rr

**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 “СТРУКТУРИ ДАНИХ”**

|  |  |  |
| --- | --- | --- |
| **Виконав**  **студент I курсу**  **групи КП-83**  **Мортіков Владислав Євгенович**  **варіант №8** |  | **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.  
***(UPD) Обов'язково використати тип Вектор (наприклад, для представлення позиції точки на площині):***

*struct Vec2D  
{  
 float x;  
 float y;  
};*

Створити обов'язкові функції:  
 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  
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.

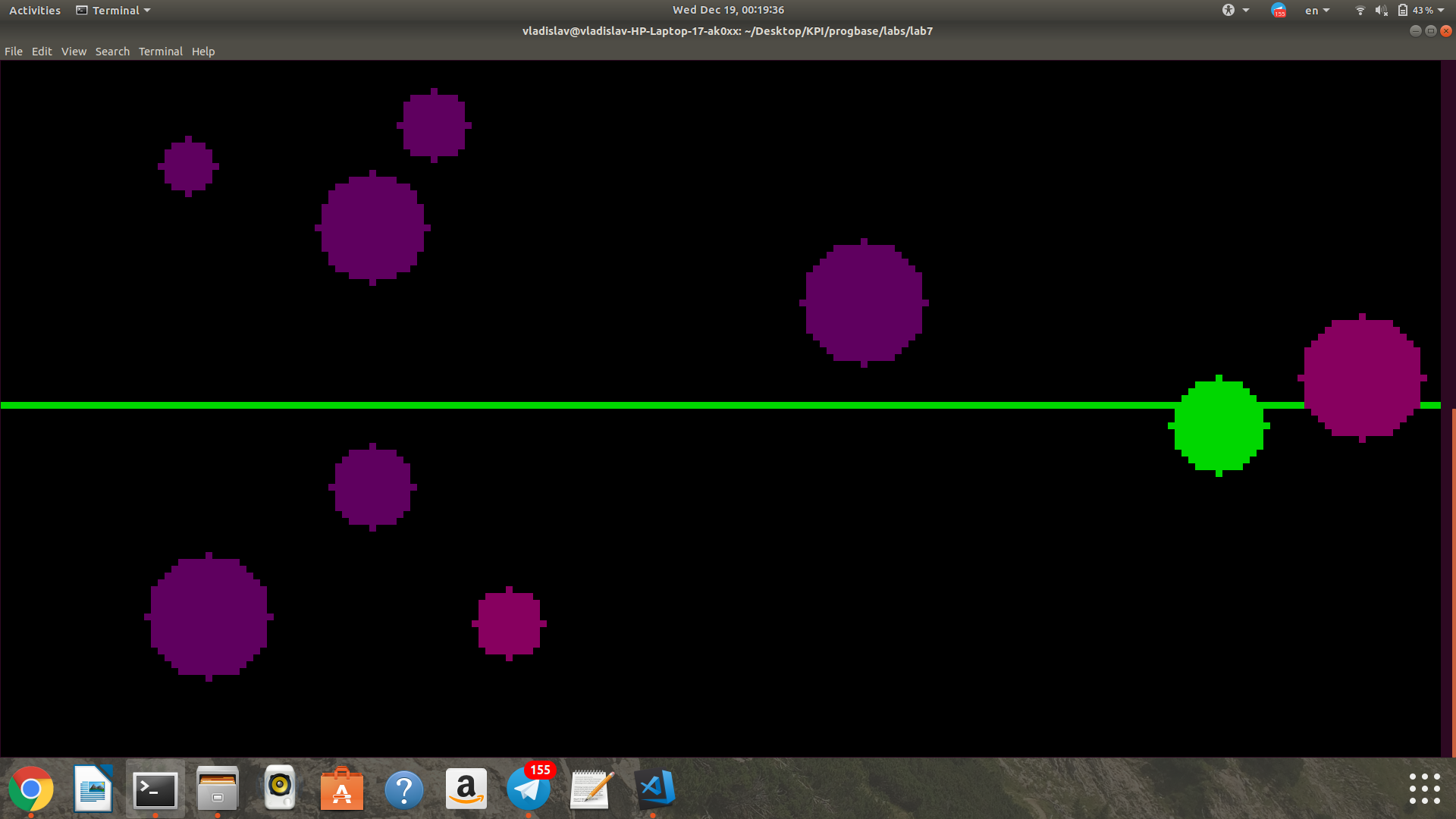
Функція mainTest() із тестами всіх чистих функцій  
***Різне:*** Функція для генерації випадкового цілого числа в заданих межах  
 Функція для генерації випадкового дробового числа в заданих межах  
 ***Режими роботи програми*** На початку роботи програми додати перевірку аргументів командного рядка і реалізувати за допомогою них такі опції програми:  
**-t** - тестовий режим роботи  
**-n VALUE** - явно задати значення N  
Режими роботи:

