

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 ІІІ
Ахаладзе А. Е.
«13» грудня 2024 р.

Виконав
студент групи ІІ-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

```
1 #include <math.h>
2 #include <stdbool.h>
3 #include <stdio.h>
4 #include <stdlib.h>
5
6 #include "algorithm.h"
7 #include "common/constants.h"
8 #include "common/types.h"
9 #include "io/double.h"
10 #include "io/equations.h"
11 #include "io/int.h"
12 #include "io/utils.h"
13 #include "io/validators.h"
14
15 const IntRange EQUATION_COUNT_RANGE = {2, 10000, true, true};
16 const DoubleRange COEF_RANGE = {-1e12, 1e12, true, true};
17 const DoubleRange PRECISION_RANGE = {DBL_EPSILON, 1.0, true, false};
18
19 void print_demo();
20
21 int main() {
22     print_demo();
23     do {
24         Equations eqs;
25         eqs.count = read_int_within_range_with_validation(
26             "Розмір_СЛАР", &EQUATION_COUNT_RANGE, NULL);
27         if (initialize_equations(&eqs) == FAILURE) {
28             return FAILURE;
29         }
30
31         DoubleRange user_coef_range;
32         user_coef_range.min = read_double_within_range_with_validation(
33             "Мінімальне_значення_коефіцієнтів", &COEF_RANGE, NULL);
34
35         user_coef_range.max = read_double_within_range_with_validation(
36             "Максимальне_значення_коефіцієнтів", &COEF_RANGE, is_greater,
37             user_coef_range.min);
38         user_coef_range.is_max_included = user_coef_range.is_min_included = true;
39
40         double precision = read_double_within_range_with_validation(
41             "Точність", &PRECISION_RANGE, NULL);
42
43         int decimal_places = (int)(-log10(precision));
44
45         if (ask_question("Чи_бажаєте_Ви_уввімкнути_випадковий_режим?")) {
46             generate_equations(&eqs, &user_coef_range);
47             print_equations(&eqs, decimal_places);
48         } else {
```

```

49     handle_equation_input(&eqs, &user_coef_range, decimal_places);
50 }
51
52 Solutions sol;
53 sol.count = eqs.count;
54
55 if (initialize_solutions(&sol) == FAILURE) {
56     return FAILURE;
57 }
58
59 if (solve_equations(&eqs, &sol, precision) == SUCCESS) {
60     print_solutions(&sol, decimal_places);
61 } else {
62     handle_error("Ця система не є збіжною для методу!");
63 };
64
65 free_program(&eqs, &sol);
66
67 } while (ask_question("Чи хочете ви повторити?"));
68
69 return 0;
70 }
71
72 void print_demo() {
73     printf(
74         "\nВас вітає програма NumberWorld, що вирішує задану\n систему\n"
75         "алгебраїчних рівнянь СЛАР() методом Гаусса-Зейделя з заданою\n"
76         "точністю\n");
77     show_warning("Наші обмеження такі:");
78     print_int_range("кількість рівнянь у системі", &EQUATION_COUNT_RANGE);
79     print_double_range(
80         "мінімальне та максимальне значення коефіцієнтів у рівняннях",
81         &COEF_RANGE);
82     printf(
83         "мінімальне значення коефіцієнтів < максимальне значення коефіцієнтів\n");
84     print_double_range("точність", &PRECISION_RANGE);
85
86     print_double_character_count_range();
87     print_int_character_count_range();
88     print_double_precision_limit();
89
90     printf("\n");
91     show_warning(
92         "Алгоритм Гаусса-Зейделя вимагає, щоб матриця була діагонально\n"
93         "домінантною, тобто:");
94     printf(
95         "Модуль будьякого діагонального елемента\n має бути більшим за суму\n"
96         "модулів елементів того самого рядка\n");
97     printf("Якщо Ви не дотримаєте цього правила, отримаєте помилку.\n");
98     show_warning(
99         "Випадкова генерація хоч і діє в проміжку, що зазначаєте Ви,\n протен\n"
100         "діагональні елементи підбираються для сходимості алгоритму.");
101     printf("\n");
102 }

```

../src/common/constants.h

```

1 #ifndef CONSTANTS_H
2 #define CONSTANTS_H

```

```

3
4 #define SUCCESS 0
5 #define FAILURE 1
6
7 #define RED "\033[31m"
8 #define GREEN "\033[32m"
9 #define YELLOW "\033[33m"
10 #define RESET "\033[0m"
11
12 #endif

