**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.  “25” вересня 2024 р. | Виконав  Студент групи ІП-43  Дутов І.А. |
| --- | --- |

**Київ 2024**

# **Комп’ютерний практикум №2**

***Тема:*** Програмування лінійних алгоритмів

**Завдання:**

Написати програму, що за введеними сторонами трикутника обчислює його:

* площу;
* периметр;
* висоти;
* бісектриси;
* медіани.

## Текст програми:

Якщо ви не хочете копіювати код або бажаєте подивитися тестування, то можна просто завантажити його за [посиланням](https://drive.google.com/drive/folders/1FgdpwemNoFk0gS75C1mESocEh-xu2NBb).

### Файлова структура

/lab2\_triangle\_props\_from\_sides

build

include

│ constants.h

│ triangle.h

└ utils.h

src

│ main.c

│ triangle.c

└ utils.c

test

unity

Makefile

README.md

### main.c

| #include "../include/triangle.h"  void process\_arguments(int argc, char \*argv[], int \*repeat\_mode,  int \*quiet\_mode, int \*help\_mode); void print\_demo(); void read\_triangle\_sides(long double sides[SIDE\_COUNT],  const char \*side\_labels[SIDE\_COUNT],  const char \*repeat\_side\_input); void read\_triangle\_properties(long double properties[PROPERTY\_COUNT],  long double a, long double b, long double c); void print\_properties(long double properties[PROPERTY\_COUNT],  const char \*property\_labels[PROPERTY\_COUNT],  long double sides[SIDE\_COUNT], unsigned decimal\_places); void print\_help();  int main(int argc, char \*argv[]) {  int repeat\_mode = 0;  int quiet\_mode = 0;  int help\_mode = 0;   process\_arguments(argc, argv, &repeat\_mode, &quiet\_mode, &help\_mode);   if (help\_mode) {  print\_help();  return 0;  }   long double sides[SIDE\_COUNT] = {0};  long double properties[PROPERTY\_COUNT] = {0};   const char \*property\_labels[PROPERTY\_COUNT] = {  "Периметр трикутника:", "Площа трикутника:",  "Висота до сторони а:", "Висота до сторони b:",  "Висота до сторони c:", "Медіана до сторони a:",  "Медіана до сторони b:", "Медіана до сторони c:",  "Бісектриса до сторони a:", "Бісектриса до сторони b:",  "Бісектриса до сторони c:"};  const char \*side\_labels[SIDE\_COUNT] = {"a", "b", "c"};   const char \*repeat\_side\_input =  "Просимо Вас вводити лише програмно коректні сторони трикутника.\n\n";  const char \*repeat\_decimal\_places\_input =  "Просимо Вас вводити лише програмно коректну точність.\n\n";  unsigned decimal\_places = 0;   do {  if (!quiet\_mode) {  print\_demo();  printf("\n");  }   *// Get sides from user*  do {  read\_triangle\_sides(sides, side\_labels, repeat\_side\_input);  } while (validate\_triangle(sides[0], sides[1], sides[2]) != 0);  printf("\n");   *// Get decimal places from user*  while (read\_decimal\_places(&decimal\_places) != 0) {  printf("%s", repeat\_decimal\_places\_input);  }  printf("\n");   *// Calculating results*  read\_triangle\_properties(properties, sides[0], sides[1], sides[2]);  print\_properties(properties, property\_labels, sides, decimal\_places);  printf("\n");   if (!quiet\_mode) {  if (!repeat\_mode) {  printf("\nPS: ");  print\_help();  printf("Дякуємо за використання нашої програми!\n");  printf("Бажаємо Вам гарного дня!\n");  printf("\nНатисніть Enter для виходу. ");  getchar();  } else {  printf("\nНатисніть будь-яку клавішу для продовження або Ctrl+C для "  "виходу.\n\n");  getchar();  }  }  } while (repeat\_mode);   return 0; }  void process\_arguments(int argc, char \*argv[], int \*repeat\_mode,  int \*quiet\_mode, int \*help\_mode) {  for (int i = 1; i < argc; i++) {  if (strcmp(argv[i], "-q") == 0 || strcmp(argv[i], "--quiet") == 0) {  \*quiet\_mode = 1;  } else if (strcmp(argv[i], "-r") == 0 || strcmp(argv[i], "--repeat") == 0) {  \*repeat\_mode = 1;  } else if (strcmp(argv[i], "-h") == 0 || strcmp(argv[i], "--help") == 0) {  \*help\_mode = 1;  }  } }  void print\_demo() {  printf("Вас вітає Простокутник -- інструмент, що допоможе вам знайти: \n");  printf("\t- периметр трикутника\n");  printf("\t- площу трикутника\n");  printf("\t- висоти трикутника\n");  printf("\t- медіани трикутника\n");  printf("\t- бісектриси трикутника\n");   printf("\nНАШІ ОБМЕЖЕННЯ:\n");  printf("\t- максимальна довжина сторони трикутника: %Lg\n", MAX\_SIDE\_LENGTH);  printf("\t- максимальна довжина сторони в символах: %u\n",  NUMBER\_LENGTH\_LIMIT);  printf("\t- мінімальна довжина сторони трикутника: %Lg\n", MIN\_SIDE\_LENGTH);  printf("\t- максимальна гарантована точність (КІЛЬКІСТЬ знаків після коми) : "  "%u\n",  MAX\_GUARANTEED\_PRECISION);  printf("\t- мінімальна гарантована точність: %u\n", MIN\_GUARANTEED\_PRECISION);  printf("\t- застосовується автоматична точність: вона збільшується до тих "  "пір, \n\t\t доки довжини не можна буде розрізнити як рядки\n");  printf("\t- округлення НЕ застосовується \n");  printf("\nДля того, щоб вийти з програми передчасно, натисніть Ctrl+C.\n"); }  void read\_triangle\_sides(long double sides[SIDE\_COUNT],  const char \*side\_labels[SIDE\_COUNT],  const char \*repeat\_side\_input) {  for (int i = 0; i < SIDE\_COUNT; i++) {  while (read\_triangle\_side(&sides[i], side\_labels[i]) != 0) {  printf("%s", repeat\_side\_input);  }  } }  void read\_triangle\_properties(long double properties[PROPERTY\_COUNT],  long double a, long double b, long double c) {  properties[0] = get\_triangle\_perimeter(a, b, c);  properties[1] = get\_triangle\_area(a, b, c);  properties[2] = get\_triangle\_height(a, b, c);  properties[3] = get\_triangle\_height(b, a, c);  properties[4] = get\_triangle\_height(c, a, b);  properties[5] = get\_triangle\_median(a, b, c);  properties[6] = get\_triangle\_median(b, a, c);  properties[7] = get\_triangle\_median(c, a, b);  properties[8] = get\_triangle\_bisector(a, b, c);  properties[9] = get\_triangle\_bisector(b, a, c);  properties[10] = get\_triangle\_bisector(c, a, b); }  void print\_properties(long double properties[PROPERTY\_COUNT],  const char \*property\_labels[PROPERTY\_COUNT],  long double sides[SIDE\_COUNT], unsigned decimal\_places) {  printf("========================================\n");  printf("РЕЗУЛЬТАТИ ОБЧИСЛЕНЬ:\n");  printf("========================================\n");  for (int i = 0; i < PROPERTY\_COUNT; i++) {  print\_with\_additional\_precision(property\_labels[i], decimal\_places,  properties[i], sides);  } };  void print\_help() {  printf("Програму можна запускати з деякими параметрами!\n"  "-q/--quiet: вимкнути додатковий вивід у програмі\n"  "-r/--repeat: програма повторюватиметься після виконання\n"  "-h/--help: вивести цей текст і припинити виконання програми\n"); } |
| --- |

