_ROW) == (COEFS_PER_ROW - 1);

      // clear_last_lines(CLEAR_LINE_COUNT); avoid clearing so that it can be
      // seen in report
      if (is_row_end) {
        int last_row_start = coef_index - (coef_index % COEFS_PER_ROW);
        print_equation_row(eqs->coefs[i], last_row_start, coef_index,
                           decimal_places);
        printf(" + \n");
      }
    }

    if (!is_convergent_row(eqs, i)) {
      int equation_line_count =
          (eqs->count + COEFS_PER_ROW - 1) / COEFS_PER_ROW;
      handle_error(
          "Введений Вами рядок коефіцієнтів не задовольняє умови для \n"
          "правильної роботи алгоритму ГауссаЗейделя-");
      clear_last_lines(equation_line_count + 1);
      i--;
      continue;
    }

    do {
      print_equation_row(eqs->coefs[i],
                         eqs->count - (eqs->count % COEFS_PER_ROW),
                         eqs->count - 1, decimal_places);
```

```
    printf("␣=␣?:␣");
  } while (read_double_within_range(&eqs->const_coefs[i], "", coef_range) ==
           FAILURE);

  // clear_last_lines(CLEAR_LINE_COUNT); avoid clearing so that it can be seen
  // in report

  print_equation_row(eqs->coefs[i], eqs->count - (eqs->count % COEFS_PER_ROW),
                     eqs->count - 1, decimal_places);
  printf("␣=␣");
  print_truncated_double(eqs->const_coefs[i], decimal_places);
  printf("\n");
 }
}
```

../src/io/equations.h

```
#ifndef EQUATIONS_H
#define EQUATIONS_H

#include "../algorithm.h"

void print_equations(Equations* eqs, int decimal_places);
void print_solutions(Solutions* sol, int decimal_places);
void handle_equation_input(Equations* eqs, DoubleRange* coef_range,
                           int decimal_places);

#endif // !EQUATIONS_H
```

../src/io/utils.h

```
#ifndef UTILS_H
#define UTILS_H

#include <stdbool.h>

// UTILS
void replace_commas_with_dots(char *string);

// OUTPUT
void clear_last_lines(int count);
void show_prompt(const char *prompt);
void handle_error(const char *format, ...);
void handle_error_overlength(int max_char_count);
void handle_error_memory_allocation(const char *reason);
void handle_error_overflow();
void handle_error_not_number();
void handle_error_not_precise(int max_significant_digits);
void show_warning(const char *format, ...);
void show_success(const char *format, ...);

// INPUT TAKING
void clear_stdin();
int read_input(char *input, int max_char_count);
bool ask_question(const char *format, ...);

// INPUT CHECKING
```

```c
bool is_input_within_length(const char *input);
bool is_input_floating_point(char *input);
bool is_input_number_after_conversion(const char *endptr, const char *input);

#endif
```

../src/io/utils.c

```c
#include "utils.h"

#include <stdarg.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "common/constants.h"

#define MOVE_UP "\033[F"
#define CLEAR_LINE "\033[2K"

#define ERROR_CLEAR_LINE_COUNT 4

// UTILS
void replace_commas_with_dots(char *string) {
  while (*string) {
    if (*string == ',') {
      *string = '.';
    }
    string++;
  }
}

// OUTPUT

void clear_last_lines(int count) {
  for (int i = 0; i < count; i++) {
    printf(MOVE_UP CLEAR_LINE);
  }
}

void show_prompt(const char *prompt) { printf("%s:", prompt); }

void continue_after_error() {
  printf("Натисніть Enter, щоб продовжити: ");
  fflush(stdout);
  clear_stdin();
  // clear_last_lines(ERROR_CLEAR_LINE_COUNT); avoid clearing so that it can be
  // seen in report
}

void handle_error(const char *format, ...) {
  va_list args;
  va_start(args, format);

  printf("\n" RED "ПОМИЛКА! ");
  vprintf(format, args);
  printf(RESET "\n");

```

