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.
«2» січня 2025 р.

Виконав
студент групи ІП-43
Дутов І. А.

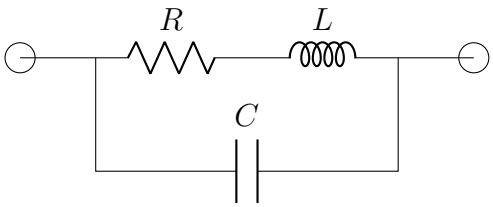
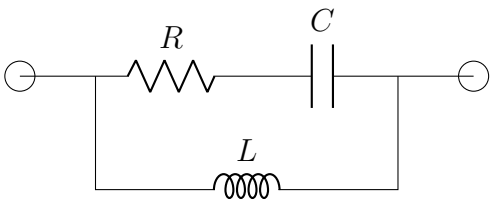
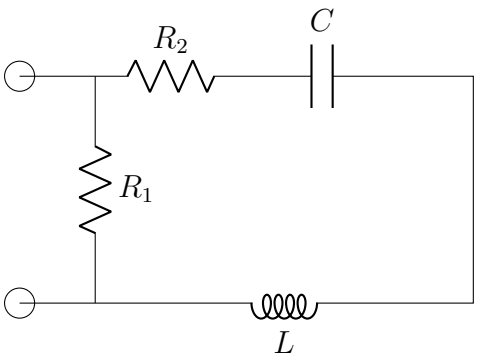
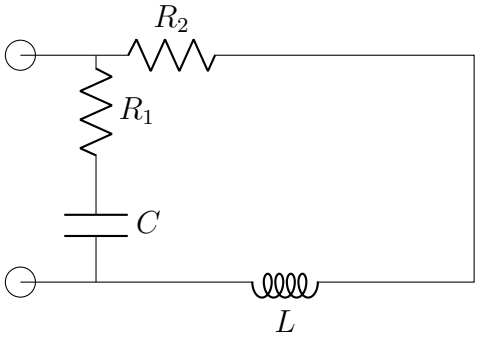
Київ 2025

Комп'ютерний практикум №8

Тема: *Структури.*

Завдання:

Написати програму для обчислення комплексного опору заданого коливального контуру в залежності від частоти струму. Варіанти коливальних контурів:

Контур 1	Контур 2
	
Контур 3	**Контур 4**
	

Переведення одиниць вимірювання здійснювати **НЕ** слід.

Текст програми

../CMakeLists.txt

```
cmake_minimum_required(VERSION 3.10)
project(FindingRoots C)

include_directories(${CMAKE_SOURCE_DIR})

set(SOURCES
    src/main.c
    src/algorithm.c
    src/io/choices.c
    src/io/utils.c
    src/io/double.c
    src/io/int.c
    src/io/validators.c
)

set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -g -fblocks")

add_executable(main ${SOURCES})
target_link_libraries(main m BlocksRuntime)
```

../src/main.h

```c
#ifndef MAIN_H
#define MAIN_H

#include "algorithm.h"
#include "common/types.h"
#include <float.h>

#define DECIMAL_PLACES DBL_MAX_DIGITS
#define COLUMN_WIDTH 10
#define R2CLR1_INDEX 2 // index of first circuit with two resistors
#define R2LCR1_INDEX 3 // index of the second circuit with two resistors
#define CIRCUIT_COUNT 4

const DoubleRange DEFAULT_RANGE = {1e-6, 1e6, true, true};

const char *CIRCUIT_DESCRIPTIONS[CIRCUIT_COUNT] = {
    "\⍰──┬────────┬──⍰nRL"
    "\n␣␣␣│␣␣␣␣␣␣␣␣␣│"
    "\n␣␣␣└────────┘C",

    "\⍰──┬────────┬──⍰nRC"
    "\n␣␣␣│␣␣␣␣␣␣␣␣␣│"
    "\n␣␣␣└────────┘L",

    "\⍰──┬────────┬──⍰nR2C"
    "\n␣␣␣R1␣␣␣␣␣␣␣␣␣│"
    "\n␣␣␣└─────────┘L",

    "\⍰──┬─────┐nR2"
    "\n␣␣␣R1␣␣␣␣␣␣│"
    "\n␣␣␣│␣␣␣␣␣␣│"
    "\n␣␣␣C␣␣␣␣␣␣␣│"
    "\⍰──┴──────┘nL",
};

const IntRange CIRCUIT_CHOICE_RANGE = {1, CIRCUIT_COUNT, true, true};
const char *CIRCUIT_PROMPT =
    "Оберіть␣електричне␣коло,␣комплексний␣опір␣якого␣ви␣хочете␣обчислити";
const CircuitSolver CIRCUIT_SOLVERS[CIRCUIT_COUNT] = {solveRLC, solveRCL,
                                                      solveR2CLR1, solveR2LCR1};

#endif
```