```

../src/common/types.h

```

1 #ifndef TYPES_H
2 #define TYPES_H
3
4 #include <stdarg.h>
5 #include <stdbool.h>
6
7 typedef enum {
8     LESS,
9     LESS_EQUAL,
10    GREATER,
11    GREATER_EQUAL,
12    WITHIN_RANGE
13 } RangeCheckResult;
14
15 typedef struct {
16     double min, max;
17     bool is_min_included, is_max_included;
18 } DoubleRange;
19
20 typedef struct {
21     int min, max;
22     bool is_min_included, is_max_included;
23 } IntRange;
24
25 typedef bool (*DoubleValidation)(double value, va_list args);
26 typedef bool (*IntValidation)(int value, va_list args);
27
28 #endif

```

../src/algorithm.h

```

1 #ifndef ALGORITHM_H
2 #define ALGORITHM_H
3
4 #include <float.h>
5 #include <stdbool.h>
6
7 #include "common/types.h"
8
9 typedef struct {
10     int count;
11     double** coefs;
12     double* const_coefs;
13 } Equations;

```

```

14
15 typedef struct {
16     int count;
17     double* corrs; // corrections
18     double* roots;
19 } Solutions;
20
21 int initialize_equations(Equations* eqs);
22 int initialize_solutions(Solutions* sol);
23 void free_program(Equations* eqs, Solutions* sol);
24
25 void generate_equations(Equations* eqs, DoubleRange* coef_range);
26
27 bool is_convergent_row(Equations* eqs, int row);
28
29 int solve_equations(Equations* eqs, Solutions* sol, double precision);
30
31 #endif

```

../src/algorithm.c

```

1 #include "algorithm.h"
2
3 #include <math.h>
4 #include <stdbool.h>
5 #include <stdlib.h>
6 #include <time.h>
7
8 #include "common/constants.h"
9 #include "io/utils.h"
10
11 const DoubleRange DIAGONAL_RANGE = {0.0, 100.0};
12
13 void initialize_generator() { srand(time(NULL)); }
14
15 int initialize_equations(Equations* eqs) {
16     eqs->coefs = (double**)malloc(eqs->count * sizeof(double*));
17
18     if (eqs->coefs == NULL) {
19         free(eqs->coefs);
20         handle_error_memory_allocation("вказівників_на_коефіцієнти_рівнянь");
21         return FAILURE;
22     };
23
24     for (int i = 0; i < eqs->count; i++) {
25         eqs->coefs[i] = (double*)malloc(eqs->count * sizeof(double));
26         if (eqs->coefs[i] == NULL) {
27             for (int j = i; j ≥ 0; j--) {
28                 free(eqs->coefs[j]);
29             }
30             free(eqs->coefs);
31             handle_error_memory_allocation("коефіцієнтів_рівнянь");
32         }
33     }
34
35     eqs->const_coefs = (double*)malloc(eqs->count * sizeof(double));
36     if (eqs->const_coefs == NULL) {
37         free(eqs->const_coefs);
38         free(eqs->coefs);

```

```

39     handle_error_memory_allocation("вільних_членів_рівнянь");
40     return FAILURE;
41 }
42 return SUCCESS;
43 }
44
45 int initialize_solutions(Solutions* sol) {
46     sol->roots = (double*)malloc(sol->count * sizeof(double));
47     if (sol->roots == NULL) {
48         free(sol->roots);
49         handle_error_memory_allocation("коренів_рівнянь");
50         return FAILURE;
51     }
52
53     sol->corrs = (double*)malloc(sol->count * sizeof(double));
54     if (sol->corrs == NULL) {
55         free(sol->roots);
56         free(sol->corrs);
57         handle_error_memory_allocation(
58             "коррекцій_коренів_використовуються(всередині_алгоритму)");
59         return FAILURE;
60     }
61     return SUCCESS;
62 }
63
64 void free_program(Equations* eqs, Solutions* sol) {
65     for (int i = 0; i < eqs->count; i++) {
66         free(eqs->coefs[i]);
67     }
68     free(eqs->coefs);
69     free(eqs->const_coefs);
70
71     free(sol->roots);
72     free(sol->corrs);
73 }
74
75 double generate_double(const DoubleRange* range) {
76     double random_0_1 = (double)rand() / ((double)RAND_MAX + 1.0);
77     return (range->min + (range->max - range->min) * random_0_1);
78 }
79
80 void generate_equations(Equations* eqs, DoubleRange* coef_range) {
81     initialize_generator();
82
83     for (int i = 0; i < eqs->count; i++) {
84         double compound_sum = 0.0;
85
86         for (int j = 0; j < eqs->count; j++) {
87             if (j == i) continue;
88             eqs->coefs[i][j] = fabs(generate_double(coef_range));
89             compound_sum += eqs->coefs[i][j];
90         }
91
92         // There would be no overflow for our constraints
93         double diagonal_item = compound_sum + generate_double(&DIAGONAL_RANGE);
94         eqs->coefs[i][i] = diagonal_item;
95
96         eqs->const_coefs[i] = generate_double(coef_range);
97     }
98 }