```c
  va_end(args);
  continue_after_error();
}

void handle_error_overflow() {
  handle_error("Число␣поза␣допустимими␣межами␣значень␣типу!");
}

void handle_error_not_precise(int max_significant_digits) {
  handle_error(
      "Кількість␣значущих␣цифр␣перевищує␣максимальну␣дозволену␣кількість␣цифр␣"
      "%d,␣тому␣розрахунки␣стануть␣неточними!",
      max_significant_digits);
}

void handle_error_overlength(int max_char_count) {
  handle_error(
      "Довжина␣значення␣в␣символах␣має␣бути␣більше␣0␣та␣меншоюрівною-␣%d.",
      max_char_count);
}

void handle_error_memory_allocation(const char *reason) {
  handle_error("Не␣вдалося␣виділити␣достатньо␣памяті'␣для␣%s.", reason);
}

void handle_error_not_number() {
  handle_error("Значення␣має␣бути␣числом␣і␣не␣містити␣додаткових␣символів!");
}

void show_warning(const char *format, ...) {
  va_list args;
  va_start(args, format);

  printf("\n" YELLOW "УВАГА!␣");
  vprintf(format, args);
  printf(RESET "\n");

  va_end(args);
}

void show_success(const char *format, ...) {
  va_list args;
  va_start(args, format);

  printf("\n" GREEN "ПЕРЕМОГА!␣");
  vprintf(format, args);
  printf(RESET "\n");

  va_end(args);
}

// INPUT TAKING
bool is_agree() {
  char choice;

  printf(
      "Введіть␣+,␣якщо␣погоджуєтесь.␣Інакше␣введіть␣будьяку-␣"
      "іншу␣клавішу:␣");

  choice = getchar();
```

```
112   clear_stdin();
113
114   if (choice == '+') {
115     return true;
116   }
117
118   return false;
119 }
120
121 void clear_stdin() {
122   int c;
123   while ((c = getc(stdin)) != '\n' && c != EOF);
124 }
125
126 int read_input(char *input, int max_char_count) {
127   if (!fgets(input, max_char_count + 2, stdin)) {
128     handle_error("Не вдалося прочитати ввід.");
129     return EXIT_FAILURE;
130   }
131   return EXIT_SUCCESS;
132 }
133
134 bool ask_question(const char *format, ...) {
135   va_list args;
136   va_start(args, format);
137   printf("\n" YELLOW);
138   vprintf(format, args);
139   printf(RESET "\n");
140   va_end(args);
141
142   return is_agree();
143 }
144
145 // INPUT CHECKING
146 bool is_input_within_length(const char *input) {
147   // Last character should be '\n' if input is within length
148   return (input[strlen(input) - 1] == '\n' && strlen(input) - 1 != 0);
149 }
150
151 bool is_input_floating_point(char *input) {
152   while (*input) {
153     switch (*input) {
154       case '.':
155       case ',':
156       case 'e':
157       case 'E':
158         return true;
159       default:
160         break;
161     }
162     input++;
163   }
164   return false;
165 }
166
167 bool is_input_number_after_conversion(const char *endptr, const char *input) {
168   return (endptr != input && *endptr == '\n');
169 }
```

../src/io/int.h

```c
#ifndef INT_H
#define INT_H

#include "common/types.h"

void print_int_range(const char *name, const IntRange *range);
void print_int_character_count_range();
int read_int_within_range_with_validation(const char *prompt,
                                  const IntRange *range,
                                  IntValidation additional_check, ...);

#endif // !INT_H
```

../src/io/int.c

```c
#include <limits.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

#include "common/constants.h"
#include "common/types.h"
#include "utils.h"

#define MAX_VALUE (long)INT_MAX
#define MIN_VALUE (long)INT_MIN
#define MAX_CHAR_COUNT_INT 10 // TODO: make it a little bit more usable

const IntRange INT_CHAR_COUNT_RANGE = {1, MAX_CHAR_COUNT_INT, true, true};

void show_prompt_within_range(const char *prompt, const IntRange *range) {
  printf("%s від (%d до %d): ", prompt, range->min, range->max);
}

void show_range_error_int(RangeCheckResult result, const IntRange *range) {
  switch (result) {
    case LESS:
      handle_error("Значення має бути ≥ %d.", range->min);
      break;
    case LESS_EQUAL:
      handle_error("Значення має бути > %d.", range->min);
      break;
    case GREATER:
      handle_error("Значення має бути ≤ %d.", range->max);
      break;
    case GREATER_EQUAL:
      handle_error("Значення має бути < %d.", range->max);
      break;
    default:
      break;
  }
}

