**кафедра комп’ютерних технологій, ДНУ**

**2016/2017 н.р.**

**ЗВІТ З ЛАБОРАТОРНОЇ РОБОТИ №3**

**за курсом «Програмування»**

**студента групи ПК-15-1**

**Гулого Тараса Олександровича**

# 1. Постановка задачі

**Постановка задачі**.

Скласти об’єктно-орієнтовану програму на мові C++, яка в діалоговому режимі керує графічними об’єктами, що відображаються на екрані дисплея.

Програма повинна підтримувати такі загальні елементи поведінки графічних об’єктів:

1. Активізація/візуалізація за вибором.
2. Переміщення зі слідом/без.
3. Відновлення початкового стану образа.
4. Зміна кольору.
5. Зміна стану видимий/невидимий.
6. Агрегація, тобто утворення нових об’єктів з вже створених. При агрегації об’єктів забезпечити можливість агрегації агрегатів.

Програма повинна надати користувачеві можливість зберігати поточну конфігурацію програми у вказаний користувачем текстовий файл на диску і завантажувати поточну конфігурацію програми із вказаного користувачем текстового файлу на диску.

**Індивідуальне завдання №8**

Номер 8 у двійковому коді записується так 01000. Отже, після завершення роботи з об’єктом він зникає, об’єкт в автоматичному режимі рухається за певною траєкторією, зміна кольору та деформація об’єктів за командою, зборка/запам’ятання агрегованого образа дублюванням.

# 2. Опис розв’язку

**Основна концепція**.

Головна програма буде представляти собою меню, яке дозволяє робити операції над головним списком, де зберігатимуться всі фігури поточної програми. Користувач може видаляти, додавати елементи в список/з списку, зберігати та завантажувати елементи списку з текстового файлу. Також для елементів головного списку є функція переходу до режиму конкретної фігури.

Перехід до режиму фігури здійснюватиме деяка функція, яка візуалізує фігури та надає користувачу вибір фігур над якими здійснюватимуться операції над фігурами. Перед входом до режиму фігури робитиметься копія фігури над якою будуть проводитися операції.

Після виходу з режиму операцій користувачу буде запропоновано відновити образ фігури до того як вона була змінена у режимі фігури, дана можливість буде реалізовувати задачу відновлення образу. У режимі фігури буде реалізовані функції, які запропоновані постановкою задачі. Ілюстративно принцип програми можна зобразити так:

**.  
.  
.**

Головний список

**.  
.  
.**

?

Фігури

Функція вибору фігури

Функція-меню роботи з фігурою

копія

*Рис.1 Принцип роботи програми*

**Функція меню роботи з головним списком фігур**.

У глобальній області видимості було оголошено головний елемент списку Head. Даний елемент є головою списку він практично не змінний і доступний всім функціям програми. Головна програма запускає функцію-меню роботи з головним списком. Ця функція фактично викликає функції які пов’язані зі списком та його елементами. В залежності від вибору програма методом розгалуження запускає данні функції. Серед функцій роботи з головним списком такі:

1. ***Додати нову фігуру.*** Функція додає нову фігуру у кінець списку або створює новий список, у цій функції реалізована концепція вибору фігури та саме одного із елементів поліморфізму. Створюється новий вказівник типу Shape який вказує на фігури наслідники цього класу, потім цей вказівник заноситься до нового елементу-списку і зберігається там до видалення. У даній функції також реалізована можливість додання агрегату. Після створення фігури відбувається візуалізація її.
2. ***Видалити фігуру.*** Функція протилежна до додавання. Вона видаляє фігуру зі списку або видаляє весь список в залежності від вмісту. Ця та наступні функції не визиваються якщо немає елементів в списку.
3. ***Зберегти поточну конфігурацію.*** Функція, яка підключає файл та виводить всі елементи які є в списку. Дана процедура реалізована за допомогою поліморфної функції ofile, яка у кожного з наслідників виводить ключові елементи за якими потім можна відновити фігуру.
4. ***Відновити конфігурацію.*** Функція, яка підключає файл та вводить звідти всі елементи які є у файлі. Дана процедура реалізована за допомогою поліморфної функції ifile, яка у кожного з наслідників вводить ключові елементи за якими генерується фігура.

**Функція вибору**.

Для реалізації роботи з конкретною фігурою зі списку існує функція яка вибирає той елемент списку з яким працюватиме меню роботи з фігурою дана функція представляє собою нескінченний цикл який проганяє всі фігури по колу доти доки користувач не вибере фігуру.

**Функція меню роботи з фігурою**.

Після повернення елементу фігури з якою буде працювати функція-меню роботи з фігурою робиться копія фігури яка буде передана у функцію. Це виконується для того, щоб після виконання деяких маніпуляцій над фігурою ми змогли повернутися до попереднього образу. У функції роботи з фігурою можливий вибір функцій які виконують задачі лабораторної роботи, а саме:

1. ***Зміна кольору.*** Так як за умово індивідуального завдання змінювати колір необхідно по команді у класи загальних фігур було введено функцію colorch яка змінює колір на переданий. Функція фактично не є віртуальною, так як поле «колір» є спільним для всіх класів.
2. ***Рух фігури.*** За навчальним прикладом graphics\_in\_console було реалізовано схожу функцію яка фактично не є віртуальною, та яка рухає фігуру доти, доки користувач не натисне esc.
3. ***Рух фігури зі слідом.*** За навчальним прикладом graphics\_in\_console було реалізовано схожу функцію яка фактично не є віртуальною, та яка рухає фігуру доти, доки користувач не натисне esc. На відміну від попередньої у функції немає метода hide(), який би витирав попередню конфігурацію тому фігура залишає слід.
4. ***Рух фігури за траєкторією.*** Під траєкторію розумітимемо послідовність команд які рухатимуть фігуру автоматично. Дану можливість можна реалізувати з циклом та без тому було обрана організувати таку можливість двома способами. Користувач може або задати певну послідовність команд або задати певну послідовність команд які виконуватимуться циклічно певну кількість ітерацій.
5. ***Деформація фігури.*** Під деформацією можна розуміти різні перетворення які змінюють контур фігури. У даній програмі під деформацією розуміється зменшення на 20 відсотків. Тобто кожне опорне значення зменшується на 20 відсотків відносно початкового. Ця функція є віртуальною так як для кожної фігури зменшення відбувається по-різному.

Після завершення роботи з фігурою користувачу пропонується відновити початковий образ фігури. Якщо користувач відмовиться збережеться нова конфігурація фігури. Фігура візуалізована безпосередньо при вході у певну функцію перетворення фігури. Така реалізація обумовлена проблемою system(“cls”) (див. проблеми)

**Модуль фігур**.

Так як у постановці не стоїть задача сховати реалізацію від користувача всі функції оголошені і реалізовані в хедер-файлі.

У хедер-файлі є оголошення всіх необхідних констант та функцій якими користуються функції головної програми.

Структура класів якими користується користувач схожа до прикладу graphic\_in\_console існує головний абстрактний клас фігур у якого є декілька наслідників. Кожний із наслідників це нова фігура, яка має свої характеристики. У загального класу є свої функції які виконується поза залежності від власних характеристик.

