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.
«1» січня 2025 р.

Виконав
студент групи ІП-43
Дутов І. А.

Київ 2025

Комп'ютерний практикум №7

Тема: *Функції та покажчики на функції.*

Завдання:

Написати програму для обчислення коренів нелінійних рівнянь на заданому проміжку.

$$\cos\left(\frac{y}{x}\right) - 2\sin\left(\frac{1}{x}\right) + \frac{1}{x} = 0,$$

$$\sin(\ln x) - \cos(\ln x) + y\ln x = 0, \text{ де } x \in [a_1, a_2];\ t, a_1, a_2 \in \mathbb{R}.$$

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

```
#ifndef MAIN_H
#define MAIN_H

#include "algorithm.h"
#include "common/types.h"
#include "io/double.h"

#define YX_INDEX 0
#define YLNX_INDEX 1
#define MAX_ABS_DOUBLE_VALUE 1e6

#define OPTION_START 1

const DoubleRange DOUBLE_RANGE = {-MAX_ABS_DOUBLE_VALUE, MAX_ABS_DOUBLE_VALUE,
                                  true, true};
const DoubleRange YLNX_RANGE = {0, MAX_ABS_DOUBLE_VALUE, false, true};
const DoubleRange PRECISION_RANGE = {TOLERANCE_DOUBLE, 1.0, true, false};

#define EQUATION_COUNT 2
const char *EQUATION_PROMPT = "Оберіть функцію для розрахунку";
const IntRange EQUATION_CHOICE_RANGE = {
```

```
   OPTION_START, EQUATION_COUNT + OPTION_START - 1, true, true};
const char *EQUATION_DESCRIPTIONS[EQUATION_COUNT] = {
   "cos(y / x) - 2 * sin(1 / x) + 1 / x",
   "y * ln(x) + sin(ln(x)) - cos(ln(x))"};
const EquationTemplate EQUATION_FUNCTION_TEMPLATES[EQUATION_COUNT] = {
   &yxTemplate, &ylnxTemplate};
const char *EQUATION_TEMPLATES[EQUATION_COUNT] = {
   "cos(%g / %.*lf) - 2 * sin(1 / %.*lf) + 1 / %.*lf",
   "%g * ln(%.*lf) + sin(ln(%.*lf)) - cos(ln(%.*lf))",
};

#define METHOD_COUNT 2
const char *METHOD_PROMPT = "Оберіть метод для розрахунку функції";
const IntRange METHOD_CHOICE_RANGE = {
   OPTION_START, METHOD_COUNT + OPTION_START - 1, true, true};
const char *METHOD_DESCRIPTIONS[METHOD_COUNT] = {"Метод половинного ділення",
                                                 "Метод Ньютона"};
const EquationMethod METHODS[METHOD_COUNT] = {&bisectorMethod, &newtonMethod};

const char *OPTION_PROMPT = "Введіть опцію";

#endif
```

../src/main.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 <Block.h>
#include <math.h>
#include <stdio.h>
#include "algorithm.h"

void print_demo() {
  printf("Вас вітає програма для знаходження коренів двох рівнянь:\n");
  showChoices(EQUATION_DESCRIPTIONS, EQUATION_COUNT);

  printf("\зап допомогою двох методів:\n");
  showChoices(METHOD_DESCRIPTIONS, METHOD_COUNT);
  printf("на дійсному проміжку [a, b]\n");

  showWarning("Приймаються тільки проміжки, що не містять точок розриву "
           "заданих функцій.");

  showWarning("Наші обмеження:");
  printDoubleRange("Нижня межа a, верхня межа b, параметр y", &DOUBLE_RANGE);
  printDoubleRange("Точність", &PRECISION_RANGE);
  printDoublePrecisionLimit();
  printDoubleCharacterCountRange();
  printf("\n");
}

int main() {
  print_demo();
```

```
  do {
    // First reading function because input boundaries depend on it
    int funcIndex = getUserChoice(EQUATION_PROMPT, EQUATION_DESCRIPTIONS,
                                  EQUATION_COUNT, &EQUATION_CHOICE_RANGE);

    EquationTemplate template = EQUATION_FUNCTION_TEMPLATES[funcIndex];

    double start, end;

    switch (funcIndex) {
    case YX_INDEX:
      start = readDoubleWithinRangeWithValidation("Нижня межа a", &DOUBLE_RANGE,
                                                  isNotZero);
      end = readDoubleWithinRangeWithValidation(
          "Верхня межа b", &DOUBLE_RANGE, isGreaterAndNotContainZero, start);
      break;
    case YLNX_INDEX:
      start = readDoubleWithinRangeWithValidation("Нижня межа a", &YLNX_RANGE,
                                                  NULL);
      end = readDoubleWithinRangeWithValidation("Верхня межа b", &YLNX_RANGE,
                                                isGreater, start);
      break;
    default:
      handleError("Не існує такої функції!");
      return SUCCESS;
    }

    double y =
        readDoubleWithinRangeWithValidation("Параметр y", &DOUBLE_RANGE, NULL);
    double precision = readDoubleWithinRangeWithValidation(
        "Точність e", &PRECISION_RANGE, NULL);