```

```

99
100 bool is_convergent_row(Equations* eqs, int row) {
101     double non_diagonal_sum = 0;
102
103     for (int i = 0; i < eqs->count; i++) {
104         if (i == row) continue;
105         non_diagonal_sum += fabs(eqs->coefs[row][i]);
106     }
107
108     if (fabs(eqs->coefs[row][row]) ≤ non_diagonal_sum) {
109         return false;
110     }
111     return true;
112 }
113
114 bool is_convergent_equations(Equations* eqs) {
115     for (int i = 0; i < eqs->count; i++) {
116         if (!is_convergent_row(eqs, i)) {
117             return false;
118         }
119     }
120     return true;
121 }
122
123 int solve_equations(Equations* eqs, Solutions* sol, double precision) {
124     // Ensure diagonal dominance for convergence
125     if (!is_convergent_equations(eqs)) {
126         return FAILURE;
127     }
128
129     for (int i = 0; i < eqs->count; i++) {
130         sol->roots[i] = eqs->const_coefs[i] / eqs->coefs[i][i]; // Initial guess
131     }
132
133     double max_correction;
134     do {
135         max_correction = 0.0;
136         for (int i = 0; i < eqs->count; i++) {
137             double compound_sum = 0.0;
138
139             for (int j = 0; j < eqs->count; j++) {
140                 if (j == i) continue;
141                 compound_sum += eqs->coefs[i][j] * sol->roots[j];
142             }
143
144             double new_value =
145                 (eqs->const_coefs[i] - compound_sum) / eqs->coefs[i][i];
146             double delta = fabs(new_value - sol->roots[i]);
147
148             if (delta > max_correction) {
149                 max_correction = delta;
150             }
151
152             sol->roots[i] = new_value;
153         }
154     } while (max_correction ≥ precision);
155
156     return SUCCESS;
157 }

```

../src/io/equations.c

```
1 #include "equations.h"
2
3 #include <stdbool.h>
4 #include <stdio.h>
5
6 #include "../algorithm.h"
7 #include "common/constants.h"
8 #include "double.h"
9 #include "utils.h"
10
11 #define COEFS_PER_ROW 5
12 #define ROOTS_PER_ROW 5
13 #define CLEAR_LINE_COUNT 1
14
15 void print_equation_row(double* row_coefs, int start, int end,
16                        int decimal_places) {
17     for (int j = start; j ≤ end; j++) {
18         print_truncated_double(row_coefs[j], decimal_places);
19         printf("x_%d", j + 1);
20         if (j < end) {
21             printf("_+_");
22         }
23     }
24 }
25
26 void print_equations(Equations* eqs, int decimal_places) {
27     for (int i = 0; i < eqs->count; i++) {
28         printf(GREEN "Рівняння №%d:" RESET "\n", i + 1); // Print equation header
29
30         for (int j = 0; j < eqs->count; j += COEFS_PER_ROW) {
31             bool is_last = (j + COEFS_PER_ROW) ≥ eqs->count;
32             int row_end = is_last ? (eqs->count - 1) : (j + COEFS_PER_ROW - 1);
33
34             if (j > 0) {
35                 printf("\t");
36             }
37
38             print_equation_row(eqs->coefs[i], j, row_end, decimal_places);
39
40             if (!is_last) {
41                 printf("_+\n");
42             }
43         }
44
45         printf("_=_");
46         print_truncated_double(eqs->const_coefs[i], decimal_places);
47         printf("\n");
48     }
49 }
50
51 void print_solutions(Solutions* sol, int decimal_places) {
52     show_success("Корені знайдено успішно!");
53     for (int i = 0; i < sol->count; i++) {
54         printf("x_%d=_", i + 1);
55         print_truncated_double(sol->roots[i], decimal_places);
56
57         if (i ≠ sol->count - 1) {
58             printf(",_");
```



```

59     }
60
61     if (i % ROOTS_PER_ROW == (ROOTS_PER_ROW - 1) &&
62         sol->count > ROOTS_PER_ROW) {
63         printf("\n");
64     }
65 }
66 }
67
68 void ask_for_coef(double* row_coefs, int coef_index, int decimal_places) {
69     int last_row_start = coef_index - (coef_index % COEFS_PER_ROW);
70
71     print_equation_row(row_coefs, last_row_start, coef_index - 1, decimal_places);
72
73     if (coef_index != 0) {
74         printf("_+_");
75     }
76
77     printf("?_·_x_%d_", coef_index + 1);
78 }
79
80 void handle_equation_input(Equations* eqs, DoubleRange* coef_range,
81                             int decimal_places) {
82     for (int i = 0; i < eqs->count; i++) {
83         printf(GREEN "Введіть_рівняння_N%d:\n" RESET, i + 1);
84
85         for (int coef_index = 0; coef_index < eqs->count; coef_index++) {
86             ask_for_coef(eqs->coefs[i], coef_index, decimal_places);
87             while (read_double_within_range(&eqs->coefs[i][coef_index], "",
88                                             coef_range) == FAILURE) {
89                 ask_for_coef(eqs->coefs[i], coef_index, decimal_places);
90             };
91
92             bool is_row_end = (coef_index % COEFS_PER_ROW) == (COEFS_PER_ROW - 1);
93
94             // clear_last_lines(CLEAR_LINE_COUNT); avoid clearing so that it can be
95             // seen in report
96             if (is_row_end) {
97                 int last_row_start = coef_index - (coef_index % COEFS_PER_ROW);
98                 print_equation_row(eqs->coefs[i], last_row_start, coef_index,
99                                 decimal_places);
100                 printf("_+_\\n");
101             }
102         }
103
104         if (!is_convergent_row(eqs, i)) {
105             int equation_line_count =
106                 (eqs->count + COEFS_PER_ROW - 1) / COEFS_PER_ROW;
107             handle_error(
108                 "Введений_Вами_рядок_коефіцієнтів_не_задовольняє_умови_для_\\n"
109                 "правильної_роботи_алгоритму_ГауссаЗейделя-");
110             clear_last_lines(equation_line_count + 1);
111             i--;
112             continue;
113         }
114
115         do {
116             print_equation_row(eqs->coefs[i],
117                               eqs->count - (eqs->count % COEFS_PER_ROW),
118                               eqs->count - 1, decimal_places);