Функції яким необхідні характеристики фігури є віртуальними, тобто якщо звертання до об’єктів відбувається через вказівник з розширенням загального(абстарактного) класу. ТО в залежності від вмісту можна виконати певні функції класів-наслідників.

Таким функціям як: move, move\_trace, colorch, hide, show не потрібні додаткові характеристики так як вони працюють з елементами об’єктів загального класу.

Функціям: ifile ofile deform draw і copy необхідні додаткові характеристики для до визначення дій які необхідно виконувати отже вони віртуальні.

У модулі також міститься оголошення елементів списку фігур з якими працюють функції головної програми та класи фігур.

**Клас «Агрегатів»**.

Одним із задач даної лабораторної роботи є створення класу-агрегату, який містить у собі список фігур які вважаються одним цілим.

Для реалізації даного класу необхідно, щоб головний клас був повністю абстрактним тобто щоб всі функції в ньому були віртуальні. Об’єкт агрегат буде виконувати всі свої функції над своєю характеристикою тобто над списком фігур всередині класу. В свою чергу агрегат не є окремим класом так як нам необхідно використовувати його в головній програмі поруч з іншими фігурами.

Для створення об’єкта-агрегата у функції створення нових елементів головного списку є можливість створити фігуру агрегат. Дана можливість реалізована в окрему функцію. Згідно з поставленою задачею агрегат створюється копіюванням тобто запускається цикл який створює новий елемент головного списку, після кожної ітерації задається питання чи необхідно створювати ще фігури які війдуть до агрегату, користувач може вийти з циклу якщо всі фігури на місці. Після завершення циклу до конструктору подається вказівник на перший елемент який було створено користувачем у режимі агрегації, тобто ми маємо підсписок списку, у якому є всі об’єкти створені в режимі агрегації. Конструктор створює елементи внутрішнього списку копіюючи їх із головного підсписку.

Після створення користувачу надається можливість в останній раз порухати кожний із об’єктів внутрішнього списку об’єктів агрегату. Ця можливість реалізована функцією move\_each яка по черзі дозволяє рухати об’єкти агрегату. Щоб об’єкти не затиралися існує функція redraw яка перемальовує всі елементи списку після кожного руху. Потім на агрегат створюється вказівник загального класу shape який записується у об’єкт елементу списку, який в свою чергу записується в головний список.

Так як режим агрегату є доступним через функцію створення елементу головної програми можливе створення агрегату в агрегаті.

**Додавання нових фігур до програми**.

Не складними маніпуляціями можна легко розширити клас фігур.

*Приклад:*

Нехай нам необхідно додати до фігур, ще дві трикутник і n-кутну зірку, щоб зробити це за даної реалізації програми необхідно відтворити такі етапи:

Етап 1 *Створити клас наслідник який буде забезпечувати необхідні операції над класом. Довизначати віртуальні функції.* Для прикладу з зіркою та трикутником необхідно створити нові класи Nstar і Triangle які реалізовуватимуть зірки та трикутники відповідно. Далі необхідно довизначати віртуальні функції тобто для класів необхідно визначити deform draw і copy. Нехай для трикутника визначається двома зміщеннями відносно основної точки, а зірка буде визначена кількістю кутів та радіусом кола у який буде вона вписана.

Етап 2 *Визначення нових фігур у всіх розгалуженнях які визначають фігури.* Відповідно у головній програмі у кожному розгалуженні необхідно додати фігуру яку ми створили. Також необхідно довизначати фігуру у розгалуженнях класу агрегату який приймає фігури з файлу та записує туди ж.

# 3. Текст вихідної програми

main.cpp

#include "graphics\_in\_console.h"

ShapeList\* Head=NULL;

void cashin();

void cashout();

void ListOptions();

void ShapeOption(Shape\* s);

void moving(Shape\* s);

void moving\_trace(Shape\* s);

void ShapeMode();

void AgregMode();

void moving\_trac(Shape \*s);

void Deleteing(ShapeList\* s);

ShapeList\* Choosing();

int main()

{

system("CHCP 1251>NUL");

cout<<"Вас приветствует лабораторная работа №3 семестра 2"<<endl;

ListOptions();

return 0;

}

ShapeList\* Choosing()

{

ShapeList\* temp=Head;

while(1)

{

system("cls");

Sleep(100);

cout<<"Используйте 'tab'- для перехода на следующюю фигуру и 'space'- для выбора"<<endl;

ShapeList\* s=temp;

s->shape->draw(s->shape->color);

int c=getch();

switch(c)

{

case 9:

{

s->shape->hide();

temp=temp->next;

if(temp==NULL) temp=Head;

continue;

}

case 32:

{

return s;

}

}

}

return NULL;

}

void ListOptions()

{

while(true)

{

cout<<"Меню работы со списокм фигур"<<endl;

cout<<"1-Добавить фигуру"<<endl;

cout<<"2-Удалить фигуру"<<endl;

cout<<"3-Работа с конкретной фигурой"<<endl;

cout<<"4-Вывести конфигурации в файл"<<endl;

cout<<"5-Загрузить конфигурацию из файл"<<endl;

cout<<"0-Выйти из програмы"<<endl;

switch(getch())

{

case'1':

{

ShapeMode();

system("cls");

continue;

}

case'2':

{

if (Head==NULL) {cout<<"Еще нет ни одной фигуры!"<<endl; system("pause"); system("cls"); continue;}

else Deleteing(Choosing());

system("cls");

continue;

}

case'3':

{

if (Head==NULL) {cout<<"Еще нет ни одной фигуры!"<<endl; system("pause"); system("cls"); continue;}

else

{

ShapeList \*ss=Choosing();

Shape\* s=ss->shape->copy();

ShapeOption(ss->shape);

cout<<"Вы хотите вернуть первоночальный образ фигуры?"<<endl;

cout<<"1-да, 2-нет"<<endl;

if(getch()=='1')

{

Shape\* temp= ss->shape;

ss->shape=s;

delete temp;

}

}

system("cls");

continue;

}

case'4':

{

if (Head==NULL) {cout<<"Еще нет ни одной фигуры!"<<endl; system("pause"); system("cls"); continue;}

cashout();

system("cls");

continue;

}

case'5':

{

cashin();

system("cls");

continue;

}

case'0':{return;}

default:{system("cls"); continue;}

}

}

}

void ShapeOption(Shape\* s)

{

Shape \*sold=NULL;

sold=s->copy();

while(true)

{

system("cls");

cout<<"Меню работы с фигурой"<<endl;

cout<<"1-Поменять цвет"<<endl;

cout<<"2-Двигать фигуру"<<endl;

cout<<"3-Двигать по траектории"<<endl;

cout<<"4-Деформировать фигуру"<<endl;

cout<<"5-Двигать со следом"<<endl;

cout<<"0-Выйти в меню работы со списком"<<endl;

switch(getch())

{

case'1':

{

cout<<"Введите идентификатор цвета"<<endl;

int i;

cin>>i;

s->colorch(i);

s->draw(s->color);

getch();

system("cls");

continue;

}

case'2':

{

moving(s);

system("cls");

continue;