// Range Checking Functions
RangeCheckResult validate_range_int(int value, const IntRange *range) {
  if (range->is_min_included && value < range->min) return LESS;
  if (!range->is_min_included && value ⩽ range->min) return LESS_EQUAL;
```

```c
  if (range->is_max_included && value > range->max) return GREATER;
  if (!range->is_max_included && value ⩾ range->max) return GREATER_EQUAL;
  return WITHIN_RANGE;
}

void print_int_range(const char *name, const IntRange *range) {
  printf("%d", range->min);
  if (range->is_min_included) {
    printf(" ≤ ");
  } else {
    printf(" < ");
  }

  printf("%s", name);

  if (range->is_max_included) {
    printf(" ≤ ");
  } else {
    printf(" < ");
  }

  printf("%d\n", range->max);
}

void print_int_character_count_range() {
  print_int_range("кількість символів у цілих числах", &INT_CHAR_COUNT_RANGE);
}

RangeCheckResult validate_overflow_int(long int value) {
  if (value > MAX_VALUE) return GREATER;
  if (value < MIN_VALUE) return LESS;
  return WITHIN_RANGE;
}

int read_int_quiet(int *value, const char *prompt) {
  char input[MAX_CHAR_COUNT_INT + 2];

  if (read_input(input, MAX_CHAR_COUNT_INT) == FAILURE) {
    return FAILURE;
  }

  if (!is_input_within_length(input)) {
    handle_error_overlength(MAX_CHAR_COUNT_INT);
    clear_stdin();
    return FAILURE;
  }

  char *endptr;
  long int temp_value = strtol(input, &endptr, 10);

  if (!is_input_number_after_conversion(endptr, input)) {
    handle_error_not_number();
    return FAILURE;
  }

  RangeCheckResult global_check = validate_overflow_int(temp_value);
  if (global_check != WITHIN_RANGE) {
    handle_error_overflow();
    return FAILURE;
  }
```

```c

  *value = (int)temp_value;

  return SUCCESS;
}

int read_int(int *value, const char *prompt) {
  show_prompt(prompt);
  return read_int_quiet(value, prompt);
}

int read_int_within_range(int *value, const char *prompt,
                          const IntRange *range) {
  show_prompt_within_range(prompt, range);
  if (read_int_quiet(value, prompt) == FAILURE) {
    return FAILURE;
  };

  RangeCheckResult range_check = validate_range_int(*value, range);
  if (range_check != WITHIN_RANGE) {
    show_range_error_int(range_check, range);
    return FAILURE;
  }
  return SUCCESS;
}

int read_int_within_range_with_validation(const char *prompt,
                                  const IntRange *range,
                                  IntValidation additional_check, ...) {
  int value;
  bool is_valid = false;

  va_list args;

  while (!is_valid) {
    if (read_int_within_range(&value, prompt, range) == SUCCESS) {
      if (additional_check != NULL) {
        va_start(args, additional_check);
        is_valid = additional_check(value, args);
        va_end(args);

      } else {
        is_valid = true;
      }
    }
  }

  return value;
}
```

../src/io/double.h

```c
#ifndef DOUBLE_H
#define DOUBLE_H

#include "common/types.h"

void print_double_range(const char *name, const DoubleRange *range);
void print_double_character_count_range();
```

```
void print_double_precision_limit();
void print_truncated_double(double num, int decimal_places);
int read_double_within_range(double *value, const char *prompt,
                      const DoubleRange *range);
double read_double_within_range_with_validation(
   const char *prompt, const DoubleRange *range,
   DoubleValidation additional_check, ...);

#endif
```

../src/io/double.c

```
#include <errno.h>
#include <float.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#include "common/constants.h"
#include "common/types.h"
#include "int.h"
#include "utils.h"