### triangle.c

| #include "../include/triangle.h" #include "../include/utils.h"  *// Core Triangle Calculations*  long double get\_triangle\_perimeter(long double a, long double b,  long double c) {  return (a + b + c); }  long double get\_triangle\_area(long double a, long double b, long double c) {  long double p = get\_triangle\_perimeter(a, b, c) / 2.0;  return sqrtl(p \* (p - a) \* (p - b) \* (p - c)); }  long double get\_triangle\_height(long double side, long double b,  long double c) {  return get\_triangle\_area(side, b, c) \* 2.0 / side; }  long double get\_triangle\_median(long double side, long double b,  long double c) {  return sqrtl(2 \* b \* b + 2 \* c \* c - side \* side) / 2.0; }  long double get\_triangle\_bisector(long double side, long double b,  long double c) {  long double p = get\_triangle\_perimeter(side, b, c) / 2;  return 2 \* sqrtl(b \* c \* p \* (p - side)) / (b + c); }  *// Triangle validation* int validate\_triangle(long double a, long double b, long double c) {  if (a <= 0) {  printf("\nПОМИЛКА! Довжина сторони a має бути додатною!\n");  return 1;  }  if (b <= 0) {  printf("\nПОМИЛКА! Довжина сторони b має бути додатною!\n");  return 1;  }   if (c <= 0) {  printf("\nПОМИЛКА! Довжина сторони c має бути додатною!\n");  return 1;  }   if (a + b <= c) {  printf("\nПОМИЛКА! Сума довжин сторін a та b має бути більшою за довжину "  "сторони c!\n");  return 1;  }   if (a + c <= b) {  printf("\nПОМИЛКА! Сума довжин сторін a та c має бути більшою за довжину "  "сторони b!\n");  return 1;  }   if (b + c <= a) {  printf("\nПОМИЛКА! Сума довжин сторін b та c має бути більшою за довжину "  "сторони a!\n");  return 1;  }   return 0; }  *// Input and precision handling functions*  int read\_triangle\_side(long double \*side, const char \*side\_name) {  char input[NUMBER\_LENGTH\_LIMIT + 1];  char \*endptr;   printf("Введіть, будь ласка, довжину сторони %s: ", side\_name);   if (!fgets(input, sizeof(input), stdin)) {  perror("\nПОМИЛКА! Не вдалося прочитати ввід!\n");  return 1;  }   if (input[strlen(input) - 1] != '\n') {  printf("\nПОМИЛКА! Довжина сторони %s в символах має бути меншою за %u.\n",  side\_name, NUMBER\_LENGTH\_LIMIT);  flush\_input\_buffer();  return 1;  }   replace\_commas\_with\_dots(input);  \*side = strtold(input, &endptr);   *// Check if conversion failed or extra characters exist*  if (endptr == input || \*endptr != '\n') {  printf("\nПОМИЛКА! Довжина сторони %s має бути числом і не містити "  "додаткових символів!\n",  side\_name);  return 1;  }   *// Check for side length constraints*  if (\*side < 0) {  printf("\nПОМИЛКА! Довжина сторони %s має бути додатною!\n", side\_name);  return 1;  }  if (\*side > MAX\_SIDE\_LENGTH) {  printf("\nПОМИЛКА! Довжина сторони %s має бути менша-рівна %Lg.\n",  side\_name, MAX\_SIDE\_LENGTH);  return 1;  }  if (\*side < MIN\_SIDE\_LENGTH - TOLERANCE) {  printf("\nПОМИЛКА! Довжина сторони %s має бути більша-рівна %Lg.\n",  side\_name, MIN\_SIDE\_LENGTH);  return 1;  }   return 0; }  int read\_decimal\_places(unsigned \*decimal\_places) {  printf("Введіть точність від 1 до %u: ", MAX\_GUARANTEED\_PRECISION);   char input[NUMBER\_LENGTH\_LIMIT + 1];  char \*endptr;  long value;   if (!fgets(input, sizeof(input), stdin)) {  printf("\nПОМИЛКА! Не вдалося прочитати ввід!\n");  return 1;  }   if (input[strlen(input) - 1] != '\n') {  printf(  "\nПОМИЛКА! Максимальна довжина значення точності в символах складає "  "%u.\n",  NUMBER\_LENGTH\_LIMIT);  flush\_input\_buffer();  return 1;  }   value = strtol(input, &endptr, 10);   if (endptr == input || \*endptr != '\n') {  printf("\nПОМИЛКА! Точність має бути числом.\n");  return 1;  }   if (value > MAX\_GUARANTEED\_PRECISION) {  printf("\nПОМИЛКА! Точність має бути менша-рівна %u.\n",  MAX\_GUARANTEED\_PRECISION);  return 1;  }   if (value < MIN\_GUARANTEED\_PRECISION) {  printf("\nПОМИЛКА! Точність має бути більша-рівна %u.\n",  MIN\_GUARANTEED\_PRECISION);  return 1;  }   \*decimal\_places = (unsigned)value;  return 0; }  *// Precision Adjustment Functions*  static int numbers\_are\_equal\_by\_string(const long double value1,  const long double value2,  unsigned precision) {  char buffer1[NUMBER\_LENGTH\_LIMIT], buffer2[NUMBER\_LENGTH\_LIMIT];  snprintf(buffer1, sizeof(buffer1), "%.\*Lf", precision, value1);  snprintf(buffer2, sizeof(buffer2), "%.\*Lf", precision, value2);  return strcmp(buffer1, buffer2) == 0; }  void print\_with\_additional\_precision(const char \*text, unsigned int precision,  long double value,  long double sides[SIDE\_COUNT]) {  char formatted\_value[NUMBER\_LENGTH\_LIMIT];  unsigned needed\_precision = precision;   for (int i = 0; i < SIDE\_COUNT; i++) {  if (!numbers\_are\_equal\_by\_string(value, sides[i], needed\_precision)) {  continue;  }   while (numbers\_are\_equal\_by\_string(value, sides[i], needed\_precision)) {  if (needed\_precision == MAX\_ACTUAL\_PRECISION) {  break;  }  needed\_precision++;  }  }   unsigned final\_precision =  (needed\_precision > precision) ? needed\_precision : precision;   *// Extra digit for rounding prevention*  snprintf(formatted\_value, sizeof(formatted\_value), "%.\*Lf",  final\_precision + 1, value);  truncate\_to\_precision(formatted\_value, final\_precision);   printf("%s %s\n", text, formatted\_value); } |
| --- |

