

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(ла)  студент(ка) I курсу  групи КП-83  Клименко Ярослав Олександрович  (*прізвище, ім’я, по батькові*)    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):

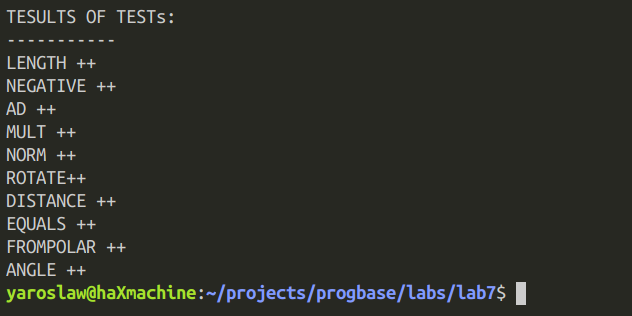
**struct** Vec2D  
{  
 **float** x;  
 **float** y;  
};

**Тексти коду програм**

|  |
| --- |
| **main.c** |
| **// Компілювати за допомогою:**  **// gcc main.c -lprogbase -lm**  **#include <stdio.h> // Для друку в термінал**  **#include <math.h> // Для математичних функцій**  **#include <stdlib.h> // Деякі додаткові функції**  **#include <progbase.h>**  **#include <time.h>**  **#include <progbase/console.h>**  **#include <progbase/canvas.h>**  **#include <string.h>**  **#include <assert.h>**  **#include <stdbool.h>**  **#include <progbase.h>**  ***struct* Vec2D**  **{**  ***float* x;**  ***float* y;**  **};**  ***struct* Color**  **{**  ***unsigned* *char* red, green, blue;**  **};**  ***struct* Ball**  **{**  ***struct* Vec2D loc;**  ***struct* Vec2D speed;**  ***int* radius;**  ***struct* Color bColor;**  **};**  ***struct* Point**  **{**  ***struct* Vec2D loc;**  ***struct* Color color;**  ***int* radius;**  ***float* power;**  ***struct* Vec2D distance;**  **};**  ***struct* Ball createBall(*int* canvasWidth, *int* canvasHeight);**  ***struct* Point createPoint(*int* canvasWidth, *int* canvasHeight);**  ***struct* Ball updateBall(*struct* Ball b, *int* cWidth, *int* cHeight, *int* gravitation, *int* pointAverage, *struct* Point p[pointAverage]);**  ***struct* Point updatePoint(*struct* Ball b, *struct* Point p, *int* cWidth, *int* cHeight);**  ***void* drawBall(*struct* Ball b);**  ***void* drawPoint(*struct* Point p);**  ***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);//**  ***\_Bool* equals(*struct* Vec2D a, *struct* Vec2D b);//+**  ***int* randRGB();**  ***int* randInt(*int* min, *int* max);**  ***float* randFloat(*float* min, *float* max);**  ***void* mainTest();**  ***int* main(*int* argc, *char* \*argv[])**  **{**  **Console\_clear();**  **if (argc == 1)**  **{**  **return 0;**  **}**  **else if (strcmp(argv[1], "-n") == 0)**  **{**  **srand(time(0));**  ***struct* ConsoleSize cs = Console\_size();**  **const *int* w = cs.columns;**  **const *int* h = cs.rows \* 2;**  **const *int* delay = 66;**  **const *int* gravitation = 4;**  **const *int* pointAverage = atoi(argv[2]);**  **const *int* ballAverage = 10;**  ***struct* Point points[pointAverage];**  **for (*int* i = 0; i < pointAverage; i++)**  **{**  **points[i] = createPoint(w, h);**  **}**  ***struct* Ball ball[ballAverage];**  **for (*int* i = 0; i < ballAverage; i++)**  **{**  **ball[i] = createBall(w, h);**  **};**  ***int* lich = 0;**  **Canvas\_setSize(w, h);**  **Canvas\_invertYOrientation();**  **do**  **{**  **lich++;**  **for (*int* i = 0; i <= lich / 10; i++)**  **{**  **if (i == ballAverage)**  **{**  **break;**  **}**  **ball[i] = updateBall(ball[i], w, h, gravitation, pointAverage, points);**  **}**  **Canvas\_beginDraw();**  **for (*int* i = 0; i <= lich; i += 10)**  **{**  **if (i / 10 == ballAverage)**  **{**  **break;**  **}**  **drawBall(ball[i / 10]);**  **}**  **for (*int* k = 0; k < pointAverage; k++)**  **{**  **drawPoint(points[k]);**  **}**  **Canvas\_endDraw();**  **sleepMillis(delay);**  **} while (!Console\_isKeyDown());**  **return 0;**  **}**  **else if (strcmp(argv[1], "-t") == 0)**  **{**  **// CREATA**  **mainTest();**  **//**  **}**  **}**  ***struct* Ball updateBall(*struct* Ball b, *int* cWidth, *int* cHeight, *int* gravitation, *int* pointAverage, *struct* Point p[pointAverage])**  **{**  **for (*int* j = 0; j < pointAverage; j++)**  **{**  **p[j].distance.x = b.loc.x - p[j].loc.x;**  **p[j].distance.y = b.loc.y - p[j].loc.y;**  **p[j].power = sqrt(pow(p[j].distance.x, 2) + pow(p[j].distance.y, 2));**  **if (p[j].loc.x > b.loc.x)**  **{**  **b.speed.x += p[j].distance.x \* gravitation / pow(p[j].power, 2);**  **}**  **if (p[j].loc.x < b.loc.x)**  **{**  **b.speed.x -= p[j].distance.x \* gravitation / pow(p[j].power, 2);**  **}**  **if (p[j].loc.y > b.loc.y)**  **{**  **b.speed.y += p[j].distance.y \* gravitation / pow(p[j].power, 2);**  **}**  **if (p[j].loc.y < b.loc.y)**  **{**  **b.speed.y -= p[j].distance.y \* gravitation / pow(p[j].power, 2);**  **}**  **}**  **if (b.loc.x + b.radius >= cWidth || b.loc.x - b.radius <= 0)**  **{**  **b.speed.x = -b.speed.x;**  **}**  **if (b.loc.y + b.radius >= cHeight || b.loc.y - b.radius <= 0)**  **{**  **b.speed.y = -b.speed.y;**  **}**  **if (b.speed.x == 0 || b.speed.y == 0)**  **{**  **b.speed.x += 3;**  **b.speed.y += 3;**  **}**  **b.loc.x += b.speed.x;**  **b.loc.y += b.speed.y;**  **return b;**  **}**  ***float* length(*struct* Vec2D v)**  **{**  ***float* length = sqrt(v.x \* v.x + v.y \* v.y);**  **return length;**  **}**  ***struct* Vec2D negative(*struct* Vec2D v)**  **{**  **v.x = -v.x;**  **v.y = -v.y;**  **return v;**  **}**  ***struct* Vec2D norm(*struct* Vec2D v)**  **{**  ***struct* Vec2D bufVec = v;**  **if(length(v)==0){**  **bufVec.x = 0;**  **bufVec.y = 0;**  **return bufVec;**  **}**  **bufVec.x = v.x / length(v);**  **bufVec.y = v.y / length(v);**  **return bufVec;**  **}**  ***float* distance(*struct* Vec2D a, *struct* Vec2D b)**  **{**  ***float* distance = sqrt(pow(b.x - a.x, 2) + pow(b.y - a.y, 2));**  **return distance;**  **}**  ***\_Bool* equals(*struct* Vec2D a, *struct* Vec2D b)**  **{**    ***float* lengthA = length(a);**  ***float* lengthB = length(b);**  **if (lengthA - lengthB < 0.01){**  **return 1;**  **}else**  **return 0;**  **}**  ***struct* Vec2D mult(*struct* Vec2D v, *float* n)**  **{**  ***struct* Vec2D bufVec = v;**  **bufVec.x = v.x \* n;**  **bufVec.y = v.y \* n;**  **return bufVec;**  **}**  ***struct* Vec2D add(*struct* Vec2D a, *struct* Vec2D b)**  **{**  ***struct* Vec2D bufVec = a;**  **bufVec.x = a.x + b.x;**  **bufVec.y = a.y + b.y;**  **return bufVec;**  **}**  ***float* angle(*struct* Vec2D v)**  **{**  **if(v.x == 0){**  **return 3.14/2;**  **}**  ***float* angle = atan2(v.y , v.x);**  **return angle;**  **}**  ***struct* Vec2D frompolar(*float* angle, *float* length)**  **{**  ***struct* Vec2D frompolar;**  **frompolar.x = length \* cos(angle);**  **frompolar.y = length \* sin(angle);**  **return frompolar;**  **}**  ***struct* Vec2D rotate(*struct* Vec2D v, *float* angle)**  **{**  **angle += atan(v.y / v.x);**  ***float* len = length(v);**  ***struct* Vec2D rotated = frompolar(angle, len);**  **return rotated;**  **}**  ***void* mainTest()**  **{**    ***struct* Vec2D v[10];**  **v[0].x = 1;**  **v[0].y = 1;**    **v[1].x = 2;**  **v[1].y = 0;**    **v[2].x = 0;**  **v[2].y = 0;**  **puts("TESULTS OF TESTs:\n-----------");**  **//LENGTH**  **assert(length(v[0]) - sqrt(2.0)< 0.01);**  **assert(!