#define TOLERANCE DBL_EPSILON
#define RELATIVE_TOLERANCE 1e-12
#define MIN_ABSOLUTE_VALUE DBL_MIN
#define MAX_ABSOLUTE_VALUE DBL_MAX
#define MAX_CHAR_COUNT_DOUBLE 23
#define OVERFLOW_ABSOLUTE_VALUE HUGE_VAL
#define MAX_SIGNIFICANT_DIGITS DBL_DIG

const IntRange DOUBLE_CHAR_COUNT_RANGE = {1, MAX_CHAR_COUNT_DOUBLE, true, true
    };

// OUTPUT
void show_prompt_double_within_range(const char *prompt,
                              const DoubleRange *range) {
  printf("%s від (%lg до %lg): ", prompt, range->min, range->max);
}

void show_range_error_double(RangeCheckResult result,
                      const DoubleRange *range) {
  switch (result) {
    case LESS:
      handle_error("Значення має бути ≥ %lg.", range->min);
      break;
    case LESS_EQUAL:
      handle_error("Значення має бути > %lg.", range->min);
      break;
    case GREATER:
      handle_error("Значення має бути ≤ %lg.", range->max);
      break;
    case GREATER_EQUAL:
      handle_error("Значення має бути < %lg.", range->max);
      break;
    default:
      break;
  }
}
```

```c

// RANGES
RangeCheckResult validate_range_double(double value, const DoubleRange *range)
    {
  if (range->is_min_included && value < range->min - TOLERANCE) return LESS;
  if (!range->is_min_included && value ⩽ range->min + TOLERANCE)
    return LESS_EQUAL;
  if (range->is_max_included && value > range->max + TOLERANCE) return GREATER;
  if (!range->is_max_included && value ⩾ range->max - TOLERANCE)
    return GREATER_EQUAL;
  return WITHIN_RANGE;
}

void print_double_range(const char *name, const DoubleRange *range) {
  printf("%g", range->min);
  if (range->is_min_included) {
    printf(" ≤ ");
  } else {
    printf(" < ");
  }

  printf("%s", name);

  if (range->is_max_included) {
    printf(" ≤ ");
  } else {
    printf(" < ");
  }
  printf("%g\n", range->max);
}

void print_double_character_count_range() {
  print_int_range("кількість символів у дійсних числах",
                  &DOUBLE_CHAR_COUNT_RANGE);
}

void print_double_precision_limit() {
  printf("Будьякі- дійсні числа приймаються з точністю, не меншою за %g",
         TOLERANCE);
}

RangeCheckResult validate_overflow_double(double value) {
  if (errno == ERANGE) {
    if (fabs(value) == OVERFLOW_ABSOLUTE_VALUE) return GREATER_EQUAL;
    if (fabs(value) < MIN_ABSOLUTE_VALUE) return LESS_EQUAL;
  }
  return WITHIN_RANGE;
}

// UTILS
double truncate_to_precision_double(double num, int decimal_places) {
  double factor = pow(10, decimal_places);
  return trunc(num * factor) / factor;
}

void print_truncated_double(double num, int decimal_places) {
  printf("%.*lf", decimal_places,
         truncate_to_precision_double(num, decimal_places));
}

```

```c
bool is_double_precise(double value) {
  if (isinf(value) || isnan(value)) {
    return false;
  }

  if (value == 0.0) {
    return true;
  }

  double scale = pow(10, MAX_SIGNIFICANT_DIGITS);
  double scaled_value = round(value * scale);
  double rounded_value = scaled_value / scale;

  double tolerance = fabs(value) * RELATIVE_TOLERANCE;

  return fabs(value - rounded_value) < tolerance;
}

// INPUT

int read_double_quiet(double *value, const char *prompt) {
  char input[MAX_CHAR_COUNT_DOUBLE + 2];
  errno = 0;

  if (read_input(input, MAX_CHAR_COUNT_DOUBLE) == FAILURE) {
    return FAILURE;
  }

  if (!is_input_within_length(input)) {
    clear_stdin();
    handle_error_overlength(MAX_CHAR_COUNT_DOUBLE);
    return FAILURE;
  }

  replace_commas_with_dots(input);

  char *endptr;
  *value = strtod(input, &endptr);

  if (!is_input_number_after_conversion(endptr, input)) {
    handle_error_not_number();
    return FAILURE;
  }