### utils.c

| #include "../include/utils.h" void replace\_commas\_with\_dots(char \*str) {  while (\*str) {  if (\*str == ',') {  \*str = '.';  }  str++;  } }  void flush\_input\_buffer() {  int ch;  while ((ch = getchar()) != '\n' && ch != EOF) {  *// Keep consuming the remaining characters until newline or EOF*  } }  void truncate\_to\_precision(char \*formatted\_value, unsigned final\_precision) {  for (int i = 0; formatted\_value[i] != '\0'; i++) {  if (formatted\_value[i] == '.') {  *// If we reach the decimal point, truncate after the required precision*  if (i + final\_precision + 1 < strlen(formatted\_value)) {  formatted\_value[i + final\_precision + 1] = '\0';  }  break;  }  } } |
| --- |

### constants.h

| *// Side lengths* #define MAX\_SIDE\_LENGTH (long double)1e6 #define MIN\_SIDE\_LENGTH (long double)1e-3  *// Precision settings* #define MAX\_GUARANTEED\_PRECISION 15 *// Maximum precision for guaranteed results* #define MAX\_ACTUAL\_PRECISION 50 *// Maximum possible precision* #define MIN\_GUARANTEED\_PRECISION 1 *// Minimum precision*  *// General tolerance for floating-point comparisons* #define TOLERANCE (long double)1e-15  *// Max length of number (whole part + fraction + decimal point)* #define NUMBER\_LENGTH\_LIMIT (7 + MAX\_GUARANTEED\_PRECISION + 1)  *// Triangle properties* #define SIDE\_COUNT 3 *// A triangle has 3 sides* #define PROPERTY\_COUNT \  (SIDE\_COUNT \* 3 + 2) *// 3 heights, 3 medians, 3 bisectors, perimeter, area*  *// Utility macros* #define MAX(a, b) (((a) > (b)) ? (a) : (b)) |
| --- |

### triangle.h

| #ifndef TRIANGLE\_H #define TRIANGLE\_H  #include "constants.h" #include <math.h> #include <stdio.h> #include <stdlib.h> #include <string.h>  *// --------------- Function Declarations ---------------*  *// Triangle property calculation functions* long double get\_triangle\_perimeter(long double a, long double b, long double c); long double get\_triangle\_area(long double a, long double b, long double c); long double get\_triangle\_height(long double side, long double b, long double c); long double get\_triangle\_median(long double side, long double b, long double c); long double get\_triangle\_bisector(long double side, long double b,  long double c);  *// Input functions* int read\_triangle\_side(long double \*side, const char \*side\_name); int read\_decimal\_places(unsigned \*decimal\_places);  *// Precision handling and printing functions* void print\_with\_additional\_precision(const char \*text, unsigned int precision,  long double value,  long double sides[SIDE\_COUNT]);  *// Triangle validation function* int validate\_triangle(long double a, long double b, long double c);  #endif *// TRIANGLE\_H* |
| --- |