```

```

119     printf("_=_?:_");
120 } while (read_double_within_range(&eqs->const_coefs[i], "", coef_range) ==
121         FAILURE);
122
123 // clear_last_lines(CLEAR_LINE_COUNT); avoid clearing so that it can be seen
124 // in report
125
126 print_equation_row(eqs->coefs[i], eqs->count - (eqs->count % COEFS_PER_ROW),
127                   eqs->count - 1, decimal_places);
128 printf("_=_");
129 print_truncated_double(eqs->const_coefs[i], decimal_places);
130 printf("\n");
131 }
132 }

```

../src/io/equations.h

```

1 #ifndef EQUATIONS_H
2 #define EQUATIONS_H
3
4 #include "../algorithm.h"
5
6 void print_equations(Equations* eqs, int decimal_places);
7 void print_solutions(Solutions* sol, int decimal_places);
8 void handle_equation_input(Equations* eqs, DoubleRange* coef_range,
9                           int decimal_places);
10
11 #endif // !EQUATIONS_H

```

../src/io/utils.h

```

1 #ifndef UTILS_H
2 #define UTILS_H
3
4 #include <stdbool.h>
5
6 // UTILS
7 void replace_commas_with_dots(char *string);
8
9 // OUTPUT
10 void clear_last_lines(int count);
11 void show_prompt(const char *prompt);
12 void handle_error(const char *format, ...);
13 void handle_error_overlength(int max_char_count);
14 void handle_error_memory_allocation(const char *reason);
15 void handle_error_overflow();
16 void handle_error_not_number();
17 void handle_error_not_precise(int max_significant_digits);
18 void show_warning(const char *format, ...);
19 void show_success(const char *format, ...);
20
21 // INPUT TAKING
22 void clear_stdin();
23 int read_input(char *input, int max_char_count);
24 bool ask_question(const char *format, ...);
25
26 // INPUT CHECKING

```

```

27 bool is_input_within_length(const char *input);
28 bool is_input_floating_point(char *input);
29 bool is_input_number_after_conversion(const char *endptr, const char *input);
30
31 #endif