(length(v[0])==length(v[2])));**  **assert(length(v[0])==length(v[0]));**  **puts("LENGTH ++");**  **//NEGATIVE**  **v[2].x = -v[0].x;**  **v[2].y = -v[0].y;**  **assert(equals(v[0], negative(v[2])));**  **assert(equals(v[0], negative(v[1])));**  **assert(equals(negative(v[0]),negative(v[0])));**  **puts("NEGATIVE ++");**  **//AD**  **v[3].x = v[0].x + v[2].x;**  **v[3].y = v[0].y + v[2].y;**  **assert(equals(add(v[0], v[2]),v[3]));**  **assert(equals(add(v[0],v[1]), add(v[0],v[1])));**  **assert(equals(add(v[3], v[2]),v[1]));**  **puts("AD ++");**  **//MULT**  **v[2].x = -v[0].x;**  **v[2].y = -v[0].y;**  **assert(equals(mult(v[2], -1), v[0]));**  **assert(!equals(mult(v[2], -1), v[3]));**  **assert(equals(mult(v[2], -1), mult(v[2], -1)));**  **puts("MULT ++");**  **//NORM**  **assert(equals(norm(v[0]), norm(v[0])));**  **assert(equals(norm(v[0]), norm(v[1])));**  **puts("NORM ++");**  **//ROTATE**  **v[0].x =1;**  **v[0].x =0;**  **v[1] = rotate(v[0], 3.14);**  **v[2].x = -1;**  **v[2].y = 0;**  **assert(equals(v[0], v[1]));**  **v[3] = rotate(v[1], 3.14);**  **assert(fabs(v[0].x - v[3].x) < 0.01 );**  **puts("ROTATE++");**    **//DISTANCE**  **v[0].x = 0;**  **v[0].y = 1;**    **v[1].x = 0;**  **v[1].y = 1;**  **assert(distance(v[0], v[1])==0);**    **v[1].x = 0;**  **v[1].y = 0;**  **assert(distance(v[0], v[1])==1);**  **puts("DISTANCE ++");**  **//EQUALS**  **v[0].x = 0;**  **v[0].y = 1;**    **v[1].x = 0;**  **v[1].y = 1;**  **v[2].x = 3;**  **assert(equals(v[0], v[1]));**  **assert(!equals(v[2], v[1]));**  **puts("EQUALS ++");**  **//FROMPOLAR**  **v[0].x = 1;**  **v[0].y = 0;**  **v[1].y = 0;**  **assert(equals(frompolar(0, 1), v[0]));**  **puts("FROMPOLAR ++");**  **//ANGLE**  **v[0].x =1;**  **v[0].y =0;**  **v[1] = rotate(v[0], 3.14);**  **v[2].x = 0;**  **v[2].y = 1;**  **assert(fabs(angle(v[0]))< 0.1);**  **assert(fabs(angle(v[2]) - 3.14/2)< 0.1);**  **assert(fabs(angle(v[1]) - 3.14)< 0.1);**  **puts("ANGLE ++");**  **}**  ***void* drawBall(*struct* Ball Ball)**  **{**  **Canvas\_setColorRGB(Ball.bColor.red, Ball.bColor.green, Ball.bColor.blue);**  **Canvas\_fillCircle(Ball.loc.x, Ball.loc.y, Ball.radius);**  **}**  ***void* drawPoint(*struct* Point p)**  **{**  **Canvas\_setColorRGB(p.color.red, p.color.green, p.color.blue);**  **Canvas\_fillCircle(p.loc.x, p.loc.y, p.radius);**  **}**  ***int* randRGB()**  **{**  ***int* a = (rand() % 255 + 1);**  **return a;**  **}**  ***int* randInt(*int* min, *int* max)**  **{**  ***int* a = (rand() % (max - min + 1) + min);**  **return a;**  **}**  ***float* randFloat(*float* min, *float* max)**  **{**  **max \*= 100;**  **min \*= 100;**  ***int* a = (rand() % ((*int*)max - (*int*)min + 1) + min);**  ***float* res = a / 100;**  **return res;**  **}**  ***struct* Point createPoint(*int* canvasWidth, *int* canvasHeight)**  **{**  ***struct* Point p;**  **p.radius = 2;**  **p.loc.x = rand() % (canvasWidth - p.radius + 1 - 10) + 10;**  **p.loc.y = rand() % (canvasHeight - p.radius + 1 - 10) + 10;**  **p.color.red = randRGB();**  **p.color.green = randRGB();**  **p.color.blue = randRGB();**  **p.power = 1;**  **return p;**  **}**  ***struct* Ball createBall(*int* canvasWidth, *int* canvasHeight)**  **{**  ***struct* Ball b;**  **b.radius = rand() % (5 + 1 - 3) + 3;**  **b.loc.x = canvasWidth / 2;**  **b.loc.y = b.radius + 1;**  **b.bColor.red = randRGB();**  **b.bColor.green = randRGB();**  **b.bColor.blue = randRGB();**  **b.speed.x = 0;**  **b.speed.y = 3;**  **return b;**  **}** |