}

case'3':

{

moving\_trac(s);

system("cls");

continue;

}

case'4':

{

system("cls");

Sleep(100);

cout<<"Объект был уменьшен на 20%"<<endl;

s->deform();

s->draw(s->color);

getch();

system("cls");

continue;

}

case'5':

{

moving\_trace(s);

system("cls");

continue;

}

case'0':{return;}

}

system("cls");

cin.clear();

\_flushall();

}

}

void moving(Shape\* s)

{

system("cls");

Sleep(100);

cout<<"Двигайтесь стрелочками по завершению нажмите нажмите esc"<<endl;

s->draw(s->color);

char c;

while (c = getch(), c!=27)

{

if (c == '\xe0')

{

c = getch();

switch (c)

{

case LARROW:

s->move(-STEP,0);

break;

case RARROW:

s->move(+STEP,0);

break;

case UARROW:

s->move(0,-STEP);

break;

case DARROW:

s->move(0,+STEP);

break;

}

}

}

}

void ShapeMode()

{

Shape \*s=NULL;

char type;

int c;

system("cls");

cout<<"Какую фигуры вы хотите добавить? "<<endl;

cout<<"c- круг"<<endl;

cout<<"r- прямоугольник"<<endl;

cout<<"s- сегмент"<<endl;

cout<<"n- n-угольная звезда"<<endl;

cout<<"t- треугольник"<<endl;

cout<<"a- режим работы с агрегатом"<<endl;

switch(getch())

{

case'c':

{

int x,y,r;

cout<<"Введите кориданты точки, радиус и цвет "<<endl;

cin>>x>>y>>r>>c;

s=new Circle(x, y, r, c);

type='c';

break;

}

case'r':

{

int x,y,dx,dy;

cout<<"Введите кориданты точки,ее смещение и цвет "<<endl;

cin>>x>>y>>dx>>dy>>c;

s=new Rectangl(x, y, dx, dy, c);

type='r';

break;

}

case'n':

{

int x,y,r,n;

cout<<"Введите кориданты точки, радиус опорного круга, количество острых концов"<<endl;

cin>>x>>y>>r>>n>>c;

s=new Nstar(x, y, r, n, c);

type='n';

break;

}

case's':

{

int x,y,dx,dy;

cout<<"Введите кориданты точки,ее смещение и цвет "<<endl;

cin>>x>>y>>dx>>dy>>c;

s=new Segment(x, y, dx, dy, c);

type='s';

break;

}

case't':

{

int x,y,dx1,dy1,dx2,dy2;

cout<<"Введите кориданты точки, 2 смещения относительно нее и цвет "<<endl;

cin>>x>>y>>dx1>>dy1>>dx2>>dy2>>c;

s=new Triangle(x,y,dx1,dy1,dx2,dy2,c);

type='t';

break;

}

case'a':

{

AgregMode();

return;

}

default:{cout<<"Такой фигуры нет"<<endl; return;}

}

if(Head==NULL) {Head=new ShapeList(s,type);system("cls");Sleep(100);cout<<"Ваша новая фигура:"<<endl; Shape\* s=Head->shape; s->draw(s->color);getch();}

else

{

ShapeList\* temp=Head;

ShapeList\* newest=new ShapeList(s,type);

while(temp->next!=NULL)

temp=temp->next;

temp->next=newest;

system("cls");

Sleep(100);

cout<<"Ваша новая фигура:"<<endl;

Shape\* s=newest->shape;

s->draw(s->color);

getch();

}

}

void AgregMode()

{

cout<<"Вы вошли в режим создания фигуры агрегата"<<endl;

cout<<"Обязательным дерективой являеться создания первой фигуры"<<endl;

cout<<"Далее фигуры будут создаваться пока вы этого не захотите"<<endl;

cout<<"Для создание первой фигуры фигуры аггрегата нажмите любую клавишу"<<endl;

getch();

Shape\* S=NULL;

ShapeMode();

ShapeList\* TempH=Head;

while(TempH->next!=NULL)

TempH=TempH->next;

while(1)

{

cout<<"Хотите добавить еще фигуру?"<<endl;

cout<<"1-да, 2-нет"<<endl;

if(getch()=='1'){ShapeMode(); continue;}

else break;

}

Agregat\* temp= new Agregat(0,0,TempH,0);

temp->redraw();

cout<<"Включен режим перемещений объектов фигуры-агрегата"<<endl;

cout<<"Для начала перемещений фигур нажмите любую клавишу"<<endl;

cout<<"Когда закончите перемещать конкретнкю фигуру нажмите esc"<<endl;

getch();

temp->move\_each();

S=temp;

ShapeList\* AgregObj= new ShapeList(S,'a');

while(TempH->next!=NULL)

TempH=TempH->next;

TempH->next=AgregObj;

system("cls");

cout<<"Фигура агрегат успешно создана копированием"<<endl;

getch();

}

void moving\_trac(Shape \*s)

{

int n=0;

char\* a;

system("cls");

Sleep(100);

cout<<"Введите длину траекторию"<<endl;

cin>>n;

a=new char[n];

cout<<"Введите последовательность из "<<n<<" елементов."<<endl;

cout<<"В последовательности w-вверх d-влево a-вправо s-вниз"<<endl;

cout<<"Поставте последним елементом с если хотите зациклить траекторию"<<endl;

for(int i=0; i<n;i++)

cin>>a[i];

s->draw(s->color);

if(a[n-1]!='c')

{

for(int i=0;i<n;i++)

{

Sleep(200);

switch (a[i])

{

case 'a':

s->move(-STEP,0);

break;

case 'd':

s->move(+STEP,0);

break;

case 'w':

s->move(0,-STEP);

break;

case 's':

s->move(0,+STEP);

break;

}

}

}

else

{

cout<<"Сколько итераций цикла траектории вы хотите проделать?"<<endl;

int j;

cin>>j;

for(int k=0;k<j;k++)

{

for(int i=0;i<n;i++)

{

Sleep(200);

switch (a[i])

{

case 'a':

s->move(-STEP,0);

break;

case 'd':

s->move(+STEP,0);

break;

case 'w':

s->move(0,-STEP);

break;

case 's':

s->move(0,+STEP);

break;

}

}

}

}

cout<<"Фигура заврешила движение."<<endl;

getch();

}

void Deleteing(ShapeList\* s)

{

system("cls");

ShapeList\* temp=Head;

while(temp!=s)

temp=temp->next;

if(temp==Head)

{

if(Head->next!=NULL){Head=Head->next;}

else{Head=NULL;}

}

else

{

ShapeList\* temp2=Head;

while(temp2->next!=temp)

temp2=temp2->next;

temp2->next=temp->next;

}

delete temp;

cout<<"Фигура удалена"<<endl;

getch();

}

void moving\_trace(Shape\* s)

{

system("cls");

Sleep(100);

cout<<"Двигайтесь стрелочками по завершению нажмите нажмите esc"<<endl;

s->draw(s->color);

char c;

while (c = getch(), c!=27)

{

if (c == '\xe0')

{

c = getch();

switch (c)