  RangeCheckResult global_check = validate_overflow_double(*value);
  if (global_check != WITHIN_RANGE) {
    handle_error_overflow();
    return FAILURE;
  }

  if (!is_double_precise(*value)) {
    handle_error_not_precise(MAX_SIGNIFICANT_DIGITS);
    return FAILURE;
  }

  return SUCCESS;
}

int read_double(double *value, const char *prompt) {
  show_prompt(prompt);
```

```
  return read_double_quiet(value, prompt);
}

int read_double_within_range(double *value, const char *prompt,
                             const DoubleRange *range) {
  show_prompt_double_within_range(prompt, range);

  if (read_double_quiet(value, prompt) == FAILURE) {
    return FAILURE;
  }

  RangeCheckResult range_check = validate_range_double(*value, range);
  if (range_check != WITHIN_RANGE) {
    show_range_error_double(range_check, range);
    return FAILURE;
  }
  return SUCCESS;
}

double read_double_within_range_with_validation(
    const char *prompt, const DoubleRange *range,
    DoubleValidation additional_check, ...) {
  double value;
  bool is_valid = false;

  va_list args;

  while (!is_valid) {
    if (read_double_within_range(&value, prompt, range) == SUCCESS) {
      if (additional_check != NULL) {
        va_start(args, additional_check);
        is_valid = additional_check(value, args);
        va_end(args);

      } else {
        is_valid = true;
      }
    }
  }

  return value;
}
```

../src/io/validators.h

```
#ifndef VALIDATORS_H
#define VALIDATORS_H

#include <stdarg.h>
#include <stdbool.h>

bool is_greater(double b, va_list args);

#endif // !VALIDATORS_H
```

../src/io/validators.c

```c
#include "validators.h"

#include <stdarg.h>
#include <stdbool.h>

#include "utils.h"

bool is_greater(double b, va_list args) {
  double a = va_arg(args, double);
  bool is_greater = b > a;
  if (!is_greater) {
    handle_error("Верхня межа повинна бути більшою за нижню");
  }
  return is_greater;
}
```

../meson.build

```
project('op-lab-6', 'c')

c_args = ['-Isrc', '-g' ]
include_dirs = include_directories('src')

main_sources = [
  'src/main.c',
  'src/algorithm.c',
  'src/io/int.c',
  'src/io/double.c',
  'src/io/validators.c',
  'src/io/utils.c',
  'src/io/equations.c',
]

executable('main.out', main_sources,
  c_args: c_args,
  include_directories: include_dirs,
  link_args: ['-lm']
)
```

Введені та одержані результати

```
Вас вітає програма NumberWorld, що вирішує задану
 систему алгебраїчних рівнянь (СЛАР) методом Гаусса-Зейделя із заданою точністю

УВАГА! Наші обмеження такі:
2 ≤ кількість рівнянь у системі ≤ 10000
-1e+12 ≤ мінімальне та максимальне значення коефіцієнтів у рівняннях ≤ 1e+12
мінімальне значення коефіцієнтів < максимальне значення коефіцієнтів
2.22045e-16 ≤ точність < 1
1 ≤ кількість символів у дійсних числах ≤ 23
1 ≤ кількість символів у цілих числах ≤ 10
Будь-які дійсні числа приймаються з точністю, не меншою за 2.22045e-16

УВАГА! Алгоритм Гаусса-Зейделя вимагає, щоб матриця була діагонально домінантною,
Модуль будь-якого діагонального елемента
 має бути більшим за суму модулів елементів того самого рядка
Якщо Ви не дотримаєте цього правила, отримаєте помилку.

УВАГА! Випадкова генерація хоч і діє в проміжку, що зазначаєте Ви,
проте діагональні елементи підбираються для сходимості алгоритму.

