

**МIНIСТЕРСТВО  ОСВIТИ І НАУКИ  УКРАЇНИ**

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

“**КИЇВСЬКИЙ  ПОЛІТЕХНІЧНИЙ  ІНСТИТУТ**

**ІМЕНІ ІГОРЯ СІКОРСЬКОГО”**

**Факультет прикладної математики**

**Кафедра програмного забезпечення комп’ютерних систем**

**Лабораторна робота № 7**

**з дисципліни “ Основи програмування ”**

**тема “ Користувацькі функції ”**

|  |  |  |
| --- | --- | --- |
| **Виконала**  **студентка I курсу**  **групи КП-83**  **Снітко Маріанна Дмитрівна**  **(прізвище, ім’я, по батькові)**  **варіант № 14** |  | **Перевірив**  “**\_\_\_\_” “\_\_\_\_\_\_\_\_\_\_\_\_” 20\_\_\_ р.**  **викладач**  **Гадиняк Руслан Анатолійович**  **(прізвище, ім’я, по батькові)** |

**Київ 2018**

**Мета роботи**

Навчитися створювати користувацькі функції та тестувати їх із різними аргументами.

**Постановка завдання**

Скопіювати у проект даного завдання рішення із попередньої лабораторної роботи та продовжити роботу над ним.

Розбити код програми на підпрограми (функції). За можливості виділити чисті функції.

Створити обов'язкові функції:

1. Для векторів (див. прототипи у Додатках):
   * length - знаходження довжини вектору
   * negative - знаходження інвертованого вектора (повертати копію)
   * add - знаходження вектора-суми двох інших векторів (повертати копію)
   * norm - нормалізація вектора (повертати копію)
   * mult - знаходження вектора-множнення вектора на число (повертати копію, див. приклад у Додатках)
   * rotate - поворот вектора на кут (в радіанах, повертати копію)
   * distance - знаходження відстані між двома точками, що описані векторами
   * angle - знаходження кута повороту вектора
   * fromPolar - створення вектора за кутом і довжиною (повертати копію)
   * equals - порівнняння двох векторів
2. Для об'єктів за варіантом (замість {ObjectName} підставити назву вашого об'єкта, див. приклад у Додатках):
   * create{ObjectName} - Функція створення (і ініціалізації) одного об'єкта (копіюванням, або за допомогою вказівника)
   * update{ObjectName}- Функція оновлення одного об'єкта (копіюванням, або за допомогою вказівника)
   * draw{ObjectName}- Функція малювання одного об'єкта
3. Функція mainTest із тестами всіх чистих функцій
4. Різне:
   * Функція для генерації випадкового цілого числа в заданих межах
   * Функція для генерації випадкового дробового числа в заданих межах