../src/main.c

```c
#include "main.h"
#include "common/constants.h"
#include "io/choices.h"
#include "io/double.h"
#include "io/utils.h"
#include "io/validators.h"
#include <math.h>
#include <stdio.h>

void print_demo() {
  printf("Вас␣вітає␣програма␣SuperContur,␣що␣обчислює␣комплексний␣опір␣\n"
```

```
      "4 можливих коливальних контурів у залежності від частоти струму.\n");

  showWarning("Наші обмеження:");
  printDoublePrecisionLimit();
  printDoubleCharacterCountRange();
  printDoubleRange("Індукція мГн(), Ємність мкФ(), Опір Ом(), Частота кГц()",
                   &DEFAULT_RANGE);
  printf("Мінімальна частота ≤ максимальна частота\n");
  printf("Крок частоти ≤ максимальна - мінімальна частота\n");
  printf("Крок не може бути нульовим.\n");
  showWarning("При всіх обчисленнях одиниці вимірювання не переводяться! "
              "Відповідь 100% не відповідатиме дійсності. Користуйтеся цією "
              "програмою при вимірюваннях на свій страх і ризик!");
}

double getCircuitParam(const char *prompt) {
  return readDoubleWithinRangeWithValidation(prompt, &DEFAULT_RANGE, NULL);
}

int main() {
  print_demo();
  do {
    int circuitIndex = getUserChoice(CIRCUIT_PROMPT, CIRCUIT_DESCRIPTIONS,
                                     CIRCUIT_COUNT, &CIRCUIT_CHOICE_RANGE);
    CircuitParams circuit;
    circuit.L = getCircuitParam("Індукція L мГн()");
    circuit.C = getCircuitParam("Ємність C мкФ()");

    if (circuitIndex == R2CLR1_INDEX || circuitIndex == R2LCR1_INDEX) {
      circuit.R1 = getCircuitParam("Опір R1 Ом()");
      circuit.R2 = getCircuitParam("Опір R2 Ом()");
    } else {
      circuit.R1 = getCircuitParam("Опір R Ом()");
    }

    double f_min = getCircuitParam("Мінімальна частота f_min кГц()");
    double f_max = readDoubleWithinRangeWithValidation(
        "Максимальна частота f_max кГц()", &DEFAULT_RANGE, isGreaterOrEqual,
        f_min);

    DoubleRange fRange = {0, f_max - f_min, true, true};
    double df = readDoubleWithinRangeWithValidation("Крок зміни частоти кГц()",
                                                    &fRange, NULL);

    double resonantFrequency = 1.0 / (2.0 * M_PI * sqrt(circuit.L * circuit.C))
        ;
    printf("Резонансна частота: %.*lg кГц.\n", MAX_DIGITS_DOUBLE,
           resonantFrequency);

    CircuitSolver solver = CIRCUIT_SOLVERS[circuitIndex];

    if (df == 0.0) {
      // Displaying once, cause we will be staying in place
      circuit.w = 2.0 * M_PI * f_min;
      Complex Z = solver(&circuit);
      printf("f = %.*lg кГц  Z = %.*lg + %.*lgj Ом\n", MAX_DIGITS_DOUBLE, f_min,
             MAX_DIGITS_DOUBLE, Z.Re, MAX_DIGITS_DOUBLE, Z.Im);

    } else {
      const int COUNT_PADDING = (int)log10((f_max - f_min) / df) + 1;
```

```
    int i = 1;
    double f = f_min;

    do {
      circuit.w = 2.0 * M_PI * f;
      Complex Z = solver(&circuit);
      printf("f%-*d = %-*.*lg кГц  Z%-*d = %-*.*lg + %-*.*lgj Ом\n",
           COUNT_PADDING, i, COLUMN_WIDTH, MAX_DIGITS_DOUBLE, f,
           COUNT_PADDING, i, COLUMN_WIDTH, MAX_DIGITS_DOUBLE, Z.Re,
           MAX_DIGITS_DOUBLE, COLUMN_WIDTH, Z.Im);

      f += df;
      i++;
    } while (f ⩽ f_max + TOLERANCE_DOUBLE);
   }

  } while (askQuestion("Чи хочете Ви повторити?"));

  return SUCCESS;
}
```

../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 isMinIncluded, isMaxIncluded;
} DoubleRange;

```

```c
typedef bool (*DoubleValidation)(double value, va_list args);

typedef struct {
  int min, max;
  bool isMinIncluded, isMaxIncluded;
} IntRange;

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

#endif
```

../src/algorithm.h

```c
#ifndef ALGORITHM_H
#define ALGORITHM_H

typedef struct {
  double Re;
  double Im;
} Complex;

typedef struct {
  double L, C, w;
  double R1, R2; // Use R1 for single resistance cases
} CircuitParams;

typedef Complex (*CircuitSolver)(const CircuitParams *);