{

case LARROW:

s->move\_trace(-STEP,0);

break;

case RARROW:

s->move\_trace(+STEP,0);

break;

case UARROW:

s->move\_trace(0,-STEP);

break;

case DARROW:

s->move\_trace(0,+STEP);

break;

}

}

}

}

void cashout()

{

system("cls");

char name[256];

ofstream fout;

cout<<"Введите имя файла"<<endl;

cin>>name;

fout.open(name);

ShapeList\* temp=Head;

while(temp!=NULL)

{

temp->shape->ofile(fout);

temp=temp->next;

}

fout.close();

cout<<"Запись в файл прошла успешно"<<endl;

getch();

}

void cashin()

{

system("cls");

char name[256];

ifstream fin;

cout<<"Введите имя файла"<<endl;

cin>>name;

fin.open(name);

ShapeList\* prew=Head;

if(Head!=NULL)

{

while(prew->next!=NULL)

prew=prew->next;

}

char type;

while(fin>>type)

{

Shape\* newshape;

switch (type)

{

case 'c':{newshape=new Circle(0,0,0,0); break;}

case 'r':{newshape=new Rectangl(0,0,0,0,0); break;}

case 's':{newshape=new Segment(0,0,0,0,0); break;}

case 'n':{newshape=new Nstar(0,0,200,0,0); break;}

case 't':{newshape=new Triangle(0,0,0,0,0,0,0); break;}

case 'a':{fin>>type; newshape=new Agregat(0,0,NULL,0); break;}

}

newshape->ifile(fin);

ShapeList\* newobj= new ShapeList(newshape,type);

if (prew!=NULL){prew->next=newobj;}

else{Head=newobj;}

prew=newobj;

}

fin.close();

cout<<"Загрузка конфигураций прошла успешно"<<endl;

getch();

}

Graphiс\_in\_console.h

#include <conio.h>

#include <iostream>

#include <stdlib.h>

#include "graphics.h"

#include <dos.h>

#include <fstream>

#include <math.h>

using namespace std;

Grfx::Graphics console\_graphics;

const double M\_PI=3.16259265358979323846;

const char LARROW = 'K';

const char RARROW = 'M';

const char UARROW = 'H';

const char DARROW = 'P';

const int BGCOLOR = 0;

const int COLOR = 2;

const int STEP = 10;

class Shape

{

protected:

int x,y;

public:

int color;

Shape(int a, int b, int c): x(a), y(b), color(c) {}

virtual ofstream& ofile(ofstream &fout){return fout;};

virtual ifstream& ifile(ifstream &fin){return fin;};

virtual Shape\* copy()= 0;

virtual void draw(int c) = 0;

virtual void show() { draw(color); }

virtual void colorch(int color1) {color=color1;}

virtual void deform()=0;

virtual void hide() { draw(BGCOLOR); }

virtual void move(int dx, int dy) { hide(); x+=dx; y+=dy; show(); }

virtual void move\_trace(int dx, int dy) { x+=dx; y+=dy; show(); }

};

class Nstar : public Shape

{

int r,n;

public:

Nstar(int a, int b, int ra,int na, int c) : Shape(a,b,c), r(ra), n(na){ show(); }

void draw(int c)

{

float a=0;

int \*xi= new int[2\*n+1];

int \*yi= new int[2\*n+1];

console\_graphics.setcolor(c);

for (int i=0; i<2\*n+1; i++)

{

if(!(i%2))

{

\*(xi+i)=x+r/2\*cos(a\*M\_PI/180);

\*(yi+i)=y+r/2\*sin(a\*M\_PI/180);

}

else

{

\*(xi+i)=x+r\*cos(a\*M\_PI/180);

\*(yi+i)=y+r\*sin(a\*M\_PI/180);

}

a=a+180/n;

}

\*(xi+2\*n)=\*(xi);

\*(yi+2\*n)=\*(yi);

int xlast=\*(xi);

int ylast=\*(yi);

for(int i=1; i<2\*n+1; i++)

{

console\_graphics.line(xlast,ylast,\*(xi+i),\*(yi+i));

xlast=\*(xi+i);

ylast=\*(yi+i);

}

delete[] xi;

delete[] yi;

}

void deform()

{

float dt=r\*0.2;

r=int(r-dt);

}

Shape\* copy()

{

Shape\* su;

int xs=Shape::x;

int ys=Shape::y;

int cs=Shape::color;

int rs=Nstar::r;

int ns=Nstar::n;

su=new Nstar(xs,ys,rs,ns,cs);

return su;

}

virtual ofstream& ofile(ofstream &fout)

{

fout<<'n'<<" "<<x<<" "<<y<<" "<<r<<" "<<n<<" "<<color<<endl;

return fout;

}

ifstream& ifile(ifstream &fin)

{

fin>>x>>y>>r>>n>>color;

return fin;

}

};

class Segment : public Shape

{

int dx,dy;

public:

Segment(int a, int b, int da, int db, int c) : Shape(a,b,c), dx(da), dy(db) { show(); }

void draw(int c)

{

console\_graphics.setcolor(c); console\_graphics.line(x,y,x+dx,y+dy);

}

void deform()

{

float dt=dx\*0.2;

dx=int(dx-dt);

dt=dy\*0.2;

dy=int(dy-dt);

}

Shape\* copy()

{

Shape\* su;

int xs=Shape::x;

int ys=Shape::y;

int cs=Shape::color;

int dxs=Segment::dx;

int dys=Segment::dy;

su=new Segment(xs,ys,dx,dy,cs);

return su;

}

virtual ofstream& ofile(ofstream &fout)

{

fout<<'s'<<" "<<x<<" "<<y<<" "<<dx<<" "<<dy<<" "<<color<<endl;

return fout;

}

ifstream& ifile(ifstream &fin)

{

fin>>x>>y>>dx>>dy>>color;

return fin;

}

};

class Circle : public Shape

{

int r;

public:

Circle(int a, int b, int ra, int c) : Shape(a,b,c), r(ra){ show(); }

void draw(int c)

{

console\_graphics.setcolor(c); console\_graphics.circle(x,y,r);

}

void deform()

{

float dt=r\*0.2;

r=int(r-dt);

}

Shape\* copy()

{

Shape\* su;

int xs=Shape::x;

int ys=Shape::y;

int cs=Shape::color;

int rs=Circle::r;

su=new Circle(xs,ys,rs,cs);

return su;

}

virtual ofstream& ofile(ofstream &fout)

{

fout<<'c'<<" "<<x<<" "<<y<<" "<<r<<" "<<color<<endl;

return fout;

}

ifstream& ifile(ifstream &fin)

{

fin>>x>>y>>r>>color;

return fin;

}

};

class Rectangl : public Shape

{

int dx,dy;

public:

Rectangl(int a, int b, int da, int db, int c) : Shape(a,b,c), dx(da), dy(db) { show(); }

void draw(int c)

{

console\_graphics.setcolor(c); console\_graphics.rectangle(x,y,x+dx,y+dy);

}

void deform()

{

float dt=dx\*0.2;

dx=int(dx-dt);

dt=dy\*0.2;

dy=int(dy-dt);

}

Shape\* copy()