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

**Тексти коду програм**

|  |
| --- |
| **main.c** |
| #include <stdio.h>  #include <math.h>  #include <stdlib.h>  #include <time.h>  #include <stdbool.h>  #include <string.h>  #include <ctype.h>  #include <assert.h>  #include <progbase.h>  #include <progbase/console.h>  #include <progbase/canvas.h>  struct vec2D  {  float x;  float y;  };  struct colour  {  char r;  char g;  char b;  };  struct ball  {  struct colour ofball;  struct vec2D loc;  int rad;  struct vec2D vel;  float k;  };  int fequals(float a, float b);  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);  struct vec2D mult(struct vec2D v, float n);  int getRand(int min, int max);  struct ball createBall(int canvasWidth, int canvasHeight);  struct ball updateBall(struct ball shar, int x1, long dt, long t, int cWidth, int cHeight);  void drowball(struct ball shar);  void mainTest();  int main(int argc, char \*argv[])  {  if (argc < 2)  {  return 1;  }  else if (strcmp(argv[1], "-n") == 0)  {  srand(time(NULL));  struct ConsoleSize cSize = Console\_size();  const int h = cSize.rows \* 2;  const int w = cSize.columns;  Console\_clear();  Canvas\_setSize(w, h);  Canvas\_invertYOrientation();  int nBalls = atoi(argv[2]);  struct ball shar[nBalls];  for (int i = 0; i < nBalls; i++)  {  shar[i] = createBall(w, h);  }  const long sleepMillisTime = 30;  long t = 0;  int x1 = 0;  do  {  Canvas\_beginDraw();  x1++;  Canvas\_setColorRGB(0, 250, 0);  Canvas\_strokeLine(0, (h / 2), w, (h / 2));  const long dt = sleepMillisTime;  t += dt;  for (int i = 0; i < nBalls; i++)  {  drowball(shar[i]);  shar[i] = updateBall(shar[i], x1, dt, t, w, h);  };  sleepMillis(sleepMillisTime);  Canvas\_endDraw();  } while (!Console\_isKeyDown());  return 0;  }  else if (strcmp(argv[1], "-t") == 0)  {  mainTest();  }  }  struct ball createBall(int canvasWidth, int canvasHeight)  {  struct ball shar =  {  .loc.x = getRand(canvasWidth / 10, canvasHeight / 1.5),  .loc.y = getRand(canvasHeight / 10, canvasHeight / 1.5),  .rad = getRand(4, 10),  .ofball.r = getRand(1, 255),  .ofball.g = getRand(1, 255),  .ofball.b = getRand(1, 255),  .vel.x = 3.14159 \* getRand(1, 10),  };  return shar;  };  struct ball updateBall(struct ball shar, int x1, long dt, long t, int cWidth, int cHeight)  {  shar.vel.y = sin(x1);  shar.loc.x += shar.vel.x;  shar.loc.y += shar.loc.y;  if (shar.loc.x + shar.rad >= cWidth || shar.loc.x - shar.rad <= 0)  {  shar.vel.x = -shar.vel.x;  }  if (shar.loc.y + shar.rad >= cHeight || shar.loc.y - shar.rad <= 0)  {  shar.loc.y = 0.2 \* shar.loc.y;  shar.vel.y = 0.5 \* shar.vel.y;  }  return shar;  };  void drowball(struct ball shar)  {  Canvas\_setColorRGB(shar.ofball.r, shar.ofball.g, shar.ofball.b);  Canvas\_fillCircle(shar.loc.x, shar.loc.y, shar.rad);  };  