Complex solveRLC(const CircuitParams *params);
Complex solveRCL(const CircuitParams *params);
Complex solveR2CLR1(const CircuitParams *params);
Complex solveR2LCR1(const CircuitParams *params);

#endif // !ALGORITHM_H
```

../src/algorithm.c

```c
#include "algorithm.h"

Complex divideComplex(Complex *numerator, Complex *denominator) {
  Complex res;
  // Creating variables to make code more readable
  double a = numerator->Re, b = numerator->Im, c = denominator->Re,
         d = denominator->Im;
  double frac = 1.0 / (c * c + d * d);

  res.Re = (a * c + b * d) * frac;
  res.Im = (b * c - a * d) * frac;

  return res;
}

// INFO: The naming of methods starts on the schematic diagram from the left
    top
// corner clockwise

Complex solveRLC(const CircuitParams *params) {
```

```
  double L = params->L, C = params->C, w = params->w, R = params->R1;

  Complex numerator, denominator;
  numerator.Re = L / C;
  numerator.Im = -R / (w * C);
  denominator.Re = R;
  denominator.Im = w * L - 1.0 / (w * C);
  return divideComplex(&numerator, &denominator);
}

Complex solveRCL(const CircuitParams *params) {
  double L = params->L, C = params->C, w = params->w, R = params->R1;

  Complex numerator, denominator;
  numerator.Re = L / C;
  numerator.Im = R / (w * C);
  denominator.Re = R;
  denominator.Im = w * L - 1.0 / (w * C);
  return divideComplex(&numerator, &denominator);
}

Complex solveR2CLR1(const CircuitParams *params) {
  double L = params->L, C = params->C, w = params->w, R1 = params->R1,
        R2 = params->R2;

  Complex numerator, denominator;
  numerator.Re = R1 * R2;
  numerator.Im = R1 * (w * L - 1.0 / (w * C));
  denominator.Re = R1 + R2;
  denominator.Im = w * L - 1.0 / (w * C);
  return divideComplex(&numerator, &denominator);
}

Complex solveR2LCR1(const CircuitParams *params) {
  double L = params->L, C = params->C, w = params->w, R1 = params->R1,
        R2 = params->R2;

  Complex numerator, denominator;
  numerator.Re = R1 * R2 + L / C;
  numerator.Im = w * L * R1 - R2 / (w * C);
  denominator.Re = R1 + R2;
  denominator.Im = w * L - 1.0 / (w * C);
  return divideComplex(&numerator, &denominator);
}
```

../src/io/int.h

```
#ifndef INT_H
#define INT_H

#include "../common/types.h"
#include <stdarg.h>
#include <stdbool.h>

#define MAX_VALUE_INT (long)INT_MAX
#define MIN_VALUE_INT (long)INT_MIN
#define MAX_CHAR_COUNT_INT 10

void printIntRange(const char *name, const IntRange *range);
```

```
13 void printIntCharacterCountRange();
14 int readIntWithinRangeWithValidation(const char *prompt, const IntRange *range,
15                                      IntValidation additionalCheck, ...);
16
17 #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 "int.h"
9  #include "utils.h"
10
11 const IntRange INT_CHAR_COUNT_RANGE = {1, MAX_CHAR_COUNT_INT, true, true};
12
13 void showPromptWithinRange(const char *prompt, const IntRange *range) {
14   printf("%s від (%d до %d): ", prompt, range->min, range->max);
15 }
16
17 void showRangeErrorInt(RangeCheckResult result, const IntRange *range) {
18   switch (result) {
19   case LESS:
20     handleError("Значення має бути ≥ %d.", range->min);
21     break;
22   case LESS_EQUAL:
23     handleError("Значення має бути > %d.", range->min);
24     break;
25   case GREATER:
26     handleError("Значення має бути ≤ %d.", range->max);
27     break;
28   case GREATER_EQUAL:
29     handleError("Значення має бути < %d.", range->max);
30     break;
31   default:
32     break;
33   }
34 }
35
36 // Range Checking Functions
37 RangeCheckResult validateRangeInt(int value, const IntRange *range) {
38   if (range->isMinIncluded && value < range->min)
39     return LESS;
40   if (!range->isMinIncluded && value ⩽ range->min)
41     return LESS_EQUAL;
42   if (range->isMaxIncluded && value > range->max)
43     return GREATER;
44   if (!range->isMaxIncluded && value ⩾ range->max)
45     return GREATER_EQUAL;
46   return WITHIN_RANGE;
47 }
48
49 void printIntRange(const char *name, const IntRange *range) {
50   printf("%d", range->min);
51   if (range->isMinIncluded) {
```

```
    printf("␣≤␣");
  } else {
    printf("␣<␣");
  }

  printf("%s", name);

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

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

void printIntCharacterCountRange() {
  printIntRange("кількість␣символів␣у␣цілих␣числах", &INT_CHAR_COUNT_RANGE);
}