```

../src/io/utls.c

```

1 #include "utls.h"
2
3 #include <stdarg.h>
4 #include <stdbool.h>
5 #include <stdio.h>
6 #include <stdlib.h>
7 #include <string.h>
8
9 #include "common/constants.h"
10
11 #define MOVE_UP "\033[F"
12 #define CLEAR_LINE "\033[2K"
13
14 #define ERROR_CLEAR_LINE_COUNT 4
15
16 // UTILS
17 void replace_commas_with_dots(char *string) {
18     while (*string) {
19         if (*string == ',') {
20             *string = '.';
21         }
22         string++;
23     }
24 }
25
26 // OUTPUT
27
28 void clear_last_lines(int count) {
29     for (int i = 0; i < count; i++) {
30         printf(MOVE_UP CLEAR_LINE);
31     }
32 }
33
34 void show_prompt(const char *prompt) { printf("%s:", prompt); }
35
36 void continue_after_error() {
37     printf("Натисніть Enter, щоб продовжити:");
38     fflush(stdout);
39     clear_stdin();
40     // clear_last_lines(ERROR_CLEAR_LINE_COUNT); avoid clearing so that it can be
41     // seen in report
42 }
43
44 void handle_error(const char *format, ...) {
45     va_list args;
46     va_start(args, format);
47
48     printf("\n" RED "ПОМИЛКА!");
49     vprintf(format, args);
50     printf(RESET "\n");
51 }

```

```

52  va_end(args);
53  continue_after_error();
54  }
55
56  void handle_error_overflow() {
57      handle_error("Число_поза_допустимими_межами_значень_типу!");
58  }
59
60  void handle_error_not_precise(int max_significant_digits) {
61      handle_error(
62          "Кількість_значущих_цифр_перевищує_максимальну_дозволену_кількість_цифр_"
63          "%d,_тому_розрахунки_стануть_неточними!",
64          max_significant_digits);
65  }
66
67  void handle_error_overlength(int max_char_count) {
68      handle_error(
69          "Довжина_значення_в_символах_має_бути_більше_0_та_меншою_рівною_%d.",
70          max_char_count);
71  }
72
73  void handle_error_memory_allocation(const char *reason) {
74      handle_error("Не_вдалося_виділити_достатньо_пам'яті'_для_%s.", reason);
75  }
76
77  void handle_error_not_number() {
78      handle_error("Значення_має_бути_числом_і_не_містити_додаткових_символів!");
79  }
80
81  void show_warning(const char *format, ...) {
82      va_list args;
83      va_start(args, format);
84
85      printf("\n" YELLOW "УВАГА!_");
86      vprintf(format, args);
87      printf(RESET "\n");
88
89      va_end(args);
90  }
91
92  void show_success(const char *format, ...) {
93      va_list args;
94      va_start(args, format);
95
96      printf("\n" GREEN "ПЕРЕМОГА!_");
97      vprintf(format, args);
98      printf(RESET "\n");
99
100     va_end(args);
101 }
102
103 // INPUT TAKING
104 bool is_agree() {
105     char choice;
106
107     printf(
108         "Введіть_+,_якщо_погоджуєтесь._Інакше_введіть_будьяку_"
109         "іншу_клавішу:_");
110
111     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

```
1 #ifndef INT_H
2 #define INT_H
3
4 #include "common/types.h"
5
6 void print_int_range(const char *name, const IntRange *range);
7 void print_int_character_count_range();
8 int read_int_within_range_with_validation(const char *prompt,
9                                           const IntRange *range,
10                                           IntValidation additional_check, ...);
11
12 #endif // !INT_H
```

../src/io/int.c

```
1 #include <limits.h>
2 #include <stdbool.h>
3 #include <stdio.h>
4 #include <stdlib.h>
5
6 #include "common/constants.h"
7 #include "common/types.h"
8 #include "utils.h"
9
10 #define MAX_VALUE (long)INT_MAX
11 #define MIN_VALUE (long)INT_MIN
12 #define MAX_CHAR_COUNT_INT 10 // TODO: make it a little bit more usable
13
14 const IntRange INT_CHAR_COUNT_RANGE = {1, MAX_CHAR_COUNT_INT, true, true};
15
16 void show_prompt_within_range(const char *prompt, const IntRange *range) {
17     printf("%s від (%d до %d): ", prompt, range->min, range->max);
18 }
19
20 void show_range_error_int(RangeCheckResult result, const IntRange *range) {
21     switch (result) {
22         case LESS:
23             handle_error("Значення має бути ≥ %d.", range->min);
24             break;
25         case LESS_EQUAL:
26             handle_error("Значення має бути > %d.", range->min);
27             break;
28         case GREATER:
29             handle_error("Значення має бути ≤ %d.", range->max);
30             break;
31         case GREATER_EQUAL:
32             handle_error("Значення має бути < %d.", range->max);
33             break;
34         default:
35             break;
36     }
37 }
38
39 // Range Checking Functions
40 RangeCheckResult validate_range_int(int value, const IntRange *range) {
41     if (range->is_min_included && value < range->min) return LESS;
42     if (!range->is_min_included && value ≤ range->min) return LESS_EQUAL;
```

