Картина, която съдържа текст, графична колекция

Описанието е генерирано автоматичноТЕХНИЧЕСКИ УНИВЕРСИТЕТ – ВАРНА

Факултет по изчислителна техника и автоматизация

Катедра „Компютърни Науки и Технологии“

**СЕМЕСТРИАЛЕН ПРОЕКТ**

по дисциплината „Обектно Ориентирано Програмиране – 1 част”

на тема: „Програма за обработка на база данни от геометрични фигури. ”

Вариант 15

|  |  |
| --- | --- |
| Изготвил: Беркант Бейсимов | Проверил: проф. М. Карова |
| Специалност: КСТ |  |
| Група: 3а |  |
| Факултетен номер: 23621455 |  |

# Съдържание

I. Задание на проекта

II. Анализ на решението

1. Константи, Изброими типове и Структури за данните в програмата

2. Реализация на основното меню

3. Работа с фигури

4. Работа с база данни

5. Визуализация на фигури

III. Примерно действие на програмата

IV. ПРИЛОЖЕНИЕ 1 – Изходен код

# I. Задание на проекта

**Програма за обработка на база данни от геометрични фигури.**

Да се напише обектно ориентирана програма, която предоставя интерфейс за

работа с база данни записана в двоичен файл и съдържаща списък от геометрични фигури.

Програмата трябва да предлага на потребителя следните възможности:

1. Създаване на празна база данни.

2. Добавяне на фигура към базата.

3. Намиране на фигура по име.

4. Справка за общото лице на всички фигури от даден тип.

5. Справка за общия периметър на всички фигури с лице по – голямо зададено от потребителя число.

6. Визуализация на фигурите(Обяснено по долу).

7. Премахване на фигура.

Всяка фигура трябва да се характеризира с име, тип и специфични за типа параметри. Трябва да се подържат следните типове:

1. Триъгълник зададен с координатите на трите си върха.

2. Правоъгълник със страни успоредни на координатните оси зададен с координатите на горния си десен ъгъл и ширината и височината си.

3. Окръжност зададена координатите на центъра и дължината на радиуса.

4. Многоъгълник зададен с координатите на върховете си по посока на часовниковата стрелка.

Формата на базата данни е без значение стига да съхранява информацията в двоичен файл.

Визуализацията на фигурите да се извършва посредством използване на външна библиотека.

Библиотеката разполага със следните методи:

1. void Start(). Инициализира библиотеката Това е първата функция която трябва да се извика преди всички останали.

2. void Stop(). Де инициализира библиотеката. Това е функцията която трябва да се извика след приключване на работа с библиотеката.

3. void Clear(). Премахва всички визуализирани фигури(изчиства екрана).

4. void AddPolygon(int n, …). Добавя многоъгълник с N върха всеки един описан с координати представляващи двойка числа от тип float първо X, а второто Y.

5. void AddElipse(int x, int y, int rx, int ry). Добавя елипса с x, y координати на центъра и радиус rx, ry ориентирани по посока на координатните оси.

Документиране на курсовия проект: .doc файл със заглавна страница, условие на задачата, кратко описание на класовете и функциите и листинг на програмата с коментари.

# II. Анализ на решението

## 1. Константи, изброими типове и структури

Основният клас `Geometric\_Figure` съдържа общите характеристики на геометричните фигури – име, лице, периметър и тип.  
Производни класове са: Triangle, Rectangle, Circle и Polygon. Всеки от тях разширява функционалността чрез специфични атрибути и методи за визуализация.

## 2. Реализация на основното меню

Основното меню предоставя избор между няколко ключови действия: управление на база данни, добавяне на фигури, търсене, визуализация, изчисление на обща площ и периметър, изтриване на фигури и изход от програмата.  
Менюто се реализира чрез класа Menu и се управлява от функцията main().

## 3. Работа с фигури

Потребителят може да добавя следните типове фигури: триъгълник, правоъгълник, многоъгълник и окръжност. За всяка фигура се събират съответните параметри, изчисляват се площта и периметърът, и се създава обект от съответния клас.  
Класовете наследяват базовия клас Geometric\_Figure.