RangeCheckResult validateOverflowInt(long int value) {
  if (value > MAX_VALUE_INT)
    return GREATER;
  if (value < MIN_VALUE_INT)
    return LESS;
  return WITHIN_RANGE;
}

int readIntQuiet(int *value) {
  char *input = readInput(MAX_CHAR_COUNT_INT, MAX_CHAR_COUNT_INT);

  if (input == NULL) {
    return FAILURE;
  }

  if (!isInputWithinLength(input)) {
    handleErrorOverlength(MAX_CHAR_COUNT_INT);
    clearStdin();
    free(input);
    return FAILURE;
  }

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

  if (!isInputNumberAfterConversion(endptr, input)) {
    handleErrorNotNumber();
    free(input);
    return FAILURE;
  }

  RangeCheckResult globalCheck = validateOverflowInt(tempValue);
  if (globalCheck != WITHIN_RANGE) {
    handleErrorOverflow();
    free(input);
    return FAILURE;
  }

  *value = (int)tempValue;

```

```
  free(input);
  return SUCCESS;
}

int readInt(int *value, const char *prompt) {
  showPrompt(prompt);
  return readIntQuiet(value);
}

int readIntWithinRange(int *value, const char *prompt, const IntRange *range)
    {
  showPromptWithinRange(prompt, range);
  if (readIntQuiet(value) == FAILURE) {
    return FAILURE;
  };

  RangeCheckResult rangeCheck = validateRangeInt(*value, range);
  if (rangeCheck != WITHIN_RANGE) {
    showRangeErrorInt(rangeCheck, range);
    return FAILURE;
  }
  return SUCCESS;
}

int readIntWithinRangeWithValidation(const char *prompt, const IntRange *range,
                                     IntValidation additionalCheck, ...) {
  int value;
  bool isValid = false;

  va_list args;

  while (!isValid) {
    if (readIntWithinRange(&value, prompt, range) == SUCCESS) {
      if (additionalCheck != NULL) {
        va_start(args, additionalCheck);
        isValid = additionalCheck(value, args);
        va_end(args);

      } else {
        isValid = true;
      }
    }
  }

  return value;
}

int readIntWithValidation(const char *prompt, IntValidation additionalCheck,
                          ...) {
  int value;
  bool isValid = false;

  va_list args;

  while (!isValid) {
    if (readInt(&value, prompt) == SUCCESS) {
      if (additionalCheck != NULL) {
        va_start(args, additionalCheck);
        isValid = additionalCheck(value, args);
        va_end(args);
```

```
    } else {
      isValid = true;
    }
  }
 }

 return value;
}
```

../src/io/double.h

```
#ifndef DOUBLE_H
#define DOUBLE_H

#include "../common/types.h"
#include <float.h>

#define MIN_ABS_VALUE_DOUBLE DBL_MIN
#define MAX_ABS_VALUE_DOUBLE DBL_MAX
#define MAX_CHAR_COUNT_DOUBLE DBL_MAX_10_EXP
#define INITIAL_CHAR_COUNT_DOUBLE 16
#define MAX_DIGITS_DOUBLE DBL_DIG
#define TOLERANCE_DOUBLE DBL_EPSILON
#define OVERFLOW_ABS_VALUE_DOUBLE HUGE_VAL

void printDoubleRange(const char *name, const DoubleRange *range);
void printDoubleCharacterCountRange();
void printDoublePrecisionLimit();
void printTruncatedDouble(double num, int decimalPlaces);
int readDoubleWithinRange(double *value, const char *prompt,
                    const DoubleRange *range);
double readDoubleWithinRangeWithValidation(const char *prompt,
                                  const DoubleRange *range,
                                  DoubleValidation additionalCheck,
                                  ...);

#endif
```

../src/io/double.c

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

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

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