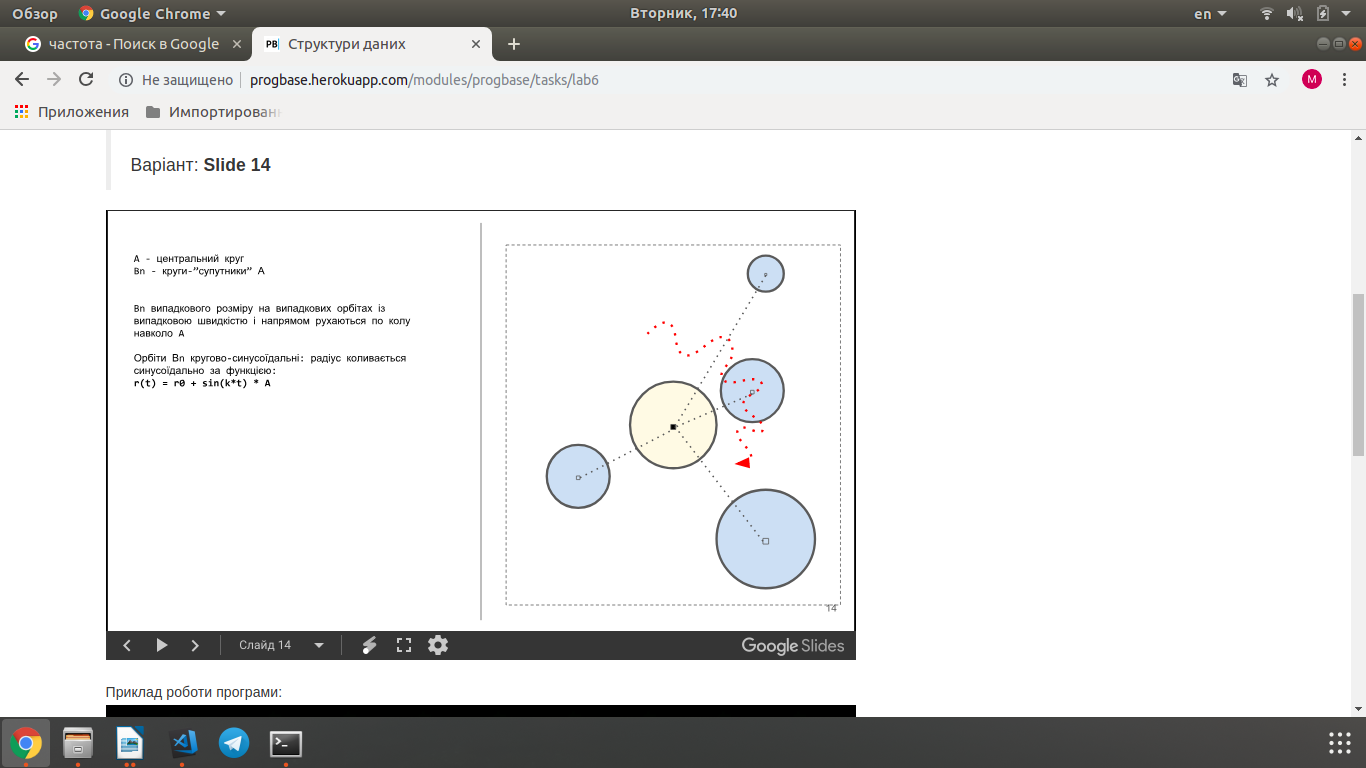
#### Режими роботи програми

На початку роботи програми додати перевірку аргументів командного рядка і реалізувати за допомогою них такі опції програми:

* -t - тестовий режим роботи
* -n VALUE - явно задати значення N

Режими роботи:

1. Нормальний - програма працює аналогічно із попередньою лабораторною роботою.
2. Тестовий - в main() запустити функцію mainTest(), що виконає тестування всіх чистих функцій за допомогою assert().