```

43     if (range->is_max_included && value > range->max) return GREATER;
44     if (!range->is_max_included && value ≥ range->max) return GREATER_EQUAL;
45     return WITHIN_RANGE;
46 }
47
48 void print_int_range(const char *name, const IntRange *range) {
49     printf("%d", range->min);
50     if (range->is_min_included) {
51         printf("_≤_");
52     } else {
53         printf("_<_");
54     }
55
56     printf("%s", name);
57
58     if (range->is_max_included) {
59         printf("_≤_");
60     } else {
61         printf("_<_");
62     }
63
64     printf("%d\n", range->max);
65 }
66
67 void print_int_character_count_range() {
68     print_int_range("кількість_символів_у_цілих_числах", &INT_CHAR_COUNT_RANGE);
69 }
70
71 RangeCheckResult validate_overflow_int(long int value) {
72     if (value > MAX_VALUE) return GREATER;
73     if (value < MIN_VALUE) return LESS;
74     return WITHIN_RANGE;
75 }
76
77 int read_int_quiet(int *value, const char *prompt) {
78     char input[MAX_CHAR_COUNT_INT + 2];
79
80     if (read_input(input, MAX_CHAR_COUNT_INT) == FAILURE) {
81         return FAILURE;
82     }
83
84     if (!is_input_within_length(input)) {
85         handle_error_overlength(MAX_CHAR_COUNT_INT);
86         clear_stdin();
87         return FAILURE;
88     }
89
90     char *endptr;
91     long int temp_value = strtol(input, &endptr, 10);
92
93     if (!is_input_number_after_conversion(endptr, input)) {
94         handle_error_not_number();
95         return FAILURE;
96     }
97
98     RangeCheckResult global_check = validate_overflow_int(temp_value);
99     if (global_check ≠ WITHIN_RANGE) {
100         handle_error_overflow();
101         return FAILURE;
102     }

```

```

103
104 *value = (int)temp_value;
105
106 return SUCCESS;
107 }
108
109 int read_int(int *value, const char *prompt) {
110     show_prompt(prompt);
111     return read_int_quiet(value, prompt);
112 }
113
114 int read_int_within_range(int *value, const char *prompt,
115                           const IntRange *range) {
116     show_prompt_within_range(prompt, range);
117     if (read_int_quiet(value, prompt) == FAILURE) {
118         return FAILURE;
119     };
120
121     RangeCheckResult range_check = validate_range_int(*value, range);
122     if (range_check != WITHIN_RANGE) {
123         show_range_error_int(range_check, range);
124         return FAILURE;
125     }
126     return SUCCESS;
127 }
128
129 int read_int_within_range_with_validation(const char *prompt,
130                                          const IntRange *range,
131                                          IntValidation additional_check, ...) {
132     int value;
133     bool is_valid = false;
134
135     va_list args;
136
137     while (!is_valid) {
138         if (read_int_within_range(&value, prompt, range) == SUCCESS) {
139             if (additional_check != NULL) {
140                 va_start(args, additional_check);
141                 is_valid = additional_check(value, args);
142                 va_end(args);
143             } else {
144                 is_valid = true;
145             }
146         }
147     }
148 }
149
150 return value;
151 }

```

../src/io/double.h

```

1 #ifndef DOUBLE_H
2 #define DOUBLE_H
3
4 #include "common/types.h"
5
6 void print_double_range(const char *name, const DoubleRange *range);
7 void print_double_character_count_range();

```



```

8 void print_double_precision_limit();
9 void print_truncated_double(double num, int decimal_places);
10 int read_double_within_range(double *value, const char *prompt,
11                             const DoubleRange *range);
12 double read_double_within_range_with_validation(
13     const char *prompt, const DoubleRange *range,
14     DoubleValidation additional_check, ...);
15
16 #endif

```

../src/io/double.c