```

```c
void showRangeErrorDouble(RangeCheckResult result, const DoubleRange *range) {
  switch (result) {
  case LESS:
    handleError("Значення має бути ≥ %lg.", range->min);
    break;
  case LESS_EQUAL:
    handleError("Значення має бути > %lg.", range->min);
    break;
  case GREATER:
    handleError("Значення має бути ≤ %lg.", range->max);
    break;
  case GREATER_EQUAL:
    handleError("Значення має бути < %lg.", range->max);
    break;
  default:
    printf("Значення знаходиться в заданому проміжку.");
    break;
  }
}

// RANGES
RangeCheckResult validateRangeDouble(double value, const DoubleRange *range) {
  if (range->isMinIncluded && value < range->min)
    return LESS;
  if (!range->isMinIncluded && value ⩽ range->min)
    return LESS_EQUAL;
  if (range->isMaxIncluded && value > range->max)
    return GREATER;
  if (!range->isMaxIncluded && value ⩾ range->max)
    return GREATER_EQUAL;
  return WITHIN_RANGE;
}

void printDoubleRange(const char *name, const DoubleRange *range) {
  printf("%g %s %s %s %g\n", range->min, (range->isMinIncluded) ? "≤" : "<",
         name, (range->isMaxIncluded) ? "≤" : "<", range->max);
}

void printDoubleCharacterCountRange() {
  printf("0 < Кількість символів у дійсних числах ≤ %d\n",
         MAX_CHAR_COUNT_DOUBLE);
}

void printDoublePrecisionLimit() {
  printf("Будьякі- дійсні числа приймаються з точністю, не меншою за %g\n",
         TOLERANCE_DOUBLE);
}

RangeCheckResult validateOverflowDouble(double value) {
  if (errno == ERANGE) {
    if (fabs(value) == OVERFLOW_ABS_VALUE_DOUBLE)
      return GREATER_EQUAL;
    if (fabs(value) < MIN_ABS_VALUE_DOUBLE)
      return LESS_EQUAL;
  }
  return WITHIN_RANGE;
}

// UTILS
double truncateToPrecisionDouble(double num, int decimalPlaces) {
```

```c
  double factor = pow(10.0, (double)decimalPlaces);
  return trunc(num * factor) / factor;
}

void printTruncatedDouble(double num, int decimalPlaces) {
  printf("%.*lf", decimalPlaces, truncateToPrecisionDouble(num, decimalPlaces))
      ;
}

bool isDoublePrecise(double value) {
  if (isinf(value) || isnan(value)) {
    return false;
  }

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

  double scale = pow(10.0, MAX_DIGITS_DOUBLE);
  double scaledValue = round(value * scale);
  double roundedValue = scaledValue / scale;

  double tolerance = fabs(value) * TOLERANCE_DOUBLE;

  return fabs(value - roundedValue) < tolerance;
}

// INPUT

int readDoubleQuiet(double *value) {
  errno = 0;

  char *input = readInput(INITIAL_CHAR_COUNT_DOUBLE, MAX_CHAR_COUNT_DOUBLE);

  if (input == NULL) {
    return FAILURE;
  }

  replaceCommasWithDots(input);

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

  if (!isInputNumberAfterConversion(endptr, input)) {
    handleErrorNotNumber();
    free(input);
    return FAILURE;
  }

  RangeCheckResult globalCheck = validateOverflowDouble(*value);
  if (globalCheck != WITHIN_RANGE) {
    handleErrorOverflow();
    free(input);
    return FAILURE;
  }

  if (!isDoublePrecise(*value)) {
    warnNotPrecise(MAX_DIGITS_DOUBLE);
  }

```

```
  free(input);

  return SUCCESS;
}

int readDouble(double *value, const char *prompt) {
  showPrompt(prompt);
  return readDoubleQuiet(value);
}

int readDoubleWithinRange(double *value, const char *prompt,
                          const DoubleRange *range) {
  showPromptDoubleWithinRange(prompt, range);

  if (readDoubleQuiet(value) == FAILURE) {
    return FAILURE;
  }

  RangeCheckResult rangeCheck = validateRangeDouble(*value, range);
  if (rangeCheck != WITHIN_RANGE) {
    showRangeErrorDouble(rangeCheck, range);
    return FAILURE;
  }
  return SUCCESS;
}

double readDoubleWithinRangeWithValidation(const char *prompt,
                                   const DoubleRange *range,
                                   DoubleValidation additionalCheck,
                                   ...) {
  double value;
  bool isValid = false;

  va_list args;

  while (!isValid) {
    if (readDoubleWithinRange(&value, prompt, range) == SUCCESS) {
      if (additionalCheck != NULL) {
        va_start(args, additionalCheck);
        isValid = additionalCheck(value, args);
        va_end(args);

      } else {
        isValid = true;
      }
    }
  }

  return value;
}
```

../src/io/validators.h

```
#ifndef VALIDATORSH
#define VALIDATORSH

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

```

```
bool isGreaterOrEqual(double b, va_list args);
bool isLess(double b, va_list args);
#endif // !VALIDATORSH
```

../src/io/validators.c

```
#include "validators.h"

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

#include "utils.h"

// Validation functions

bool isGreaterOrEqual(double b, va_list args) {
  double a = va_arg(args, double);
  if (b < a) {
    handleError("Верхня межа повинна бути більшою за нижню");
    return false;
  }
  return true;
}

bool isLess(double b, va_list args) {
  double a = va_arg(args, double);
  if (b ⩾ a) {
    handleError("Значення має бути < %g", a);
    return false;
  }
  return true;
}
```

../src/io/choices.h

```
#ifndef CHOICES_H
#define CHOICES_H

#include "../common/types.h"
void showChoices(const char **choices, int choice_count);
int getUserChoice(const char *prompt, const char *choices[], int choice_count,
                  const IntRange *range);

#endif
```