Розмір СЛАР (від 2 до 10000): 2
Мінімальне значення коефіцієнтів (від -1e+12 до 1e+12): 1
Максимальне значення коефієнтів (від -1e+12 до 1e+12): 100
Точність (від 2.22045e-16 до 1): 1e-12

Чи бажаєте Ви уввімкнути випадковий режим?
Введіть +, якщо погоджуєтесь. Інакше введіть будь-яку іншу клавішу: -
Введіть рівняння №1:
? • x_1  (від 1 до 100): 2
2.000000000000x_1 + ? • x_2  (від 1 до 100): 1
2.000000000000x_1 + 1.000000000000x_2 = ?:  (від 1 до 100): 20
2.000000000000x_1 + 1.000000000000x_2 = 20.000000000000
Введіть рівняння №2:
? • x_1  (від 1 до 100): 3
3.000000000000x_1 + ? • x_2  (від 1 до 100): 10
3.000000000000x_1 + 10.000000000000x_2 = ?:  (від 1 до 100): -2

ПОМИЛКА! Значення має бути ≥ 1.
Натисніть Enter, щоб продовжити:
3.000000000000x_1 + 10.000000000000x_2 = ?:  (від 1 до 100): 2
3.000000000000x_1 + 10.000000000000x_2 = 2.000000000000

ПЕРЕМОГА! Корені знайдено успішно!
x_1 = 11.647058823529, x_2 = -3.294117647058
Чи хочете ви повторити?
Введіть +, якщо погоджуєтесь. Інакше введіть будь-яку іншу клавішу:
```

```
Розмір СЛАР (від 2 до 10000): 3
Мінімальне значення коефіцієнтів (від -1e+12 до 1e+12): -100
Максимальне значення коефієнтів (від -1e+12 до 1e+12): 100
Точність (від 2.22045e-16 до 1): 1e-3

Чи бажаєте Ви уввімкнути випадковий режим?
Введіть +, якщо погоджуєтесь. Інакше введіть будь-яку іншу клавішу: +
Рівняння №1:
164.086x_1 + 75.828x_2 + 27.603x_3 = -62.921
Рівняння №2:
70.916x_1 + 210.448x_2 + 39.887x_3 = 63.813
Рівняння №3:
21.099x_1 + 98.123x_2 + 143.394x_3 = -42.319

ПЕРЕМОГА! Корені знайдено успішно!
x_1 = -0.559, x_2 = 0.611, x_3 = -0.631
Чи хочете ви повторити?
Введіть +, якщо погоджуєтесь. Інакше введіть будь-яку іншу клавішу:
```

```
Розмір СЛАР (від 2 до 10000): 4
Мінімальне значення коефіцієнтів (від -1e+12 до 1e+12): 1
Максимальне значення коефієнтів (від -1e+12 до 1e+12): 5
Точність (від 2.22045e-16 до 1): 4

ПОМИЛКА! Значення має бути < 1.
Натисніть Enter, щоб продовжити:
Точність (від 2.22045e-16 до 1): 1e-4