****

**Тексти коду програм**

|  |
| --- |
| **main.c** |
| #include <stdlib.h>  #include <stdio.h>  #include <math.h>  #include <progbase.h>  #include <progbase/console.h>  #include <progbase/canvas.h>  #include <time.h>  #include <assert.h>  #include <string.h>  struct Vec2D  {  float x;  float y;  };  struct Color  {  int red;  int green;  int blue;  };  struct ball  {  struct Vec2D loc;  float radius;  float sinRadius;  struct Color fillColor;  float radAmplitude;  float radFrequency;  float rotRadius;  float rotAlpha;  float rotSpeed;  };  float length(struct Vec2D v);  struct Vec2D negative(struct Vec2D v);  struct Vec2D add(struct Vec2D a, struct Vec2D b);  struct Vec2D mult(struct Vec2D v, float n);  struct Vec2D norm(struct Vec2D v);  struct Vec2D rotate(struct Vec2D v, float angle);  float distance(struct Vec2D a, struct Vec2D b);  float angle(struct Vec2D v);  struct Vec2D fromPolar(float angle, float length);  int equals(struct Vec2D a, struct Vec2D b);  int fequals(float a, float b);  int randInt(int max, int min);  float randFloat(float max, float min);  void mainTest(void);  //  struct ball createBall(int canvasWidth, int canvasHeight);  struct ball updateBall(struct ball b, float dt, float t, int cWidth, int cHeight);  void drawBall(struct ball ball1);  int main(int argc, char \*argv[])  {  if (argc < 2)  {  return 1;  }  else if (strcmp(argv[1], "-n") == 0)  {  float pi = 3.141592;  srand(time(0));  Console\_clear();  struct ConsoleSize consoleSize = Console\_size();  int width = consoleSize.columns;  int height = consoleSize.rows \* 2;  Canvas\_setSize(width, height);  Canvas\_invertYOrientation();  int nBalls = atoi(argv[2]);  struct ball balls[nBalls];  struct Vec2D center = {width / 2, height / 2};  const int delay = 30;  float dt = delay / 1000.0;  float t = 0;  for (int i = 0; i < nBalls; i++)  {  balls[i] = createBall(width, height);  }  while (!Console\_isKeyDown())  {  t += dt;  for (int i = 0; i < nBalls; i++)  {  balls[i] = updateBall(balls[i], dt, t, width, height);  }  //  Canvas\_beginDraw();  Canvas\_setColorRGB(255, 0, 125);  Canvas\_fillCircle((width / 2), (height / 2), 4);  Canvas\_setColorRGB(125, 5, 125);  Canvas\_putPixel((width / 2), (height / 2));  for (int i = 0; i < nBalls; i++)  {  drawBall(balls[i]);  }  Canvas\_endDraw();  sleepMillis(delay);  }  return 0;  }  else if (strcmp(argv[1], "-t") == 0)  {  mainTest();  }  }  float length(struct Vec2D v)  {  return sqrt(v.x \* v.x + v.y \* v.y);  }  struct Vec2D negative(struct Vec2D v)  {  struct Vec2D neg = v;  neg.x \*= -1;  neg.y \*= -1;  return neg;  }  struct Vec2D add(struct Vec2D a, struct Vec2D b)  {  struct Vec2D sum;  sum.x = a.x + b.x;  sum.y = a.y + b.y;  return sum;  }  struct Vec2D norm(struct Vec2D v)  {  struct Vec2D normVec = v;  float locLength = length(v);  float invLen = (1 / locLength);  normVec.x \*= invLen;  normVec.y \*= invLen;  return normVec;  }  struct Vec2D mult(struct Vec2D v, float n)  {  struct Vec2D newVec = v;  newVec.x \*= n;  newVec.y \*= n;  return newVec;  }  struct Vec2D rotate(struct Vec2D v, float angle)  {  angle += acos(v.x / length(v));  float len = length(v);  struct Vec2D rotated = fromPolar(angle, len);  return rotated;  }  float angle(struct Vec2D v)  {  float angle = acos(v.x / length(v));  return angle;  }  float distance(struct Vec2D a, struct Vec2D b)  {  float dist = sqrt(pow((b.x - a.x), 2) + pow((b.y - a.y), 2));  return dist;  }  int fequals(float a, float b)  {  return fabs(a - b) < 1e-4;  }  int equals(struct Vec2D a, struct Vec2D b)  {  return fequals(a.x, b.x) && fequals(a.y, b.y);  }  int randInt(int max, int min)  {  int r = rand() % (max - min + 1) + min;  return r;  }  float randFloat(float max, float min)  {  float r = (rand() / (float)RAND\_MAX) \* 2 \* max + (min);  return r;  }  struct Vec2D fromPolar(float angle, float length)  {  struct Vec2D frompolar;  frompolar.x = length \* cos(angle);  frompolar.y = length \* sin(angle);  return frompolar;  }  struct ball createBall(int canvasWidth, int canvasHeight)  {  float pi = 3.14159;  struct ball ball = {  .radius = randInt(7, 2),  .fillColor.red = randInt(255, 0),  .fillColor.green = randInt(255, 0),  .fillColor.blue = randInt(255, 0),  .rotRadius = randInt(36, 15),  .rotAlpha = randFloat(pi, -pi),  .rotSpeed = randFloat(pi, -pi),  .radFrequency = randInt(25, 17),  .radAmplitude = randInt(5, 2),  };  return ball;  }  struct ball updateBall(struct ball b, float dt, float t, int cWidth, int cHeight)  {  struct Vec2D center = {cWidth / 2, cHeight / 2};  b.loc.x = center.x + fromPolar(b.rotAlpha, b.sinRadius).x;  b.loc.y = center.y + fromPolar(b.rotAlpha, b.sinRadius).y;  b.rotAlpha += b.rotSpeed \* dt;  b.sinRadius = b.rotRadius + sin(b.radFrequency \* t) \* b.radAmplitude;  return b;  }  void drawBall(struct ball ball1)  {  Canvas\_setColorRGB(ball1.fillColor.red, ball1.fillColor.green, ball1.fillColor.blue);  Canvas\_fillCircle(ball1.loc.x, ball1.loc.y, ball1.radius);  }  void mainTest(void)  {  struct Vec2D a = {1, 0};  struct Vec2D b = {2, -2};  struct Vec2D c = {3, 4};  assert(equals(a, a));  assert(fequals(length(a), 1));  struct Vec2D expected = {-1, 0};  assert(equals(negative(a), expected));  struct Vec2D ex = {3, -2};  assert(equals(add(a, b), ex));  // norm  struct Vec2D s = {0.6, 0.8};  assert(equals(norm(c), s));  // mult  struct Vec2D k = {2, 0};  assert(equals(mult(a, 2), k));  // distance  float m = sqrt(5);  assert(fequals(distance(a, b), m));  //angle  assert(fequals(angle(a), acos(1)));  // fromPolar  struct Vec2D polar = {length(a) \* cos(90), length(a) \* sin(90)};  assert(equals(fromPolar(90, length(a)), polar));  // rotate  struct Vec2D rot = {1 \* cos(90 + acos(1)), 1 \* sin(90 + acos(1))};  assert(equals(rotate(a, 90), rot));  } |

**Приклади результатів**

****

**Висновок**

Протягом виконання цієї лабораторної роботи я ознайомилася із програмуванням на мові С у спеціалізованому редакторі коду.

Більше того, під час виконання лабораторної роботи я навчилася створювати користувацькі функції та тестувати їх із різними аргументами.

Компіляція всього коду відбувалася за допомогою компілятора gcc.

Отже, мета лабораторної роботи досягнена, всі завдання виконані і їх розвʼязання наведені.