{

Shape\* su;

int xs=Shape::x;

int ys=Shape::y;

int cs=Shape::color;

int dxs=Rectangl::dx;

int dys=Rectangl::dy;

su=new Rectangl(xs,ys,dx,dy,cs);

return su;

}

virtual ofstream& ofile(ofstream &fout)

{

fout<<'r'<<" "<<x<<" "<<y<<" "<<dx<<" "<<dy<<" "<<color<<endl;

return fout;

}

ifstream& ifile(ifstream &fin)

{

fin>>x>>y>>dx>>dy>>color;

return fin;

}

};

class Triangle : public Shape

{

int dx1,dy1,dx2,dy2;

public:

Triangle(int a, int b, int da1, int db1,int da2, int db2, int c) : Shape(a,b,c), dx1(da1), dy1(db1), dx2(da2), dy2(db2){ show(); }

void draw(int c)

{

console\_graphics.setcolor(c);

console\_graphics.line(x,y,x+dx1,y+dy1);

console\_graphics.line(x,y,x+dx2,y+dy2);

console\_graphics.line(x+dx1,y+dy1,x+dx2,y+dy2);

}

void deform()

{

float dt=dx1\*0.2;

dx1=int(dx1-dt);

dt=dy1\*0.2;

dy1=int(dy1-dt);

dt=dx2\*0.2;

dx2=int(dx2-dt);

dt=dy2\*0.2;

dy2=int(dy2-dt);

}

Shape\* copy()

{

Shape\* su;

int xs=Shape::x;

int ys=Shape::y;

int cs=Shape::color;

int dx1s=Triangle::dx1;

int dy1s=Triangle::dy1;

int dx2s=Triangle::dx2;

int dy2s=Triangle::dy2;

su=new Triangle(xs,ys,dx1s,dy1s,dx2s,dy2s,cs);

return su;

}

virtual ofstream& ofile(ofstream &fout)

{

fout<<'t'<<" "<<x<<" "<<y<<" "<<dx1<<" "<<dy1<<" "<<dx2<<" "<<dy2<<" "<<color<<endl;

return fout;

}

ifstream& ifile(ifstream &fin)

{

fin>>x>>y>>dx1>>dy1>>dx2>>dy2>>color;

return fin;

}

};

struct ShapeList

{

char name;

Shape \* shape;

ShapeList\* next;

ShapeList(Shape\* s, char type)

{

shape=s;

name=type;

next=NULL;

}

};

class Agregat : public Shape

{

public:ShapeList \*Headake;

Agregat(int a, int b, ShapeList\* List, int c) : Shape(a,b,c)

{

ShapeList\* temp=List;

if(temp==NULL){return;}

ShapeList\* prew=NULL;

Shape\* h;

h=temp->shape->copy();

Headake=new ShapeList(h,temp->name);

prew=Headake;

temp=temp->next;

while(temp!=NULL)

{

ShapeList\* now=NULL;

Shape\* h;

h=temp->shape->copy();

now=new ShapeList(h,temp->name);

prew->next=now;

prew=now;

temp=temp->next;

}

show();

}

void draw (int c)

{

ShapeList\* temp= Headake;

while(temp!=NULL)

{

temp->shape->draw(temp->shape->color);

temp=temp->next;

}

}

void show ()

{

ShapeList\* temp= Headake;

while(temp!=NULL)

{

temp->shape->show();

temp=temp->next;

}

}

void hide()

{

ShapeList\* temp=Headake;

while(temp!=NULL)

{

temp->shape->hide();

temp=temp->next;

}

}

void deform()

{

ShapeList\* temp=Headake;

while(temp!=NULL)

{

temp->shape->deform();

temp=temp->next;

}

}

void colorch(int color1)

{

ShapeList\* temp=Headake;

while(temp!=NULL)

{

temp->shape->colorch(color1);

temp=temp->next;

}

}

void move(int dx, int dy)

{

ShapeList\* temp=Headake;

while(temp!=NULL)

{

temp->shape->move(dx,dy);

temp=temp->next;

}

}

void move\_trace(int dx, int dy)

{

ShapeList\* temp=Headake;

while(temp!=NULL)

{

temp->shape->move\_trace(dx,dy);

temp=temp->next;

}

}

void move\_each()

{

ShapeList\* temp2=Headake;

while(temp2!=NULL)

{

system("cls");

redraw();

Sleep(200);

int cl= temp2->shape->color;

temp2->shape->draw(6);

Sleep(200);

temp2->shape->draw(cl);

redraw();

cout<<"Двигайтесь стрелочками по завершению нажмите нажмите esc"<<endl;

char c;

while (c = getch(), c!=27)

{

if (c == '\xe0')

{

c = getch();

switch (c)

{

case LARROW:

temp2->shape->move(-STEP,0);

break;

case RARROW:

temp2->shape->move(+STEP,0);

break;

case UARROW:

temp2->shape->move(0,-STEP);

break;

case DARROW:

temp2->shape->move(0,+STEP);

break;

}

redraw();

}

}

temp2=temp2->next;

}

}

Shape\* copy()

{

Shape\* su=new Agregat(Shape::x,Shape::y,Headake,Shape::color);

return su;

}

void redraw()

{

ShapeList\* t=Headake;

if(t==NULL) return;

while(t!=NULL)

{

Shape\* r;

r=t->shape;

Sleep(100);

r->draw(r->color);

t=t->next;

}

}

virtual ofstream& ofile(ofstream &fout)

{

fout<<'a'<<endl;

fout<<'{'<<endl;

ShapeList\* temp=Headake;

while(temp!=NULL)

{

temp->shape->ofile(fout);

temp=temp->next;

}

fout<<'}'<<endl;

return fout;

}

ifstream& ifile(ifstream &fin)

{

char type;

ShapeList\* temp=NULL;

fin>>type;

while(type!='}')

{

Shape\* newshape;

switch (type)

{

case 'c':{newshape=new Circle(0,0,0,0); break;}

case 'r':{newshape=new Rectangl(0,0,0,0,0); break;}

case 's':{newshape=new Segment(0,0,0,0,0); break;}

case 'n':{newshape=new Nstar(0,0,0,200,0); break;}

case 't':{newshape=new Triangle(0,0,0,0,0,0,0); break;}

case 'a':{fin>>type;newshape=new Agregat(0,0,NULL,0); break;}

}

newshape->ifile(fin);

ShapeList\* newobj= new ShapeList(newshape,type);

if (temp!=NULL){temp->next=newobj;}

else{Headake=newobj;}

temp=newobj;

fin>>type;

}

return fin;