    EquationFunc func = template(y);
    int decimalPlaces = -(int)log10(precision);
    double x;

    do {
      int methodIndex = getUserChoice(METHOD_PROMPT, METHOD_DESCRIPTIONS,
                                      METHOD_COUNT, &METHOD_CHOICE_RANGE);

      EquationMethod method = METHODS[methodIndex];
      x = method(start, end, func, precision);

      if (x != INFINITY) { // INFINITY means error
        showSuccess("x = %.*lf", decimalPlaces, x);
        printf(EQUATION_TEMPLATES[funcIndex], y, decimalPlaces, x,
               decimalPlaces, x, decimalPlaces, x);
        printf("= %.*lg", decimalPlaces, func(x));
      }
    } while (
        x == INFINITY &&
        askQuestion("Метод не знайшов розвязків' на цьому "
                    "проміжку, чи не бажаєте ви використати інший метод?"));
    Block_release(func);
  } while (askQuestion("Чи бажаєте Ви повторити програму?"));
  return SUCCESS;
}
```

../src/common/constants.h

```c
#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

```c
#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;

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 double (^EquationFunc)(double x);

typedef double (*EquationMethod)(double start, double end, EquationFunc func,
                                 double precision);
typedef EquationFunc (*EquationTemplate)(double);
```

```

EquationFunc yxTemplate(double y);
EquationFunc ylnxTemplate(double y);
double newtonMethod(double start, double end, EquationFunc func,
                    double precision);
double bisectorMethod(double start, double end, EquationFunc func,
                      double precision);

#endif // !ALGORITHM_H
```

../src/algorithm.c

```
#include "algorithm.h"
#include <Block.h>

#include <math.h>

#define RELATIVE_EPSILON 1e-6

#include "io/utils.h"

#define MAX_BISECTOR_ITERATIONS 1000000
#define MAX_NEWTON_ITERATIONS 1000

EquationFunc yxTemplate(double y) {
  // Blocks are awesome for creating lambdas
  double (^f)(double) = ^double(double x) {
    double frac = 1.0 / x;
    return cos(y * frac) - 2.0 * sin(frac) + frac;
  };
  return Block_copy(f);
}

EquationFunc ylnxTemplate(double y) {
  double (^f)(double) = ^double(double x) {
    double ln = log(x);
    return y * ln + sin(ln) - cos(ln);
  };
  return Block_copy(f);
}

double calculateDerivative(double x, EquationFunc func) {
  return (func(x + RELATIVE_EPSILON) - func(x)) / RELATIVE_EPSILON;
}

double bisectorMethod(double start, double end, EquationFunc func,
                      double precision) {
  int iteration = 0;

  if (func(start) * func(end) > 0.0) {
    handleError("Половинний метод не гарантує знаходження кореня, якщо "
                "значення функцій у межах має однаковий знак");
    return INFINITY;
  }
  double x;

  do {
    x = (start + end) / 2.0;

```

```
    double f_mid = func(x);
    if (fabs(f_mid) < precision) {
      return x;
    }

    if (func(start) * f_mid > 0.0) {
      start = x;
    } else {
      end = x;
    }

    iteration++;
    if (iteration > MAX_BISECTOR_ITERATIONS) {
      showWarning("Половинному методу не вдалося зійтися за %d ітерацій.",
                  MAX_BISECTOR_ITERATIONS);
      return INFINITY;
    }
  } while (fabs(end - start) > precision);

  return x;
}

double newtonMethod(double start, double end, EquationFunc func,
                    double precision) {
  int iteration = 0;
  double x = start;
  double delta;
  do {
    double derivative = calculateDerivative(x, func);
    if (fabs(derivative) < 1e-20) { // Avoid division by zero
      showWarning("Похідна функції занадто близька до нуля: %.lg", derivative);
      return INFINITY;
    }

    delta = func(x) / derivative;
    x -= delta;

    iteration++;
    if (iteration > MAX_NEWTON_ITERATIONS) {
      showWarning("Методу Ньютона не вдалося зійтись за %d ітерацій.",
                  MAX_NEWTON_ITERATIONS);
      return INFINITY;
    }

    if (fabs(delta) ⩽ precision) {
      break;
    }

    if (isnan(delta) || isinf(delta)) {
      showWarning("Метод Ньютона зустрівся з проблемою: delta is NaN or Inf.");
      return INFINITY;
    }

  } while (true);

  if (x < start || x > end) {
    showWarning("Рішення вийшло за межі заданого інтервалу.");
  }

  return x;
```

```
}
```