../src/io/choices.c

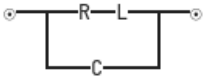
```
#include "../common/types.h"
#include "int.h"
#include "utils.h"
#include <stddef.h>
#include <stdio.h>

void showChoices(const char **choices, int choice_count) {
  for (int i = 0; i < choice_count; i++) {
```

```
    printf("%d. %s\n", i + 1, choices[i]);
  }
}

int getUserChoice(const char *prompt, const char *choices[], int choice_count,
                  const IntRange *range) {
  printf("\n");
  showPrompt(prompt);
  printf("\n");
  showChoices(choices, choice_count);
  printf("\n");

  int choice = readIntWithinRangeWithValidation("Оберіть опцію", range, NULL);
  return choice - 1;
}
```

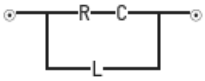
Введені та одержані результати

Контур 1

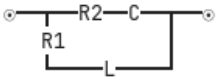
```
Вас вітає програма SuperContur, що обчислює комплексний опір
4 можливих коливальних контурів у залежності від частоти струму.

УВАГА! Наші обмеження:
Будь-які дійсні числа приймаються з точністю, не меншою за 2.22045e-16
0 < Кількість символів у дійсних числах ≤ 308
1e-06 ≤ Індукція (мГн), Ємність (мкФ), Опір (Ом), Частота (кГц) ≤ 1e+06
Мінімальна частота ≤ максимальна частота
Крок частоти ≤ максимальна - мінімальна частота
Крок не може бути нульовим.

УВАГА! При всіх обчисленнях одиниці вимірювання не переводяться! Відповідь 100% не відповідатиме дійсності
 Користуйтеся цією програмою при вимірюваннях на свій страх і ризик!

Оберіть електричне коло, комплексний опір якого ви хочете обчислити:
1.
```



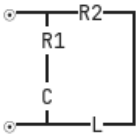

```
2.
```




```
3.
```




```
4.
```




```
Оберіть опцію (від 1 до 4): 1
Індукція L (мГн) (від 1e-06 до 1e+06): 36
Ємність C (мкФ) (від 1e-06 до 1e+06): 20.5
Опір R (Ом) (від 1e-06 до 1e+06): 12
Мінімальна частота f_min (кГц) (від 1e-06 до 1e+06): 1
Максимальна частота f_max (кГц) (від 1e-06 до 1e+06): 4
Крок зміни частоти (кГц) (від 0 до 3): 1
Резонансна частота: 0.00585857441599779 кГц.
f1 = 1          кГц  Z1 = 1.40980219751981e-08 + -0.007763921493j Ом
f2 = 2          кГц  Z2 = 8.82939698075807e-10 + -0.003881861166j Ом
f3 = 3          кГц  Z3 = 1.74474334462341e-10 + -0.00258789512 j Ом
f4 = 4          кГц  Z4 = 5.52121372391143e-11 + -0.001940918103j Ом

Чи хочете Ви повторити?
Введіть +, якщо погоджуєтесь. Інакше введіть будь-яку іншу клавішу: -
```

Контур 2

```
Оберіть опцію (від 1 до 4): 2
Індукція L (мГн) (від 1e-06 до 1e+06): 25.3
Ємність C (мкФ) (від 1e-06 до 1e+06): 1
Опір R (Ом) (від 1e-06 до 1e+06): 1e3
Мінімальна частота f_min (кГц) (від 1e-06 до 1e+06): 0.3
Максимальна частота f_max (кГц) (від 1e-06 до 1e+06): 0.4
Крок зміни частоти (кГц) (від 0 до 0.1): 0.02
Резонансна частота: 0.031641704549874 кГц.
f1 = 0.3        кГц  Z1 = 0.0502068942396984 + 0.5281487771   j Ом
f2 = 0.32       кГц  Z2 = 0.0502251989998383 + 0.4948292881   j Ом
f3 = 0.34       кГц  Z3 = 0.0502366603333302 + 0.4654111007   j Ом
f4 = 0.36       кГц  Z4 = 0.050242540411478 + 0.4392440337   j Ом
f5 = 0.38       кГц  Z5 = 0.0502437788975905 + 0.4158147881   j Ом
f6 = 0.4        кГц  Z6 = 0.0502410872192639 + 0.3947127265   j Ом
```