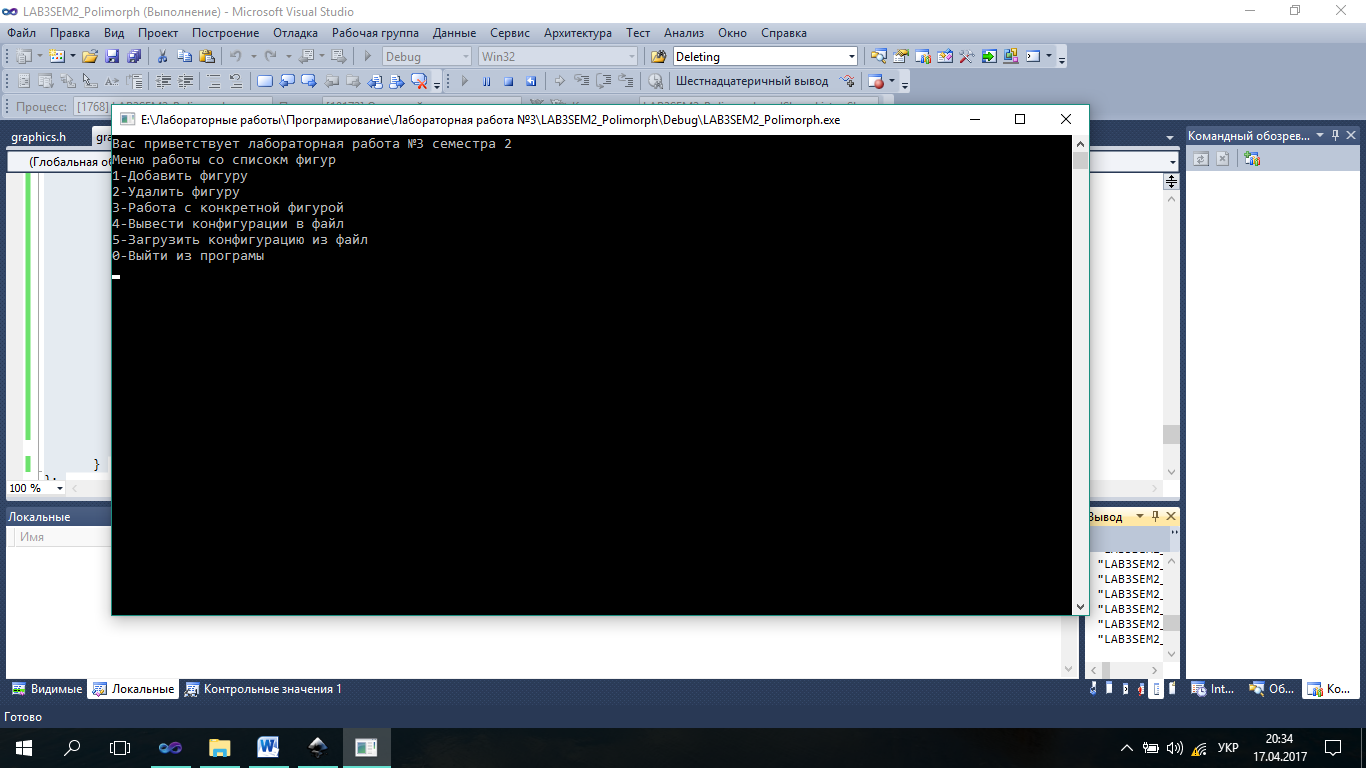
}

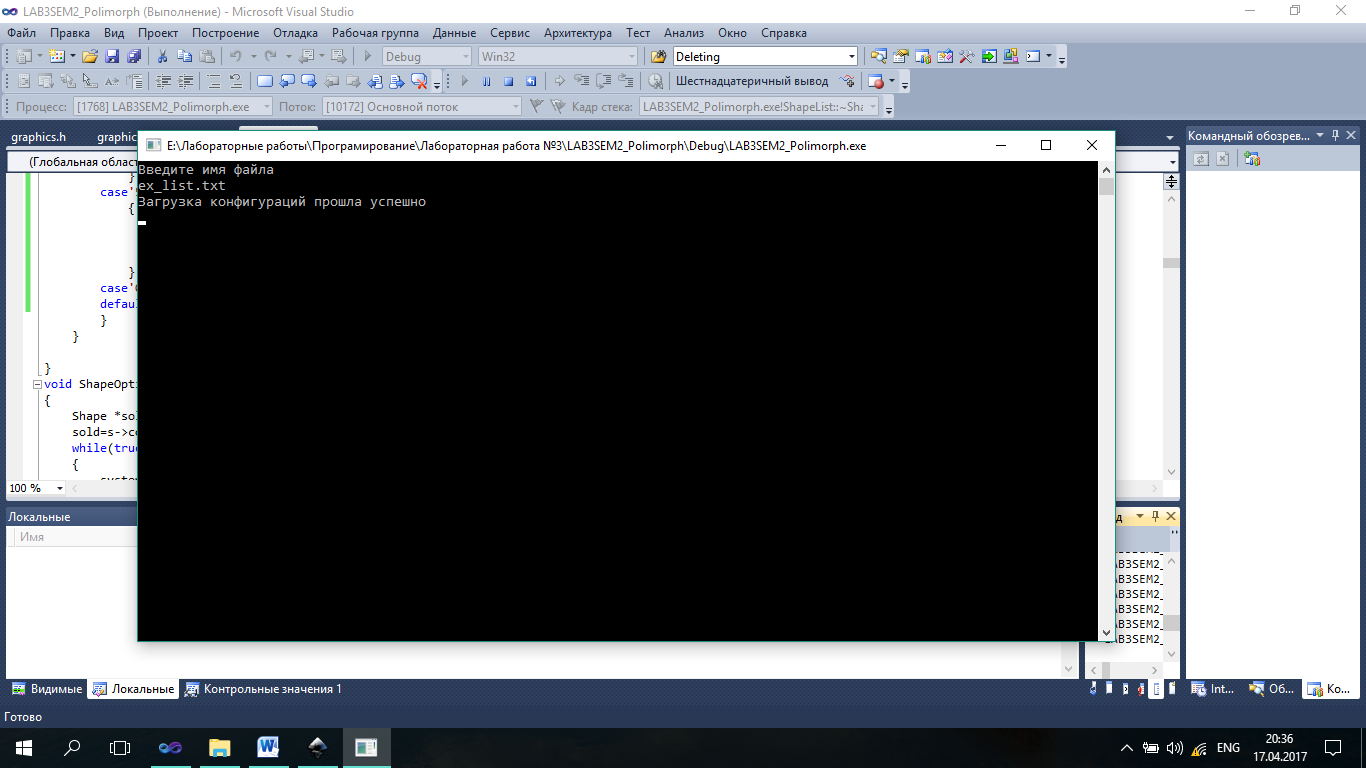
};