**Приклади результатів**

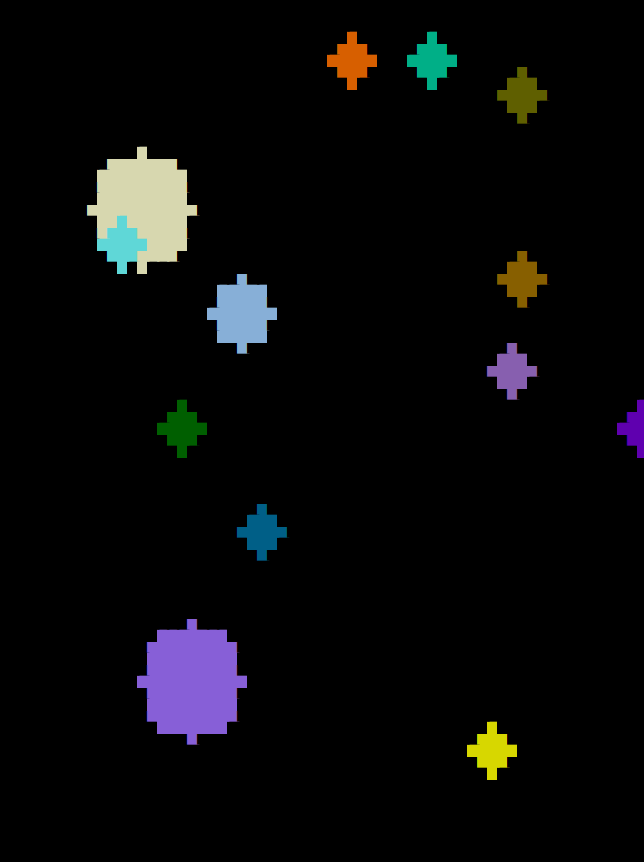
Тести:



Запит для нормального режиму роботи:



Результат:



(10 точок)

**Висновки**

Під час виконання даної лабораторної роботи, я ознайомився з принципами роботи із функціями у мові програмування С.

За допомогою тестів, було просто перевірити чисті функції. Інші ж функції спростили процес обрахунку фізики руху об”єктів та їх вивід.

Реалізація програми відбулася за допомогою вкладених циклів, а також структур даних. При виконанні завдання, я користувався бібліотекою для мови С - libprogbase.h та її функціями, що допомогли спростити процес малювання фігур. Реалізація переміщення об”єктів відбулася за допомогою векторів та структур даних, що допомогли ними керувати.

Компіляція всього коду відбувалася за допомогою компілятора gcc.