Контур 3

```
Оберіть опцію (від 1 до 4): 3
Індукція L (мГн) (від 1e-06 до 1e+06): 10
Ємність C (мкФ) (від 1e-06 до 1e+06): 2.53
Опір R1 (Ом) (від 1e-06 до 1e+06): 1000
Опір R2 (Ом) (від 1e-06 до 1e+06): 2000
Мінімальна частота f_min (кГц) (від 1e-06 до 1e+06): 0.1
Максимальна частота f_max (кГц) (від 1e-06 до 1e+06): 10
Крок зміни частоти (кГц) (від 0 до 9.9): 0.7
Резонансна частота: 0.031641704549874 кГц.
f1  = 0.1        кГц  Z1  = 666.667850699849 + 0.6282327002   j Ом
f2  = 0.8        кГц  Z2  = 666.759926484651 + 5.574756371    j Ом
f3  = 1.5        кГц  Z3  = 666.995036962144 + 10.45700426    j Ом
f4  = 2.2        кГц  Z4  = 667.372563062146 + 15.32320165    j Ом
f5  = 2.9        кГц  Z5  = 667.891535986739 + 20.1690225     j Ом
f6  = 3.6        кГц  Z6  = 668.550629469475 + 24.98876318    j Ом
f7  = 4.3        кГц  Z7  = 669.348168127419 + 29.77655739    j Ом
f8  = 5          кГц  Z8  = 670.282138022642 + 34.52659099    j Ом
f9  = 5.7        кГц  Z9  = 671.350199244796 + 39.23317536    j Ом
f10 = 6.4        кГц  Z10 = 672.549700363913 + 43.89078658    j Ом
f11 = 7.1        кГц  Z11 = 673.877694588272 + 48.49409294    j Ом
f12 = 7.8        кГц  Z12 = 675.33095744688 + 53.03797688    j Ом
f13 = 8.5        кГц  Z13 = 676.906005803317 + 57.51755366    j Ом
f14 = 9.2        кГц  Z14 = 678.599117998186 + 61.9281873     j Ом
f15 = 9.9        кГц  Z15 = 680.406354911174 + 66.26550421    j Ом
```

Контур 4

```
Оберіть опцію (від 1 до 4): 4
Індукція L (мГн) (від 1e-06 до 1e+06): 30
Ємність C (мкФ) (від 1e-06 до 1e+06): 16.7
Опір R1 (Ом) (від 1e-06 до 1e+06): 5
Опір R2 (Ом) (від 1e-06 до 1e+06): 7
Мінімальна частота f_min (кГц) (від 1e-06 до 1e+06): 1
Максимальна частота f_max (кГц) (від 1e-06 до 1e+06): 3
Крок зміни частоти (кГц) (від 0 до 2): 1
Резонансна частота: 0.007110518467412 77 кГц.
f1 = 1          кГц  Z1 = 4.99209341257736 + 0.1226017328   j Ом
f2 = 2          кГц  Z2 = 4.99801749958529 + 0.06148711381  j Ом
f3 = 3          кГц  Z3 = 4.99911840523888 + 0.04101448343  j Ом
```

Перевірка

Контур	Точна частота f_0 (20 знаків)	Похибка f_0
1	0.00585857441599778588	$-4.11725485973641672236 \cdot 10^{-18}$
2	0.03164170454987403931	$3.93107350757087349964 \cdot 10^{-17}$
3	0.03164170454987403931	$3.93107350757087349964 \cdot 10^{-17}$
4	0.00711051846741277015	$1.48953750247194067709 \cdot 10^{-19}$

Як бачимо, похибка f_0 залишається відповідною до стандартної похибки `double`.

Контур 1

Ітерація	Частота f	Точне $\Re(Z)$ (до 20 знаків)	Точне $\Im(Z)$ (до 20 знаків)
1	1	0.00000001409802197518	-0.00776392149291284873
2	2	0.00000000088293969809	-0.00388186116604694071
3	3	0.00000000017447433445	-0.00258789511978029926
4	4	$5.52121372398586064427 \cdot 10^{-11}$	-0.0019409181030332087

Ітерація	Частота f	Похибка $\Re(Z)$	Похибка $\Im(Z)$
1	1	$-1.76904226348413114325 \cdot 10^{-20}$	$8.71512665107231285046 \cdot 10^{-14}$
2	2	$1.19021431981047963673 \cdot 10^{-20}$	$-4.69407142948570390907 \cdot 10^{-14}$
3	3	$-8.78864913442258524997 \cdot 10^{-21}$	$2.19700736159491981619 \cdot 10^{-13}$
4	4	$7.44306442692903014374 \cdot 10^{-22}$	$-3.32087046658783033601 \cdot 10^{-14}$

Контур 2

Ітерація	Частота f	Точне $\Re(Z)$ (до 20 знаків)	Точне $\Im(Z)$ (до 20 знаків)
1	0.3	0.05020689423969842992	0.52814877707627276884
2	0.32	0.05022519899983832007	0.4948292881412661826
3	0.34	0.05023666033333016152	0.46541110071089034862
4	0.36	0.05024254041147799669	0.4392440337136812447
5	0.38	0.05024377889759046586	0.41581478813400051174
6	0.4	0.05024108721926390659	0.39471272652552947396