```

1 #include <errno.h>
2 #include <float.h>
3 #include <math.h>
4 #include <stdio.h>
5 #include <stdlib.h>
6
7 #include "common/constants.h"
8 #include "common/types.h"
9 #include "int.h"
10 #include "utils.h"
11
12 #define TOLERANCE DBL_EPSILON
13 #define RELATIVE_TOLERANCE 1e-12
14 #define MIN_ABSOLUTE_VALUE DBL_MIN
15 #define MAX_ABSOLUTE_VALUE DBL_MAX
16 #define MAX_CHAR_COUNT_DOUBLE 23
17 #define OVERFLOW_ABSOLUTE_VALUE HUGE_VAL
18 #define MAX_SIGNIFICANT_DIGITS DBL_DIG
19
20 const IntRange DOUBLE_CHAR_COUNT_RANGE = {1, MAX_CHAR_COUNT_DOUBLE, true, true};
21
22 // OUTPUT
23 void show_prompt_double_within_range(const char *prompt,
24                                     const DoubleRange *range) {
25     printf("%s_від(%%lg_до_%%lg):_", prompt, range->min, range->max);
26 }
27
28 void show_range_error_double(RangeCheckResult result,
29                             const DoubleRange *range) {
30     switch (result) {
31         case LESS:
32             handle_error("Значення_має_бути_≥_%%lg.", range->min);
33             break;
34         case LESS_EQUAL:
35             handle_error("Значення_має_бути_>_%%lg.", range->min);
36             break;
37         case GREATER:
38             handle_error("Значення_має_бути_≤_%%lg.", range->max);
39             break;
40         case GREATER_EQUAL:
41             handle_error("Значення_має_бути_<_%%lg.", range->max);
42             break;
43         default:
44             break;
45     }
46 }

```

```

47
48 // RANGES
49 RangeCheckResult validate_range_double(double value, const DoubleRange *range)
50 {
51     if (range->is_min_included && value < range->min - TOLERANCE) return LESS;
52     if (!range->is_min_included && value ≤ range->min + TOLERANCE)
53         return LESS_EQUAL;
54     if (range->is_max_included && value > range->max + TOLERANCE) return GREATER;
55     if (!range->is_max_included && value ≥ range->max - TOLERANCE)
56         return GREATER_EQUAL;
57     return WITHIN_RANGE;
58 }
59 void print_double_range(const char *name, const DoubleRange *range) {
60     printf("%g", range->min);
61     if (range->is_min_included) {
62         printf("_≤_");
63     } else {
64         printf("_<_");
65     }
66     printf("%s", name);
67     if (range->is_max_included) {
68         printf("_≤_");
69     } else {
70         printf("_<_");
71     }
72     printf("%g\n", range->max);
73 }
74 void print_double_character_count_range() {
75     print_int_range("кількість_символів_у_дійсних_числах",
76                     &DOUBLE_CHAR_COUNT_RANGE);
77 }
78 void print_double_precision_limit() {
79     printf("Будьякі_дійсні_числа_приймаються_з_точністю,_не_меншою_за_%g",
80           TOLERANCE);
81 }
82 RangeCheckResult validate_overflow_double(double value) {
83     if (errno == ERANGE) {
84         if (fabs(value) == OVERFLOW_ABSOLUTE_VALUE) return GREATER_EQUAL;
85         if (fabs(value) < MIN_ABSOLUTE_VALUE) return LESS_EQUAL;
86     }
87     return WITHIN_RANGE;
88 }
89 // UTILS
90 double truncate_to_precision_double(double num, int decimal_places) {
91     double factor = pow(10, decimal_places);
92     return trunc(num * factor) / factor;
93 }
94 void print_truncated_double(double num, int decimal_places) {
95     printf("%.*lf", decimal_places,
96           truncate_to_precision_double(num, decimal_places));
97 }
98
99

```

```

106 bool is_double_precise(double value) {
107     if (isinf(value) || isnan(value)) {
108         return false;
109     }
110
111     if (value == 0.0) {
112         return true;
113     }
114
115     double scale = pow(10, MAX_SIGNIFICANT_DIGITS);
116     double scaled_value = round(value * scale);
117     double rounded_value = scaled_value / scale;
118
119     double tolerance = fabs(value) * RELATIVE_TOLERANCE;
120
121     return fabs(value - rounded_value) < tolerance;
122 }
123
124 // INPUT
125
126 int read_double_quiet(double *value, const char *prompt) {
127     char input[MAX_CHAR_COUNT_DOUBLE + 2];
128     errno = 0;
129
130     if (read_input(input, MAX_CHAR_COUNT_DOUBLE) == FAILURE) {
131         return FAILURE;
132     }
133
134     if (!is_input_within_length(input)) {
135         clear_stdin();
136         handle_error_overlength(MAX_CHAR_COUNT_DOUBLE);
137         return FAILURE;
138     }
139
140     replace_commas_with_dots(input);
141
142     char *endptr;
143     *value = strtod(input, &endptr);
144
145     if (!is_input_number_after_conversion(endptr, input)) {
146         handle_error_not_number();
147         return FAILURE;
148     }
149
150     RangeCheckResult global_check = validate_overflow_double(*value);
151     if (global_check != WITHIN_RANGE) {
152         handle_error_overflow();
153         return FAILURE;
154     }
155
156     if (!is_double_precise(*value)) {
157         handle_error_not_precise(MAX_SIGNIFICANT_DIGITS);
158         return FAILURE;
159     }
160
161     return SUCCESS;
162 }
163
164 int read_double(double *value, const char *prompt) {
165     show_prompt(prompt);

```