### utils.h

| #ifndef UTILS\_H #define UTILS\_H  #include "constants.h" #include <stdio.h> #include <string.h>  void flush\_input\_buffer(); void replace\_commas\_with\_dots(char \*str); void truncate\_to\_precision(char \*formatted\_value, unsigned final\_precision);  #endif *// UTILS\_H* |
| --- |

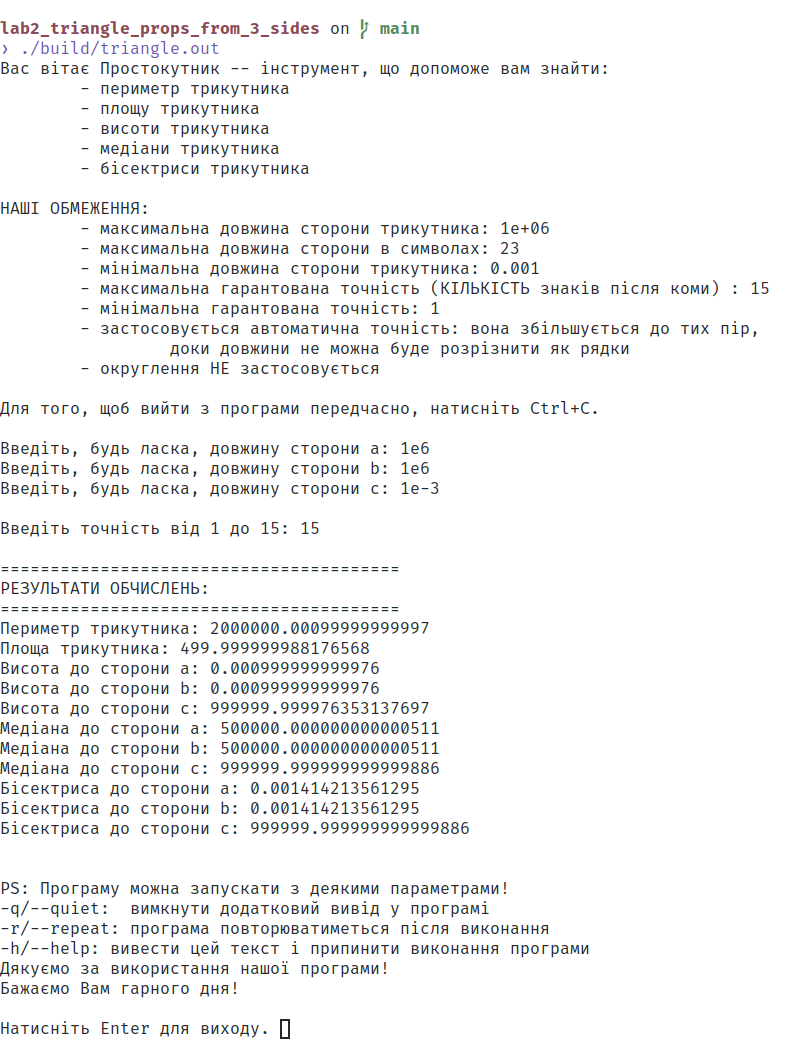
### Makefile

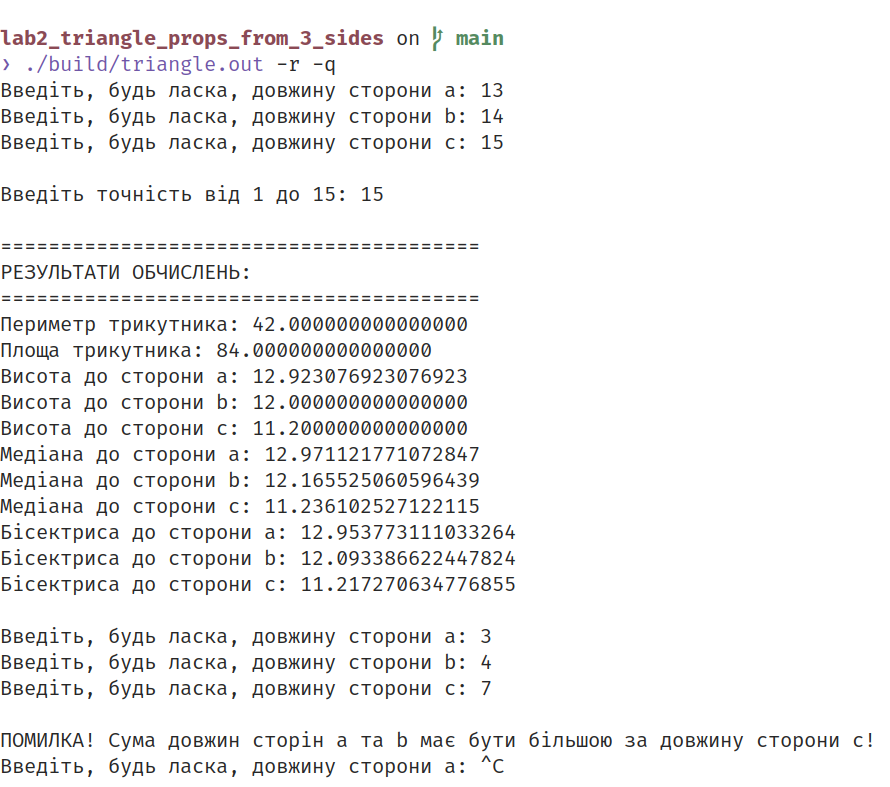
| *# Compiler* CC = clang CFLAGS = -I. -I$(PATHU) -I$(PATHS) -g -O2 -ffast-math -march=native LDFLAGS = -lm  *# Directories* PATHU = unity/src/ PATHS = src/ PATHT = test/ PATHB = build/ PATHD = build/depends/ PATHO = build/objs/ PATHR = build/results/  *# Create all necessary directories* BUILD\_PATHS = $(PATHB) $(PATHD) $(PATHO) $(PATHR)  EXECS = triangle EXECT = triangle\_test  *# Source files* MAIN\_SRCS = $(wildcard $(PATHS)\*.c) *# Main source files* TEST\_SRCS = $(wildcard $(PATHT)\*.c) *# Test source files* UNITY\_SRC = $(PATHU)unity.c *# Unity source file*  *# Object files* MAIN\_OBJS = $(MAIN\_SRCS:$(PATHS)%.c=$(PATHO)%.o) TEST\_OBJS = $(TEST\_SRCS:$(PATHT)%.c=$(PATHO)%.o) UNITY\_OBJ = $(PATHO)unity.o  TARGET\_EXTENSION = out  .PHONY: all clean test  all: $(BUILD\_PATHS) $(PATHB)$(EXECS).