../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);
void printIntCharacterCountRange();
int readIntWithinRangeWithValidation(const char *prompt, const IntRange *range,
                                     IntValidation additionalCheck, ...);

#endif // !INT_H
```

../src/io/int.c

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

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

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

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

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

```
}

// Range Checking Functions
RangeCheckResult validateRangeInt(int value, const IntRange *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 printIntRange(const char *name, const IntRange *range) {
  printf("%d", range->min);
  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

```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);
}

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", MAX_CHAR_COUNT_DOUBLE);
}

```

```
58 void printDoublePrecisionLimit() {
59   printf("Будьякі- дійсні числа приймаються з точністю, не меншою за %g\n",
60          TOLERANCE_DOUBLE);
61 }
62
63 RangeCheckResult validateOverflowDouble(double value) {
64   if (errno == ERANGE) {
65     if (fabs(value) == OVERFLOW_ABS_VALUE_DOUBLE)
66       return GREATER_EQUAL;
67     if (fabs(value) < MIN_ABS_VALUE_DOUBLE)
68       return LESS_EQUAL;
69   }
70   return WITHIN_RANGE;
71 }
72
73 // UTILS
74 double truncateToPrecisionDouble(double num, int decimalPlaces) {
75   double factor = pow(10.0, (double)decimalPlaces);
76   return trunc(num * factor) / factor;
77 }
78
79 void printTruncatedDouble(double num, int decimalPlaces) {
80   printf("%.*lf", decimalPlaces, truncateToPrecisionDouble(num, decimalPlaces))
       ;
81 }
82
83 bool isDoublePrecise(double value) {
84   if (isinf(value) || isnan(value)) {
85     return false;
86   }
87
88   if (value == 0.0) {
89     return true;
90   }
91
92   double scale = pow(10.0, MAX_DIGITS_DOUBLE);
93   double scaledValue = round(value * scale);
94   double roundedValue = scaledValue / scale;
95
96   double tolerance = fabs(value) * TOLERANCE_DOUBLE;
97
98   return fabs(value - roundedValue) < tolerance;
99 }
100
101 // INPUT
102
103 int readDoubleQuiet(double *value) {
104   errno = 0;
105
106   char *input = readInput(INITIAL_CHAR_COUNT_DOUBLE, MAX_CHAR_COUNT_DOUBLE);
107
108   if (input == NULL) {
109     return FAILURE;
110   }
111
112   replaceCommasWithDots(input);
113
114   char *endptr;
115   *value = strtod(input, &endptr);
116
```

```
  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 VALIDATORS_H
#define VALIDATORS_H

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

bool isGreater(double b, va_list args);
bool isNotZero(double b, va_list args);
bool isNotContainZero(double b, va_list args);
bool isGreaterAndNotContainZero(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"

bool isGreater(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 isNotZero(double b, va_list args) {
 (void)args;
 if (b == 0) {
  handleError("Значення не може дорівнювати 0!");
  return false;
 }
 return true;
}

bool isNotContainZero(double b, va_list args) {
 double a = va_arg(args, double);
 if (a * b <= 0) {
  handleError("Проміжок не повинен містити 0!");
  return false;
 }
```

```
  return true;
}

bool isGreaterAndNotContainZero(double b, va_list args) {
  va_list argsCopy;
  va_copy(argsCopy, args);
  bool result = isNotContainZero(b, args) && isGreater(b, argsCopy);
  va_end(argsCopy);
  return result;
}
```

../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

Випадок 1

```
Вас вітає програма для знаходження коренів двох рівнянь:
1. cos(y / x) - 2 * sin(1 / x) + 1 / x
2. y * ln(x) + sin(ln(x)) - cos(ln(x))

за допомогою двох методів:
1. Метод половинного ділення
2. Метод Ньютона
на дійсному проміжку [a, b]

УВАГА! Приймаються тільки проміжки, що не містять точок розриву заданих функцій.

УВАГА! Наші обмеження:
-1e+06 ≤ Нижня межа a, верхня межа b, параметр y ≤ 1e+06
2.22045e-16 ≤ Точність < 1
Будь-які дійсні числа приймаються з точністю, не меншою за 2.22045e-16
0 < кількість символів у дійсних числах ≤ 308

Оберіть функцію для розрахунку:
1. cos(y / x) - 2 * sin(1 / x) + 1 / x
2. y * ln(x) + sin(ln(x)) - cos(ln(x))

Оберіть опцію (від 1 до 2): 1
Нижня межа a (від -1e+06 до 1e+06): 10
Верхня межа b (від -1e+06 до 1e+06): 12
Параметр y (від -1e+06 до 1e+06): 155
Точність e (від 2.22045e-16 до 1): 1e-12

Оберіть метод для розрахунку функції:
1. Метод половинного ділення
2. Метод Ньютона

Оберіть опцію (від 1 до 2): 1
x = 11.034645541553
cos(155 / 11.034645541553) - 2 * sin(1 / 11.034645541553) + 1 / 11.034645541553= 6.49064135771e-13
Чи бажаєте Ви повторити програму?
Введіть +, якщо погоджуєтесь. Інакше введіть будь-яку іншу клавішу:
```