Чи бажаєте Ви уввімкнути випадковий режим?
Введіть +, якщо погоджуєтесь. Інакше введіть будь-яку іншу клавішу: -
Введіть рівняння №1:
? • x_1  (від 1 до 5): 4
4.0000x_1 + ? • x_2  (від 1 до 5): 1
4.0000x_1 + 1.0000x_2 + ? • x_3  (від 1 до 5): 1
4.0000x_1 + 1.0000x_2 + 1.0000x_3 + ? • x_4  (від 1 до 5): 1
4.0000x_1 + 1.0000x_2 + 1.0000x_3 + 1.0000x_4 = ?:  (від 1 до 5): 1
4.0000x_1 + 1.0000x_2 + 1.0000x_3 + 1.0000x_4 = 1.0000
Введіть рівняння №2:
? • x_1  (від 1 до 5): 1
1.0000x_1 + ? • x_2  (від 1 до 5): 4
1.0000x_1 + 4.0000x_2 + ? • x_3  (від 1 до 5): 1
1.0000x_1 + 4.0000x_2 + 1.0000x_3 + ? • x_4  (від 1 до 5): 1
1.0000x_1 + 4.0000x_2 + 1.0000x_3 + 1.0000x_4 = ?:  (від 1 до 5): 2
1.0000x_1 + 4.0000x_2 + 1.0000x_3 + 1.0000x_4 = 2.0000
Введіть рівняння №3:
? • x_1  (від 1 до 5): 1
1.0000x_1 + ? • x_2  (від 1 до 5): 1
1.0000x_1 + 1.0000x_2 + ? • x_3  (від 1 до 5): 4
1.0000x_1 + 1.0000x_2 + 4.0000x_3 + ? • x_4  (від 1 до 5): 1
1.0000x_1 + 1.0000x_2 + 4.0000x_3 + 1.0000x_4 = ?:  (від 1 до 5): 3
1.0000x_1 + 1.0000x_2 + 4.0000x_3 + 1.0000x_4 = 3.0000
Введіть рівняння №4:
? • x_1  (від 1 до 5): 1
1.0000x_1 + ? • x_2  (від 1 до 5): 1
1.0000x_1 + 1.0000x_2 + ? • x_3  (від 1 до 5): 1
1.0000x_1 + 1.0000x_2 + 1.0000x_3 + ? • x_4  (від 1 до 5): 4
1.0000x_1 + 1.0000x_2 + 1.0000x_3 + 4.0000x_4 = ?:  (від 1 до 5): 4
1.0000x_1 + 1.0000x_2 + 1.0000x_3 + 4.0000x_4 = 4.0000

ПЕРЕМОГА! Корені знайдено успішно!
x_1 = -0.1428, x_2 = 0.1904, x_3 = 0.5238, x_4 = 0.8571
Чи хочете ви повторити?
Введіть +, якщо погоджуєтесь. Інакше введіть будь-яку іншу клавішу:
```

```
Розмір СЛАР (від 2 до 10000): 10
Мінімальне значення коефіцієнтів (від -1e+12 до 1e+12): -100
Максимальне значення коефієнтів (від -1e+12 до 1e+12): 100
Точність (від 2.22045e-16 до 1): 1e-10

Чи бажаєте Ви уввімкнути випадковий режим?
Введіть +, якщо погоджуєтесь. Інакше введіть будь-яку іншу клавішу: +
Рівняння №1:
544.2442106083x_1 + 48.3446130529x_2 + 85.9307836741x_3 + 53.3846558071x_4 + 75.3171826712x_5 +
        11.1498164944x_6 + 22.6787787862x_7 + 91.5360751561x_8 + 64.6735191345x_9 + 39.7906413301x_10 = -43.9933824352
Рівняння №2:
32.3513666167x_1 + 585.0872702896x_2 + 75.9092286229x_3 + 66.5525016374x_4 + 20.0213695876x_5 +
        95.7208361476x_6 + 97.7150163613x_7 + 60.6805434450x_8 + 65.9885268658x_9 + 49.9402101151x_10 = -74.6553321368
Рівняння №3:
97.8051814250x_1 + 22.2138401120x_2 + 496.7742037028x_3 + 22.4028622731x_4 + 96.0093244910x_5 +
        17.5048520788x_6 + 60.2490698918x_7 + 9.0301644057x_8 + 90.7289128750x_9 + 9.5173848792x_10 = -95.2018707990
Рівняння №4:
56.1327290721x_1 + 17.9424052126x_2 + 15.9479457885x_3 + 523.7287939991x_4 + 66.5460496209x_5 +
        9.4784803688x_6 + 51.2744266539x_7 + 73.2445917092x_8 + 87.6452306285x_9 + 92.7189557813x_10 = -63.5544592514
Рівняння №5:
59.2714574187x_1 + 85.5745888315x_2 + 59.2752953991x_3 + 61.5564410574x_4 + 605.4263013415x_5 +
        46.2551322765x_6 + 25.2638222649x_7 + 88.3837690576x_8 + 86.6704741492x_9 + 0.0808455981x_10 = 8.8843142613
Рівняння №6:
77.5162920355x_1 + 82.1982750669x_2 + 91.3794621825x_3 + 37.7653619274x_4 + 26.8318892456x_5 +
        704.6072255820x_6 + 82.1083750575x_7 + 71.7520229518x_8 + 84.2066667042x_9 + 86.9065043516x_10 = 33.7357386015
Рівняння №7:
2.8544501401x_1 + 61.3387024961x_2 + 56.7857810296x_3 + 45.8711232058x_4 + 34.5832942053x_5 +
        44.4310115650x_6 + 403.3298916649x_7 + 38.5900789871x_8 + 59.8207473754x_9 + 7.9854708164x_10 = -74.2461585439
Рівняння №8:
32.7392338775x_1 + 40.5820226296x_2 + 72.0089737325x_3 + 92.5245883874x_4 + 28.9657917805x_5 +
        58.6794478818x_6 + 7.5562573038x_7 + 445.6672783475x_8 + 15.1547423563x_9 + 32.4362377636x_10 = -2.6469825766
Рівняння №9:
41.0567755810x_1 + 92.2746033407x_2 + 70.5211281776x_3 + 58.9484004303x_4 + 64.0266263857x_5 +
        86.3144615665x_6 + 72.0418960787x_7 + 8.0886214971x_8 + 578.7289902567x_9 + 20.0502001680x_10 = -46.7499190010
Рівняння №10:
63.2644191384x_1 + 84.9414308555x_2 + 87.8333752043x_3 + 81.1665924265x_4 + 53.6486481316x_5 +
        71.9873721711x_6 + 10.8479368500x_7 + 48.4898155555x_8 + 46.2335307151x_9 + 570.2067064121x_10 = -10.9281617216