$(TARGET\_EXTENSION)  *# Build main executable* $(PATHB)$(EXECS).$(TARGET\_EXTENSION): $(MAIN\_OBJS)  @echo "Linking main executable..."  $(CC) -o $@ $^ $(LDFLAGS)  *# Build test executable* $(PATHB)$(EXECT).$(TARGET\_EXTENSION): $(TEST\_OBJS) $(UNITY\_OBJ)  @echo "Linking test executable..."  $(CC) -o $@ $^ $(LDFLAGS)  *# Compile main source files* $(PATHO)%.o: $(PATHS)%.c  @echo "Compiling $<..."  $(CC) $(CFLAGS) -c $< -o $@  *# Compile test source files* $(PATHO)%.o: $(PATHT)%.c  @echo "Compiling $<..."  $(CC) $(CFLAGS) -DTEST -c $< -o $@ *# Define TEST for test compilation*  *# Compile Unity* $(UNITY\_OBJ): $(UNITY\_SRC)  @echo "Compiling Unity..."  $(CC) $(CFLAGS) -c $< -o $@  *# Create directories if they don't exist* $(BUILD\_PATHS):  @mkdir -p $@  *# Clean build artifacts* clean:  @echo "Cleaning up..."  rm -f $(PATHO)\*.o  rm -f $(PATHB)\*.$(TARGET\_EXTENSION)  rm -f $(PATHR)\*.txt  *# Run tests* test: $(PATHB)$(EXECT).$(TARGET\_EXTENSION)  @echo "Running tests..."  ./$(PATHB)$(EXECT).$(TARGET\_EXTENSION) |
| --- |

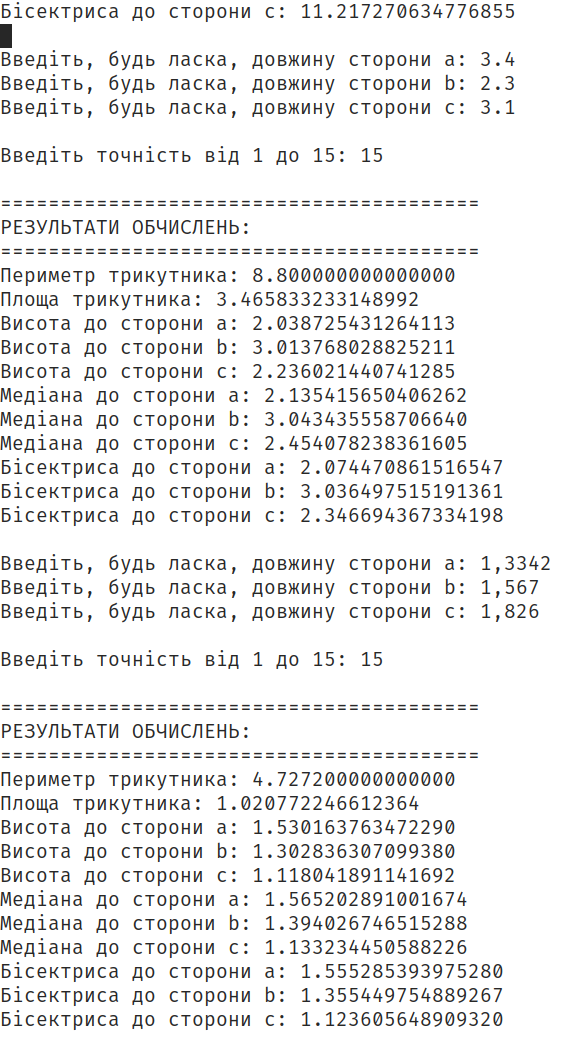
## Введені та одержані результати:

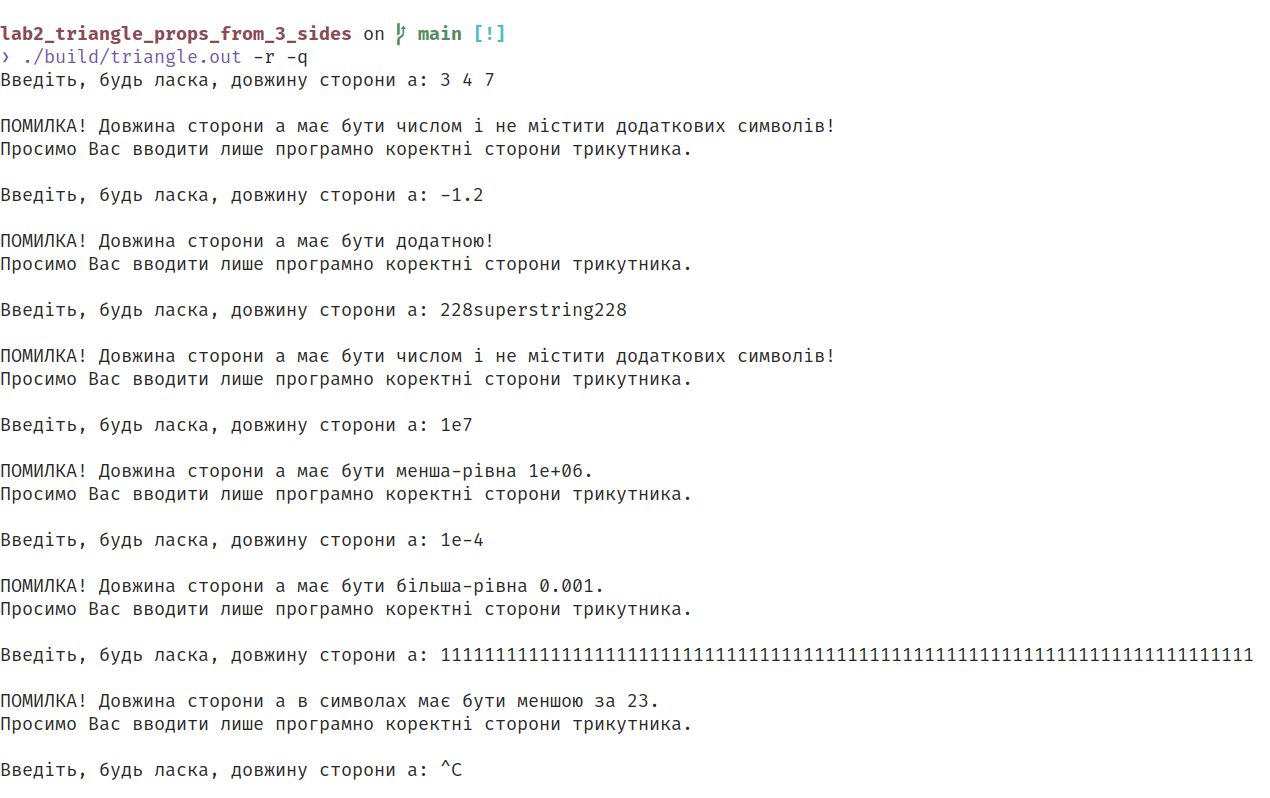
### Сторони:

1. Showcase
   1. 1e6 1e6 1e-3
2. Цілі + помилка
   1. 13 14 15
   2. 3 4 7
3. Дробові
   1. 3.4 2.3 3.1
   2. 1.3342 1.567 1.826
4. Error cases
   1. 3 4 7 – multiple symbols
   2. -1.2
   3. 228superstring228
   4. 1e7
   5. 1e-4