Випадок 2

```
Оберіть функцію для розрахунку:
1. cos(y / x) - 2 * sin(1 / x) + 1 / x
2. y * ln(x) + sin(ln(x)) - cos(ln(x))

Оберіть опцію (від 1 до 2): 2
Нижня межа a (від 0 до 1e+06): 0.5
Верхня межа b (від 0 до 1e+06): 4
Параметр y (від -1e+06 до 1e+06): 30
Точність e (від 2.22045e-16 до 1): 1e-4

Оберіть метод для розрахунку функції:
1. Метод половинного ділення
2. Метод Ньютона

Оберіть опцію (від 1 до 2): 2
x = 1.0328
30 * ln(1.0328) + sin(ln(1.0328)) - cos(ln(1.0328))= -9.874e-08
Чи бажаєте Ви повторити програму?
Введіть +, якщо погоджуєтесь. Інакше введіть будь-яку іншу клавішу: -
```

Випадок 3

```
Оберіть функцію для розрахунку:
1. cos(y / x) - 2 * sin(1 / x) + 1 / x
2. y * ln(x) + sin(ln(x)) - cos(ln(x))

Оберіть опцію (від 1 до 2): 1
Нижня межа a (від -1e+06 до 1e+06): -22
Верхня межа b (від -1e+06 до 1e+06): -18
Параметр y (від -1e+06 до 1e+06): -25
Точність e (від 2.22045e-16 до 1): 1e-12

Оберіть метод для розрахунку функції:
1. Метод половинного ділення
2. Метод Ньютона

Оберіть опцію (від 1 до 2): 1

Метод не знайшов розв'язків на цьому проміжку, чи не бажаєте ви використати інший
метод?
Введіть +, якщо погоджуєтесь. Інакше введіть будь-яку іншу клавішу: +

Оберіть метод для розрахунку функції:
1. Метод половинного ділення
2. Метод Ньютона

Оберіть опцію (від 1 до 2): 2

УВАГА! Рішення вийшло за межі заданого інтервалу.
x = -15.279329899441
cos(-25 / -15.279329899441) - 2 * sin(1 / -15.279329899441) + 1 / -15.279329899441
= 4.16333634234e-17
Чи бажаєте Ви повторити програму?
Введіть +, якщо погоджуєтесь. Інакше введіть будь-яку іншу клавішу: -
```

Випадок 4

```
Оберіть функцію для розрахунку:
1. cos(y / x) - 2 * sin(1 / x) + 1 / x
2. y * ln(x) + sin(ln(x)) - cos(ln(x))

Оберіть опцію (від 1 до 2): 2
Нижня межа a (від 0 до 1e+06): 1.5
Верхня межа b (від 0 до 1e+06): 2.5
Параметр y (від -1e+06 до 1e+06): .1
Точність e (від 2.22045e-16 до 1): 1e-15

Оберіть метод для розрахунку функції:
1. Метод половинного ділення
2. Метод Ньютона

Оберіть опцію (від 1 до 2): 1
x = 2.082372835673411
0.1 * ln(2.082372835673411) + sin(ln(2.082372835673411)) - cos(ln(2.082372835673411))
= 4.44089209850063e-16
Чи бажаєте Ви повторити програму?
Введіть +, якщо погоджуєтесь. Інакше введіть будь-яку іншу клавішу: -
```

Перевірка

Точність	Точний x (20 знаків)	Похибка x	Похибка рівняння з неточним x
$1 \cdot 10^{-12}$	11.034645541552382	$6.181889852449607 \cdot 10^{-13}$	$7.887451345657911 \cdot 10^{-13}$
0.0001	1.032766867509054	0.000033132490946	0.000995520906751
$1 \cdot 10^{-12}$	-15.279329899440546	$4.541144885258572 \cdot 10^{-13}$	$4.658835740471967 \cdot 10^{-14}$
$1 \cdot 10^{-15}$	2.082372835662682	$1.07293906086701 \cdot 10^{-11}$	$5.223076203313137 \cdot 10^{-16}$

Як бачимо, похибка рівняння з неточним x не перевищує заявлену точність. У другому випадку точність невисока, тому метод Ньютона видав недостатньо правильні результати.

Висновки: Програма працює коректно. Програма вирішує поставлене завдання.