```

166     return read_double_quiet(value, prompt);
167 }
168
169 int read_double_within_range(double *value, const char *prompt,
170                             const DoubleRange *range) {
171     show_prompt_double_within_range(prompt, range);
172
173     if (read_double_quiet(value, prompt) == FAILURE) {
174         return FAILURE;
175     }
176
177     RangeCheckResult range_check = validate_range_double(*value, range);
178     if (range_check != WITHIN_RANGE) {
179         show_range_error_double(range_check, range);
180         return FAILURE;
181     }
182     return SUCCESS;
183 }
184
185 double read_double_within_range_with_validation(
186     const char *prompt, const DoubleRange *range,
187     DoubleValidation additional_check, ...) {
188     double value;
189     bool is_valid = false;
190
191     va_list args;
192
193     while (!is_valid) {
194         if (read_double_within_range(&value, prompt, range) == SUCCESS) {
195             if (additional_check != NULL) {
196                 va_start(args, additional_check);
197                 is_valid = additional_check(value, args);
198                 va_end(args);
199             } else {
200                 is_valid = true;
201             }
202         }
203     }
204 }
205
206 return value;
207 }

```

../src/io/validators.h

```

1 #ifndef VALIDATORS_H
2 #define VALIDATORS_H
3
4 #include <stdarg.h>
5 #include <stdbool.h>
6
7 bool is_greater(double b, va_list args);
8
9 #endif // !VALIDATORS_H

```

../src/io/validators.c

```
1 #include "validators.h"
2
3 #include <stdarg.h>
4 #include <stdbool.h>
5
6 #include "utils.h"
7
8 bool is_greater(double b, va_list args) {
9     double a = va_arg(args, double);
10    bool is_greater = b > a;
11    if (!is_greater) {
12        handle_error("Верхня_межа_повинна_бути_більшою_за_нижню");
13    }
14    return is_greater;
15 }
```

../meson.build

```
1 project('op-lab-6', 'c')
2
3 c_args = ['-Isrc', '-g' ]
4 include_dirs = include_directories('src')
5
6 main_sources = [
7     'src/main.c',
8     'src/algorithm.c',
9     'src/io/int.c',
10    'src/io/double.c',
11    'src/io/validators.c',
12    'src/io/utils.c',
13    'src/io/equations.c',
14 ]
15
16 executable('main.out', main_sources,
17     c_args: c_args,
18     include_directories: include_dirs,
19     link_args: ['-lm']
20 )
```

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:

$2 \leq$ кількість рівнянь у системі ≤ 10000

$-1e+12 \leq$ мінімальне та максимальне значення коефіцієнтів у рівняннях $\leq 1e+12$

мінімальне значення коефіцієнтів $<$ максимальне значення коефіцієнтів

$2.22045e-16 \leq$ точність < 1

$1 \leq$ кількість символів у дійсних числах ≤ 23

$1 \leq$ 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 дотримуєте цього правила, отримуєте 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 коефіцієнтів (від $-1e+12$ до $1e+12$): 100

Точність (від $2.22045e-16$ до 1): $1e-12$

Чи бажаєте Ви увімкнути 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 коефіцієнтів (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 коефіцієнтів (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 коефіцієнтів (від $-1e+12$ до $1e+12$): 100

Точність (від $2.22045e-16$ до 1): $1e-10$

Чи бажаєте Ви увімкнути 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.

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

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

Випадок 2.

$$164.086 \cdot -0.559 + 75.828 \cdot 0.611 + 27.603 \cdot -0.631 = -62.8106590000000015 \approx -62.921$$

$$70.916 \cdot -0.559 + 210.448 \cdot 0.611 + 39.887 \cdot -0.631 = 63.772987000000001 \approx 63.813$$

$$21.099 \cdot -0.559 + 98.123 \cdot 0.611 + 143.394 \cdot -0.631 = -42.322802000000001 \approx -42.319$$

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.

$$\begin{aligned}x_1 &= -0.0441097134 \\x_2 &= 0.0235341127 \\x_3 &= -0.0872235223 \\x_4 &= -0.1649598249 \\x_5 &= -0.1063362650 \\x_6 &= 0.0563139343 \\x_7 &= 0.1029099953 \\x_8 &= -0.1500373963 \\x_9 &= 0.0388550855 \\x_{10} &= -0.0370713693\end{aligned}$$

Висновки: Програма працює коректно. Програма вирішує поставлене завдання.