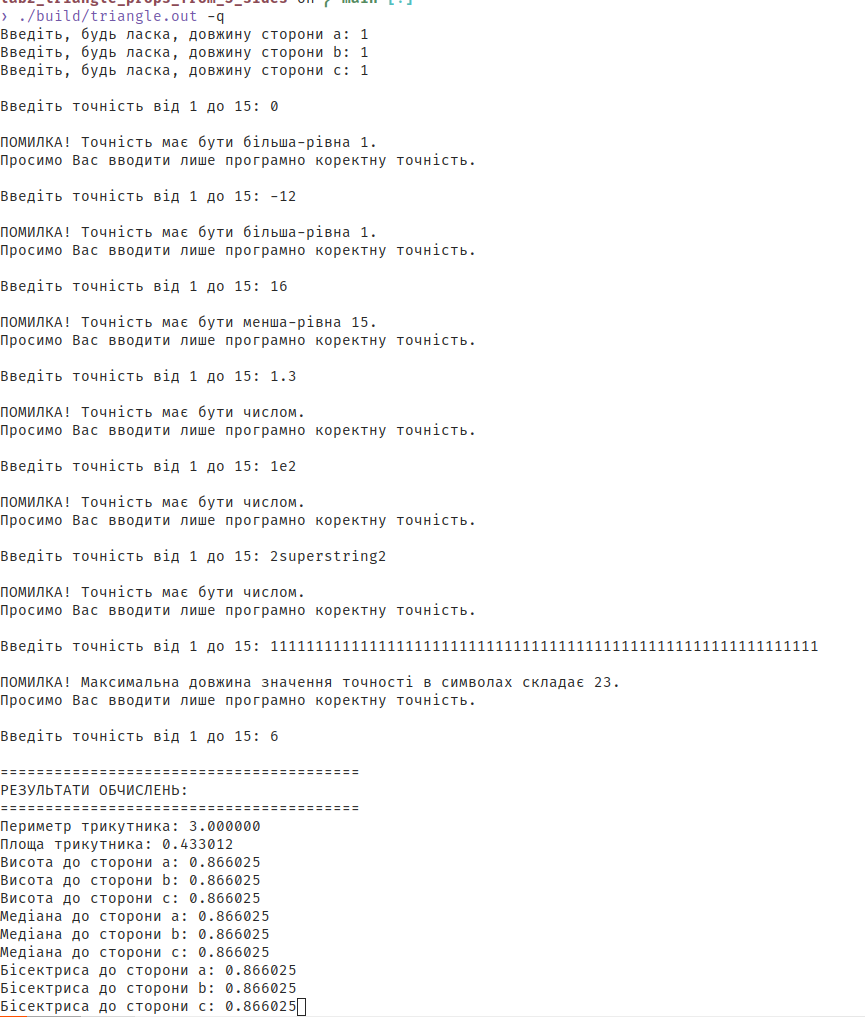






### Десяткова частина:

1. 0
2. -12
3. 16
4. 1.3
5. 1e2
6. 2superstring2
7. 6



## Теоретичні розрахунки:

### Випадок 1: 1000000 1000000 0.001

|  | **Неточне (з округленням)** | **Точне (до 100 знаків)** | **Різниця** |
| --- | --- | --- | --- |
|  | 2000000.000999999999976 | 2000000.0010000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 | 0.0000000000689178705215454101562500000000000000000000000000000000000000000000000000000000000000000000 |
|  | 499.999999988176569 | 499.9999999999999999374999999999999999960937499999999999995117187499999999999237060546874999999866485596 | 0.0000000118234310605480670928955078085937499999999999995117187499999999999237060546874999999866485596 |
|  | 0.000999999999976 | 0.0009999999999999999998749999999999999999921874999999999999990234374999999999998474121093749999999733 | 0.0000000000000236468621210961341857910156171874999999999999990234374999999999998474121093749999999733 |
|  | 0.000999999999976 | 0.0009999999999999999998749999999999999999921874999999999999990234374999999999998474121093749999999733 | 0.0000000000000236468621210961341857910156171874999999999999990234374999999999998474121093749999999733 |
|  | 999999.999976353137640 | 999999.9999999999998749999999999999999921874999999999999990234374999999999998474121093749999999732971191406 | 0.0000236468621210961341857910156171874999999999999990234374999999999998474121093749999999732971191406 |
|  | 500000.000000000000512 | 500000.0000000000004999999999999999997500000000000000002499999999999999996875000000000000004375000000000000 | 0.0000000000004999999999999999997500000000000000002499999999999999996875000000000000004375000000000000 |
|  | 500000.000000000000512 | 500000.0000000000004999999999999999997500000000000000002499999999999999996875000000000000004375000000000000 | 0.0000000000004999999999999999997500000000000000002499999999999999996875000000000000004375000000000000 |
|  | 999999.999999999999886 | 999999.9999999999998749999999999999999921874999999999999990234374999999999998474121093749999999732971191406 | -0.0000000000001250000000000000000078125000000000000009765625000000000001525878906250000000267028808594 |
|  | 0.001414213561296 | 0.0014142135613124348780383334345586761482409546093868175561927479131279952207816813458904794991502151 | 0.0000000000000167208751971605496308755245409319119380870874427479131279952207816813458904794991502151 |
|  | 0.001414213561296 | 0.0014142135613124348780383334345586761482409546093868175561927479131279952207816813458904794991502151 | 0.0000000000000167208751971605496308755245409319119380870874427479131279952207816813458904794991502151 |
|  | 999999.999999999999886 | 999999.9999999999998749999999999999999921874999999999999990234374999999999998474121093749999999732971191406 | -0.0000000000001250000000000000000078125000000000000009765625000000000001525878906250000000267028808594 |

### Випадок 2: 13 14 15

|  | **Неточне (з округленням)** | **Точне (до 100 знаків)** | **Різниця** |
| --- | --- | --- | --- |
|  | 42.000000000000000 | 42.0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 | 0.0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 |
|  | 84.000000000000000 | 84.0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 | 0.0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 |
|  | 12.923076923076923 | 12.9230769230769230769230769230769230769230769230769230769230769230769230769230769230769230769230769231 | -0.0000000000000002732856676000385330273554875300480769230769230769230769230769230769230769230769230769 |
|  | 12.000000000000000 | 12.0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 | 0.0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 |
|  | 11.200000000000000 | 11.2000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 | 0.0000000000000007105427357601001858711242675781250000000000000000000000000000000000000000000000000000 |
|  | 12.971121771072847 | 12.9711217710728473447539379076715901149025606700983137402131659383197465319906703220640902558603805564 | 0.0000000000000007601379927319158211005440401622858137402131659383197465319906703220640902558603805564 |
|  | 12.165525060596439 | 12.1655250605964393779993684904041341241699401895728223728383060929726654506378204796061328559156973269 | 0.0000000000000007001370164390976181920868103067603223728383060929726654506378204796061328559156973269 |
|  | 11.236102527122116 | 11.2361025271221159322990702227453397149809830800756281967484568371603783226455597990093353332561824893 | 0.0000000000000003859227349627527104253539078847631281967484568371603783226455597990093353332561824893 |
|  | 12.953773111033264 | 12.9537731110332644481634888395467037087594442901633214371555426630678453623155594172972972623322559378 | 0.0000000000000001056681030493029588586007528839133214371555426630678453623155594172972972623322559378 |
| bb | 12.093386622447824 | 12.0933866224478244785499198454556566967015944584128600819048885838958124265385323044402314000758459655 | 0.0000000000000007812926567251918417705541335209128600819048885838958124265385323044402314000758459655 |
| bc | 11.217270634776856 | 11.2172706347768555785931328321304318327818115630740993284325206193734082835782547184143704773908600105 | -0.0000000000000004633877308331001403908815185150509006715674793806265917164217452815856295226091399895 |