Ітерація	Частота f	Похибка $\Re(Z)$	Похибка $\Im(Z)$
1	0.3	$2.99184644322130205045 \cdot 10^{-17}$	$-2.37272311590389109593 \cdot 10^{-11}$
2	0.32	$2.00716028753072704784 \cdot 10^{-17}$	$4.12661825989502437927 \cdot 10^{-11}$
3	0.34	$-3.84776790206111648643 \cdot 10^{-17}$	$1.08903486237919056849 \cdot 10^{-11}$
4	0.36	$-3.31274905974073736596 \cdot 10^{-18}$	$1.36812447004242308868 \cdot 10^{-11}$
5	0.38	$-3.41420991110137382798 \cdot 10^{-17}$	$3.40005117415388220274 \cdot 10^{-11}$
6	0.4	$6.58887685126423257842 \cdot 10^{-18}$	$2.55294739639829211481 \cdot 10^{-11}$

Контур 3

Ітерація	Частота f	Точне $\Re(Z)$ (до 20 знаків)	Точне $\Im(Z)$ (до 20 знаків)
1	0.1	666.66785069984931009112	0.62823270021006591314
2	0.8	666.75992648465116069501	5.57475637146421330734
3	1.5	666.99503696214372100347	10.457004260688929464 67
4	2.2	667.3725630621462118375	15.32320164887321154815
5	2.9	667.89153598673879704263	20.16902249753623605242
6	3.6	668.55062946947550440404	24.9887631765542152486
7	4.3	669.34816812741890413929	29.7765573928002872621
8	5	670.28213802264163923358	34.52659099205203435966
9	5.7	671.3501992447965169777	39.23317535750681236979
10	6.4	672.54970036391283943468	43.89078658366077801414
11	7.1	673.87769458827222372597	48.49409294112370610103
12	7.8	675.33095744688060852292	53.03797688462372243323
13	8.5	676.90600580331681983838	57.51755366490055749605
14	9.2	678.59917799818567511242	61.92818730105002078471
15	9.9	680.4063549111737033273	66.26550421181522028304

Ітерація	Частота f	Похибка $\Re(Z)$	Похибка $\Im(Z)$
1	0.1	$3.10091123677341466774 \cdot 10^{-13}$	$1.00659131368242024298 \cdot 10^{-11}$
2	0.8	$1.60695013194950162142 \cdot 10^{-13}$	0.00000000046421330734
3	1.5	$-2.78996534724557438382 \cdot 10^{-13}$	0.00000000068892946467
4	2.2	$2.11837502515885589345 \cdot 10^{-13}$	-0.00000000112678845185
5	2.9	$-2.02957368409468302345 \cdot 10^{-13}$	-0.00000000246376394758
6	3.6	$5.04404044948807686289 \cdot 10^{-13}$	-0.0000000034457847514
7	4.3	$-9.58607102806125355845 \cdot 10^{-14}$	0.0000000028002872621
8	5	$-3.60766416294420769694 \cdot 10^{-13}$	0.00000000205203435966
9	5.7	$5.16977700898573142364 \cdot 10^{-13}$	-0.00000000249318763021
10	6.4	$-1.60565317184232574218 \cdot 10^{-13}$	0.00000000366077801414
11	7.1	$2.23725971166320783997 \cdot 10^{-13}$	0.00000000112370610103
12	7.8	$6.08522924413416966836 \cdot 10^{-13}$	0.00000000462372243323
13	8.5	$-1.80161621693370181465 \cdot 10^{-13}$	0.00000000490055749605
14	9.2	$-3.24887582651685971525 \cdot 10^{-13}$	0.00000000105002078471
15	9.9	$-2.9667269523717640259 \cdot 10^{-13}$	0.00000000181522028304

Контур 4

Ітерація	Частота f	Точне $\Re(Z)$ (до 20 знаків)	Точне $\Im(Z)$ (до 20 знаків)
1	1	4.99209341257735594866	0.12260173282042317696
2	2	4.99801749958528704525	0.06148711380585848756
3	3	4.99911840523888026289	0.041014483429299555

Ітерація	Частота f	Похибка $\Re(Z)$	Похибка $\Im(Z)$
1	1	$-4.05134001348232516157 \cdot 10^{-15}$	$2.04231769602624518845 \cdot 10^{-11}$
2	2	$-2.95474548707521296378 \cdot 10^{-15}$	$-4.14151243865350025309 \cdot 10^{-12}$
3	3	$2.62893078048372091875 \cdot 10^{-16}$	$-7.00444997061909216369 \cdot 10^{-13}$

Як бачимо, похибка $\Im(Z)$ часто виявляється суттєвішою за похибку $\Re(Z)$, що можна пояснити наявністю доданків типу $\frac{1}{wC}$ у формулах розрахунку Z. Водночас точність $\Re(Z)$ сильно просідає у прикладі 3 контуру, що викликано великими значеннями R_1 та R_2 ($1\,k\Omega$ та $2\,k\Omega$ відповідно).

Висновки: Програма працює коректно. Програма вирішує поставлене завдання.