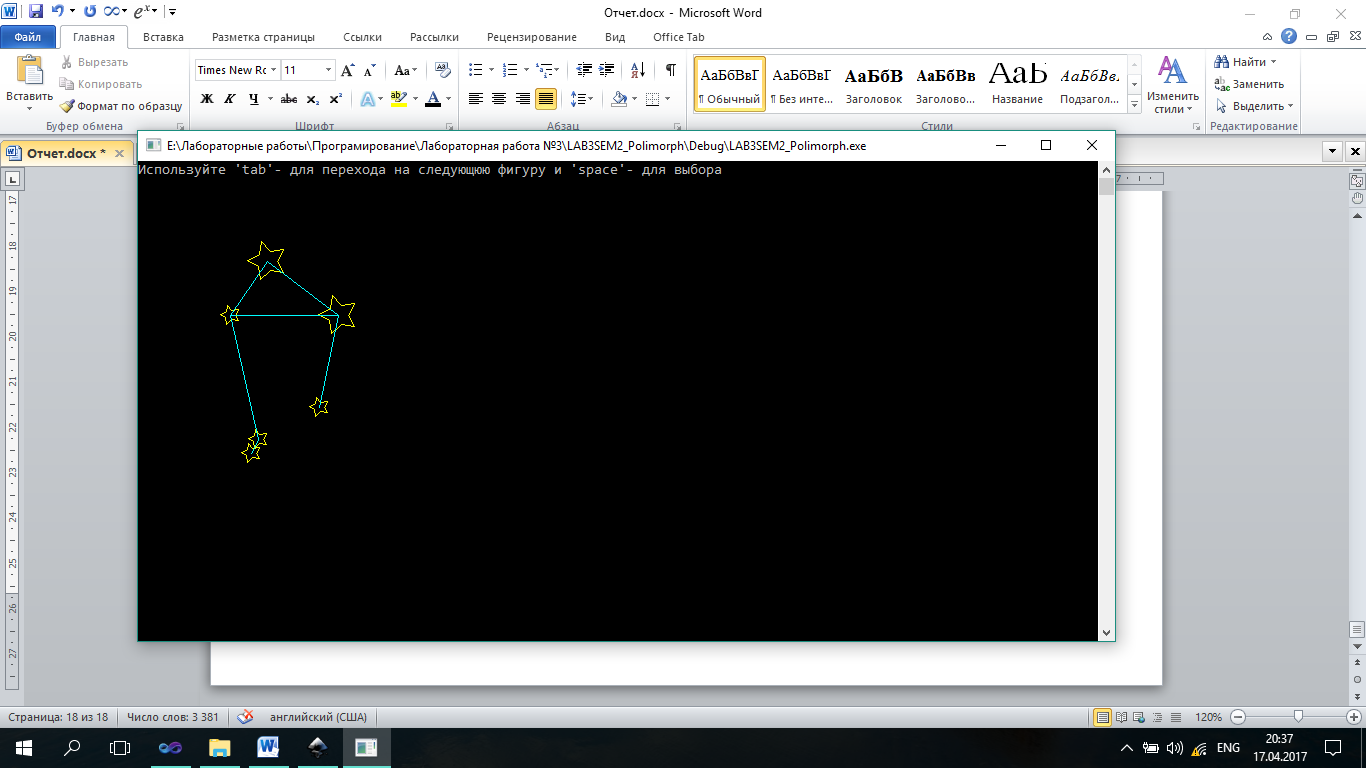
# 4. Опис інтерфейсу

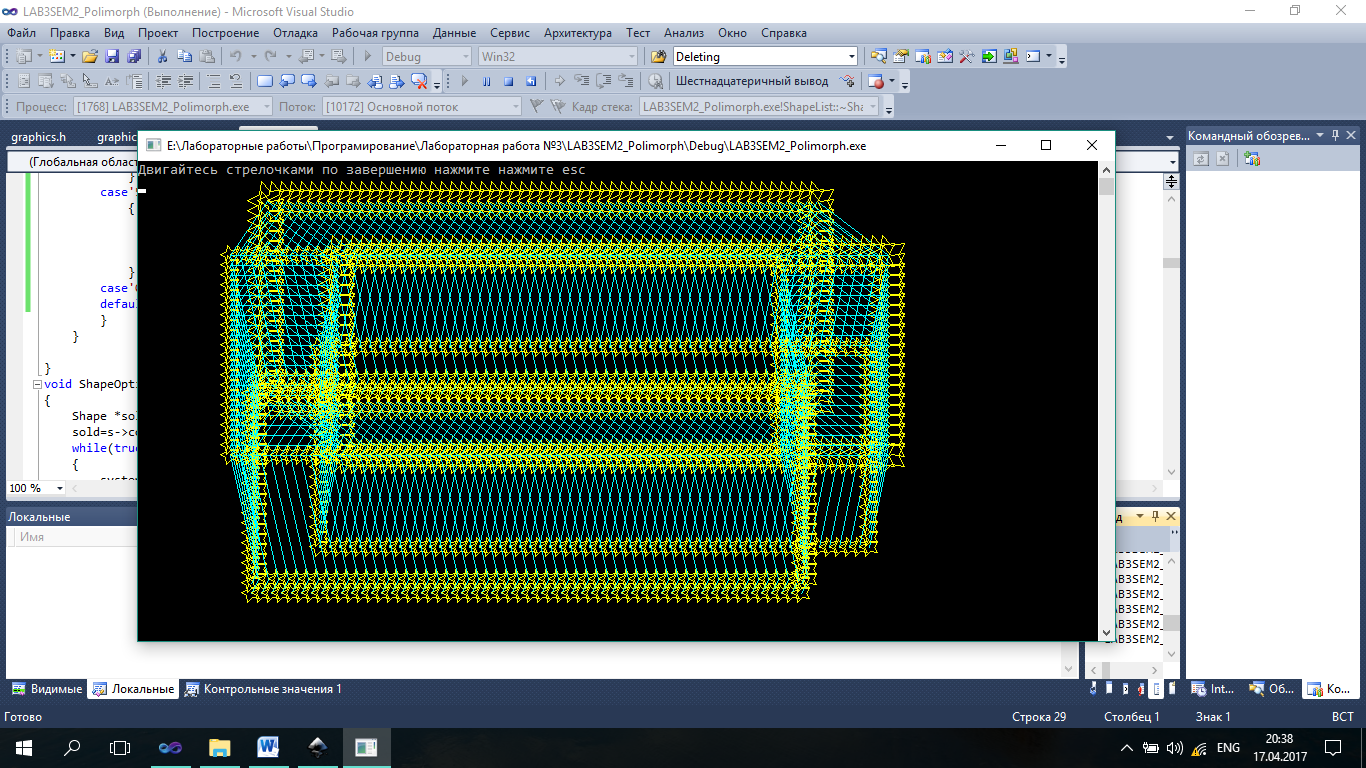
Програма дотримується всіх принципів дружнього інтерфейсу. Користувач працює з фігурами через меню, фігури візуалізуються, за принципом – «дія – відображення». Для розвантаження екрану використовується системне очищення консолі. Після виконання операцій у меню фігури користувач повретається до меню роботи зі списокм і працює в ньому доти доки не забажає вийти.