int getRand(int min, int max)  {  int a = rand() % (max - min + 1) + min;  return a;  };  float length(struct vec2D v)  {  float length = 0;  length = sqrt(pow(v.x, 2) + pow(v.y, 2));  return length;  };  struct vec2D negative(struct vec2D v)  {  struct vec2D copy =  {  copy.x = -v.x,  copy.y = -v.y,  };  return copy;  };  struct vec2D add(struct vec2D a, struct vec2D b)  {  struct vec2D copy =  {  copy.x = a.x + b.x,  copy.y = a.y + b.y,  };  return copy;  };  struct vec2D mult(struct vec2D v, float n)  {  struct vec2D newVec = v;  newVec.x \*= n;  newVec.y \*= n;  return newVec;  };  struct vec2D norm(struct vec2D v)  {  float length = 0;  length = sqrt(pow(v.x, 2) + pow(v.y, 2));  struct vec2D copy =  {  copy.x = v.x / fabs(length),  copy.y = v.y / fabs(length),  };  return copy;  };  float distance(struct vec2D a, struct vec2D b)  {  float distance = 0;  distance = sqrt(pow((b.x - a.x), 2) + pow((b.y - a.y), 2));  return distance;  };  float angle(struct vec2D v)  {  float angle = acos((v.x \* 1) / (sqrt(pow(v.x, 2) + pow(v.y, 2))));  return angle;  };  struct vec2D rotate(struct vec2D v, float angle)  {  struct vec2D copy =  {  copy.x = v.x \* cos(angle) - v.y \* sin(angle),  copy.y = v.x \* cos(angle) + v.y \* sin(angle),  };  return copy;  };  struct vec2D fromPolar(float angle, float length)  {  struct vec2D fromPolar =  {  fromPolar.x = length \* cos(angle),  fromPolar.y = length \* sin(angle),  };  return fromPolar;  };  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);  };  void mainTest()  {  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 newvector = {-1, 0};  assert(equals(negative(a), newvector));  struct vec2D ex = {3, -2};  assert(equals(add(a, b), ex));  //--------------------------  struct vec2D polar = {length(a) \* cos(90), length(a) \* sin(90)};  assert(equals(fromPolar(90, length(a)), polar));  //--------------------------  struct vec2D k = {2, 0};  assert(equals(mult(a, 2), k));  //--------------------------  assert(fequals(angle(a), acos(1)));  //---------------------------  struct vec2D rot = {1 \* cos(90 + acos(1)), 1 \* sin(90 + acos(1))};  assert(equals(rotate(a, 90), rot));  //---------------------------  struct vec2D s = {0.6, 0.8};  assert(equals(norm(c), s));  //---------------------------  float m = sqrt(5);  assert(fequals(distance(a, b), m));  } |

**￼Приклади результатів**

****

**Висновки**

Виконавши дану лабораторну роботу ми навчилися створювати користувацькі функції та тестувати їх із різними аргументами.

Під час роботи також навчилися користуватися з аргументами командного рядка, що, сподіваюся, буде дуже дуже доречним.

Програма працює. Тестування виконується. При недостатній кількості аргументів у командному рядку, програма повертає 1.

Компіляція всього коду відбувалася за допомогою компілятора gcc.