ПЕРЕМОГА! Корені знайдено успішно!
x_1 = -0.0441097133, x_2 = -0.0872235222, x_3 = -0.1649598248, x_4 = -0.1063362650, x_5 = 0.0563139343,
x_6 = 0.1029099952, x_7 = -0.1500373963, x_8 = 0.0388550854, x_9 = -0.0370713692, x_10 = 0.0235341127
```

Теоретичні розрахунки

Слід виконати перевірку відповідей підстановкою коренів у рівняння:

Випадок 1.

$2 \cdot 11.647058823529 + 1 \cdot -3.294117647058 = 20 == 20$

$3 \cdot 11.647058823529 + 10 \cdot -3.294117647058 = 2.0000000000069917 \approx 2$

Випадок 2.

$164.086 \cdot -0.559 + 75.828 \cdot 0.611 + 27.603 \cdot -0.631 = -62.810659000000015 \approx$ -62.921

$70.916 \cdot -0.559 + 210.448 \cdot 0.611 + 39.887 \cdot -0.631 = 63.77298700000001 \approx$ 63.813

$21.099 \cdot -0.559 + 98.123 \cdot 0.611 + 143.394 \cdot -0.631 = -42.32280200000001 \approx$ -42.319

Випадок 3.

$4 \cdot -0.1428 + 1 \cdot 0.1904 + 1 \cdot 0.5238 + 1 \cdot 0.8571 = 1.0001 \approx 1$

$1 \cdot -0.1428 + 4 \cdot 0.1904 + 1 \cdot 0.5238 + 1 \cdot 0.8571 = 1.9997 \approx 2$

$1 \cdot -0.1428 + 1 \cdot 0.1904 + 4 \cdot 0.5238 + 1 \cdot 0.8571 = 2.9999000000000002 \approx 3$

$1 \cdot -0.1428 + 1 \cdot 0.1904 + 1 \cdot 0.5238 + 4 \cdot 0.8571 = 3.9998 \approx 4$

Випадок 4. Розв'язки цілком співпадають зі знайденими коренями.

$x_1 = -0.0441097134$

$x_2 = 0.0235341127$

$x_3 = -0.0872235223$

$x_4 = -0.1649598249$

$x_5 = -0.1063362650$

$x_6 = 0.0563139343$

$x_7 = 0.1029099953$

$x_8 = -0.1500373963$

$x_9 = 0.0388550855$

$x_{10} = -0.0370713693$

Висновки: Програма працює коректно. Програма вирішує поставлене завдання.