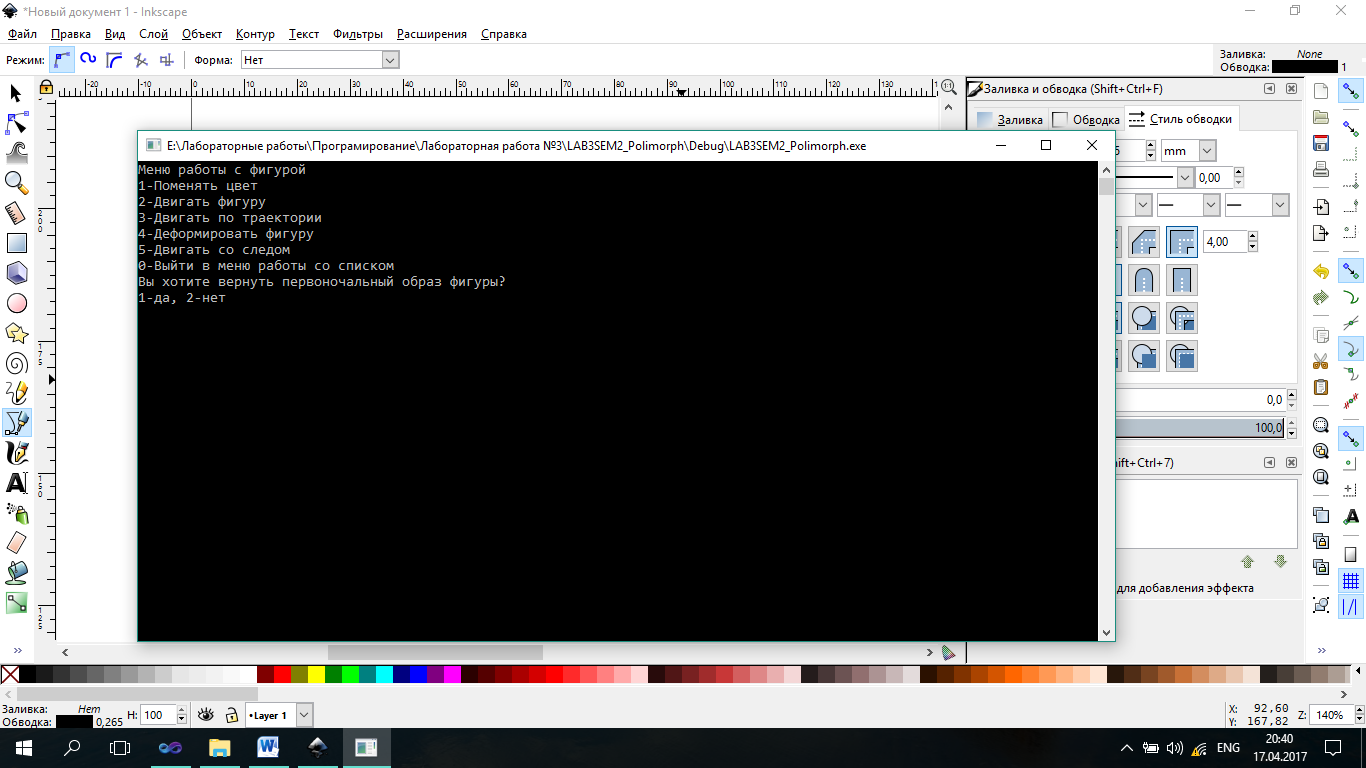
# 5. Описання тестових прикладів

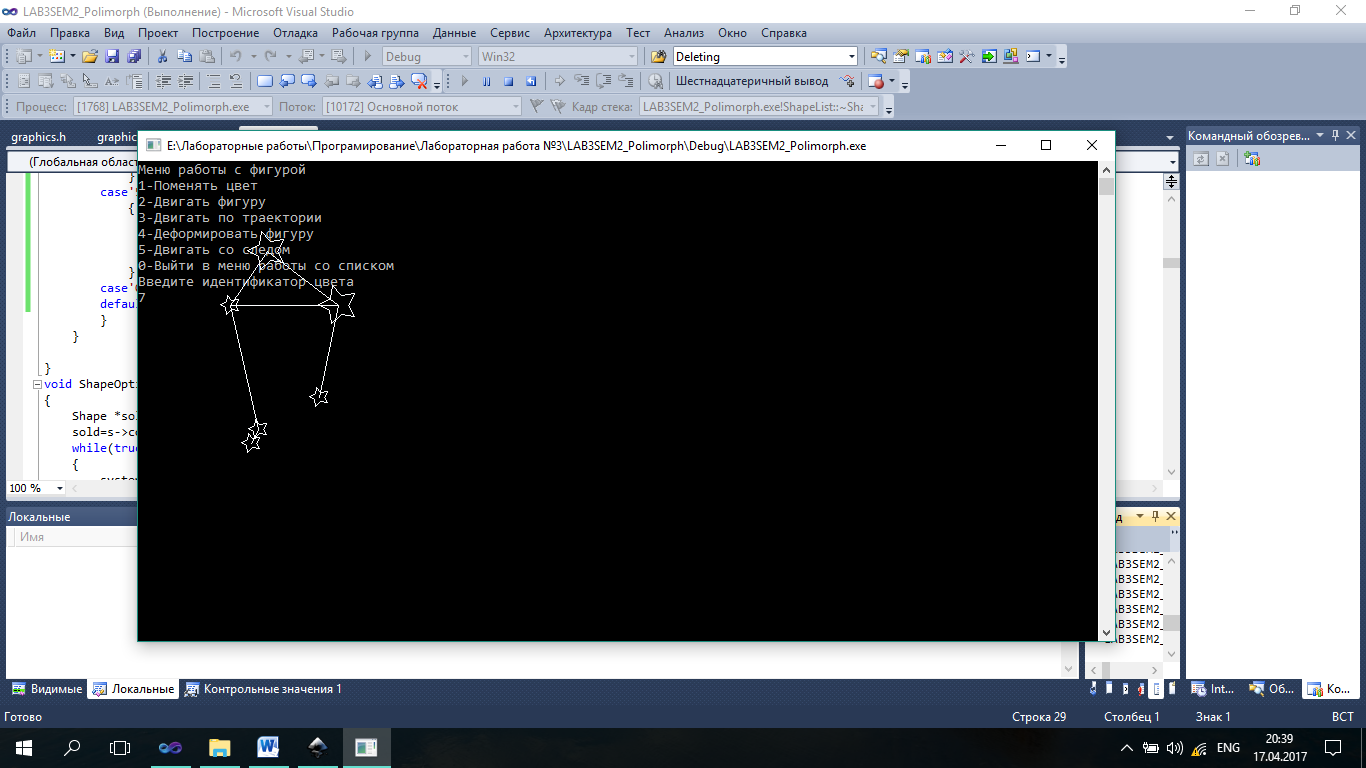












# 6.Аналіз помилок

Зауваження 1 Щоб не було великого нагромадження фігур тільки в режимі агрегації ми можемо побачити декілька фігур на екрані. В іншому одна фігура – одна функція.

Проблема 1 Стандартна функція стирання екрану system(“cls”) стирає і об’єкти також тому було вирішено не робити функцію для очищення від об’єктів. Крім того після заврешення роботи з об’єктом він зникає, що робить доцільним використання функції system(“cls”), хоча і у меню фігур фігури з якою ми працюємо не видно.

Проблема 2 Фігура зірка може генеруватися трішки неправильно при маленькому радіусі та високій кількості кутів в силу того, що генерація є процесом над дійсними числами (операції cos і sin частково визначені на множині цілих чисел тому можливі незначні похибка) тому за меньших елементів є більша похибка.

Проблема 3 Зірка видає помилку на створенні, якщо кількість кутів дорівнює нулю.(в силу того що неможливо поділити елементи на нуль).