### Випадок 3: 3.4 2.3 3.1

|  | **Неточне (з округленням)** | **Точне (до 100 знаків)** | **Різниця** |
| --- | --- | --- | --- |
|  | 8.800000000000000 | 8.8000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 | -0.0000000000000007105427357601001858711242675781250000000000000000000000000000000000000000000000000000 |
|  | 3.465833233148993 | 3.4658332331489927012564082558920281842836939983865203989409738399584716218130910775484730673342403719 | 0.0000000000000002030271177516506659701952540569802703989409738399584716218130910775484730673342403719 |
|  | 2.038725431264113 | 2.0387254312641133536802401505247224613433494108156002346711610823285127187135829867932194513730825717 | 0.0000000000000001455506110215628985806708945768312252346711610823285127187135829867932194513730825717 |
|  | 3.013768028825211 | 3.0137680288252110445707897877321984211162556507708873034269337738769318450548618065638896237689046712 | 0.0000000000000001572370932688239436188892353870990123034269337738769318450548618065638896237689046712 |
|  | 2.236021440741286 | 2.2360214407412856137138117779948568930862541925074325154457895741667558850407039209990148821511228206 | -0.0000000000000001125475552392919727573158148992894424845542104258332441149592960790009851178488771794 |
|  | 2.135415650406262 | 2.1354156504062622421623047931191421252564575538921160227956212085698017967302816034081296871915567489 | -0.0000000000000000024248918261307381886168334617328839772043787914301982032697183965918703128084432511 |
|  | 3.043435558706640 | 3.0434355587066403335195342156548795649609690833884317196689996879461191254061367858994439471755496756 | 0.0000000000000001574785424968904626915059214759665567196689996879461191254061367858994439471755496756 |
|  | 2.454078238361605 | 2.4540782383616052920374011345539151143468152187332368423007883280511212837422820688560902544306447502 | 0.0000000000000001899238293598737476777470837734207368423007883280511212837422820688560902544306447502 |
|  | 2.074470861516548 | 2.0744708615165475813815645364723247011298024145321276288156265366817555864513311349594402515224291641 | -0.0000000000000001114339285282273698131447947534366223711843734633182444135486688650405597484775708359 |
|  | 3.036497515191362 | 3.0364975151913616557711617850611503845426647017232278734679618631527694416407474249626434851763081098 | -0.0000000000000001399011028489883471602228077103861471265320381368472305583592525750373565148236918902 |
|  | 2.346694367334199 | 2.3466943673341986325602191977417998700945320323664336444174997905149494115912411585636108599644889893 | 0.0000000000000001825788330501859047203867378429133086444174997905149494115912411585636108599644889893 |

### Випадок 4: 1.3342 1.567 1.826

|  | **Неточне (з округленням)** | **Точне (до 100 знаків)** | **Різниця** |
| --- | --- | --- | --- |
|  | 4.727200000000000 | 4.7272000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000 | 0.0000000000000001534772309241816401481628417968750000000000000000000000000000000000000000000000000000 |
|  | 1.020772246612365 | 1.0207722466123647505718966286711809891766949004757023779140086087247806034017598802302502208613775988 | -0.0000000000000000136458763435167776685648517303836726220859913912752193965982401197697497791386224012 |
|  | 1.530163763472290 | 1.5301637634722901372686203397859106418478412538985195291770478319963732624820265031183484048289275953 | -0.0000000000000000797694872450267677548539714902421054708229521680036267375179734968816515951710724047 |
|  | 1.302836307099381 | 1.3028363070993806644185023977934664826760624128598626393286644655070588428867388388388643533648724937 | -0.0000000000000000327202106055755817551245357316713873606713355344929411571132611611611356466351275063 |
|  | 1.118041891141692 | 1.1180418911416919502430412143167371184848794090642961422935472165660247572856077549071743930573686734 | -0.0000000000000001066338361448560578494049063575372663577064527834339752427143922450928256069426313266 |
|  | 1.565202891001675 | 1.5652028910016745813051265300076207566650234412124130068090891759429853400029582711607652116733468069 | -0.0000000000000000641607421770983622920124210167953994931909108240570146599970417288392347883266531931 |
|  | 1.394026746515288 | 1.3940267465152883500466489314783212194324603989087487640143403317728787191753712077044896433767823419 | 0.0000000000000001102550188554084707448364368393384362640143403317728787191753712077044896433767823419 |
|  | 1.133234450588227 | 1.1332344505882266902731334018526859936727600566049690621729724804829833537535620949644792050942558156 | 0.0000000000000001074497521519433968124403076640268440621729724804829833537535620949644792050942558156 |
|  | 1.555285393975281 | 1.5552853939752805495757971148409066366992453542593960618515279805939418069745037195695120882360725890 | -0.0000000000000001004493864893161574733414193185921664381484720194060581930254962804304879117639274110 |
|  | 1.355449754889268 | 1.3554497548892677549790268531925260189821278130699973119310031511871241264865390429522356485475610651 | 0.0000000000000000888209787663928448323061024224449973119310031511871241264865390429522356485475610651 |
|  | 1.123605648909320 | 1.1236056489093201134569246386981491059289499305048939501446747904295518674540998452480219172546720091 | 0.0000000000000000928782630689547319025025888465205189501446747904295518674540998452480219172546720091 |

Як бачимо, значення відрізняються не більш ніж на (крім першого випадку, де похибка відбувається через ділення чисел, що сильно відрізняються порядком), що, очевидно, забезпечується типом .

## Висновки:

Теоретичні розрахунки відповідають отримані з точністю не менше за 15 знаків після коми. Програма працює коректно, а користувач отримує теплий та приємний UX, що враховує знакові уподобання (кома чи крапка) та автоматично збільшує кількість знаків після коми в разі їх співпадіння з сторонами трикутника.