## 4. Работа с база данни

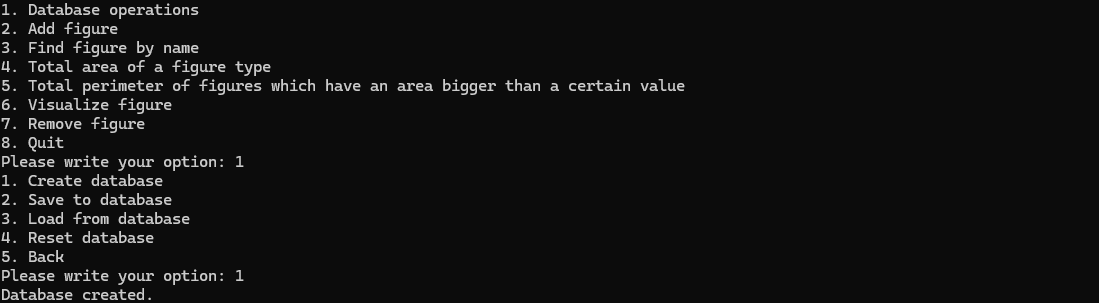
Класът Database позволява:  
- Създаване и инициализация на база от фигури  
- Добавяне и премахване на фигури  
- Запис и зареждане на фигури от двоичен файл  
- Търсене на фигура по име  
- Изчисление на обща площ по тип и общ периметър над зададена площ

## 5. Визуализация на фигури

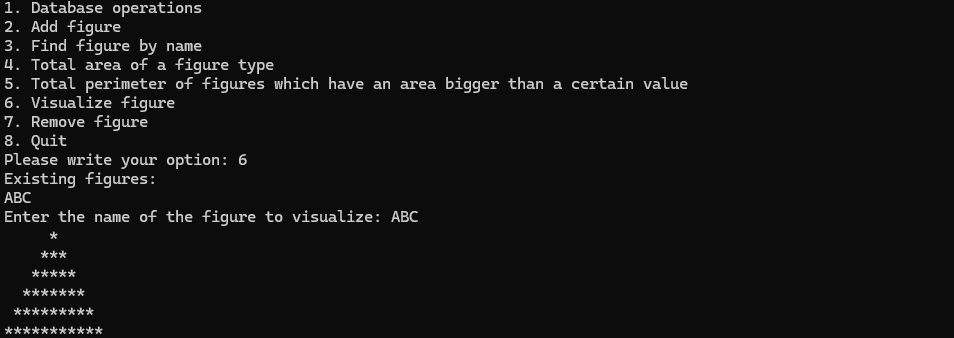
Визуализацията на всяка фигура се реализира чрез ASCII графика. Всеки производен клас има метод `visualizeXXX()` за изобразяване на фигурата на екрана според характеристиките ѝ. Фигурите се изобразяват с ‘\*’

# III. Примерно действие на програмата

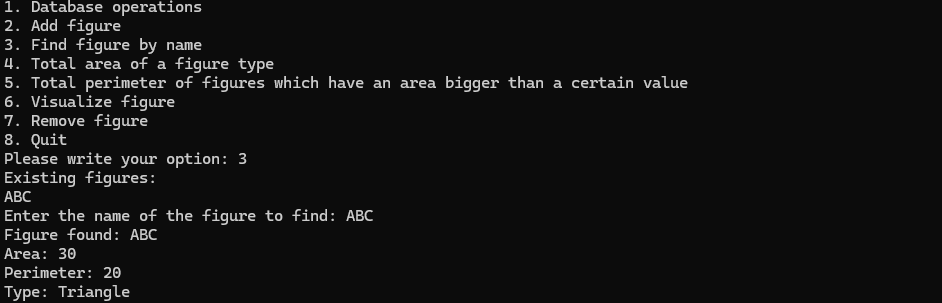
След стартиране на програмата се показва главно меню. Потребителят може да избере създаване на база, добавяне на фигура и извеждане на резултати. Следва примерен сценарий:

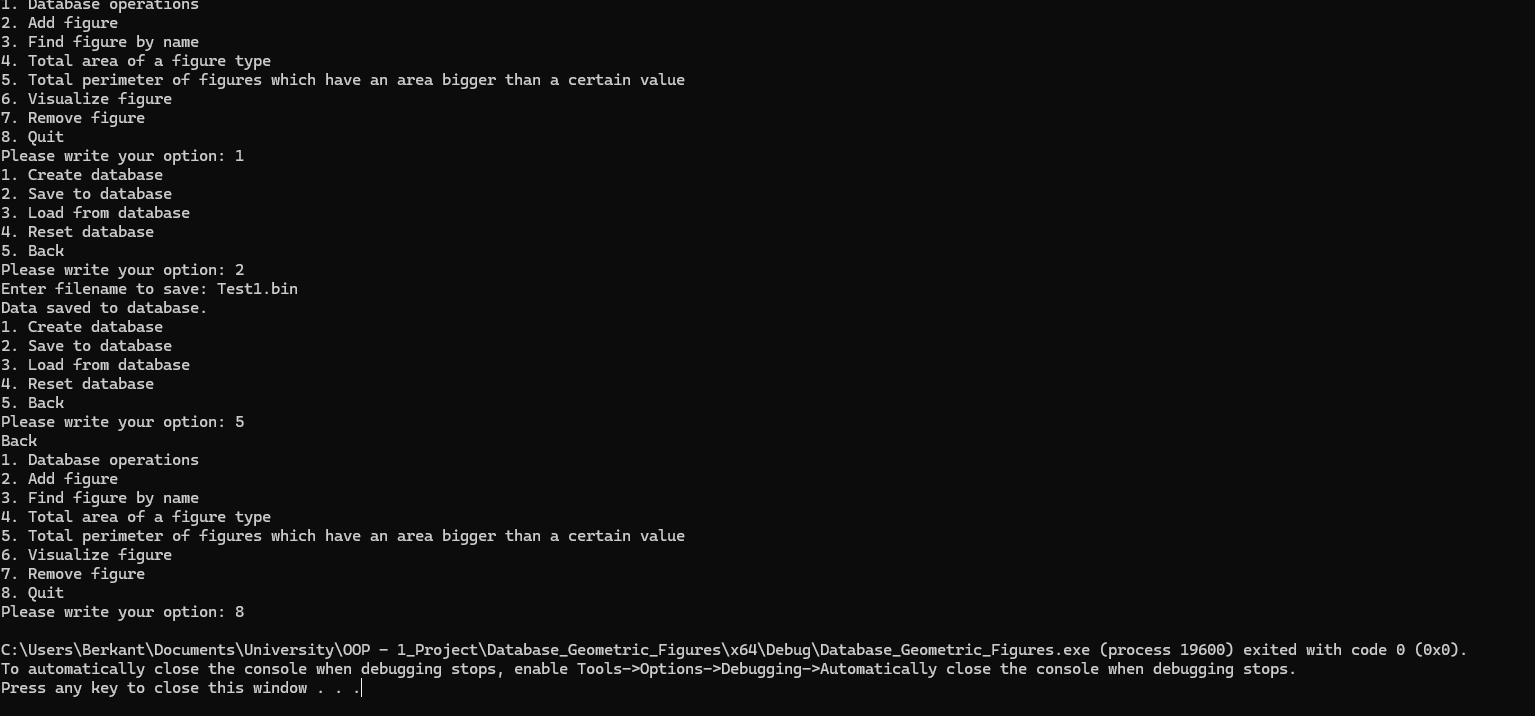
1. Създаване на база данни
2. A black screen with white text

   AI-generated content may be incorrect.Добавяне на окръжност с име ABC с височина 6, основа 10, страна А – 5, страна Б – 5
3. Визуализация на ABC -> Програмата отпечатва триъгълник с помощта на символи '\*'



1. Търсене на фигура по име (ABC) и извеждане на лице и периметър



1. Запазване на данните във двоичен файл Test1.bin и спиране на програмата

# IV. ПРИЛОЖЕНИЕ 1 – Изходен код

Кодът е организиран в следните файлове:  
- Main.cpp  
- Menu.{h,cpp}  
- Geometric\_Figure.{h,cpp}  
- Circle.{h,cpp}  
- Rectangle.{h,cpp}  
- Triangle.{h,cpp}  
- Polygon.{h,cpp}  
- Database.{h,cpp}

Пример: Main.cpp

#include <iostream>

#include <vector>

#include "Geometric\_Figure.h"

#include "Triangle.h"

#include "Rectangle.h"

#include "Circle.h"

#include "Polygon.h"

#include "Menu.h"

#include "Database.h"

using namespace std;

int main()

{

int userChoice;

Database\* db{};

Menu menu;

do

{

menu.DisplayMenu();

userChoice = menu.getUserChoice();

switch (userChoice)

{

case 1:

{

int dbUserChoice;

Menu dbMenu;

do

{

dbMenu.DisplayDatabaseMenu();

dbUserChoice = dbMenu.getUserChoice();

switch (dbUserChoice)

{

case 1:

if (db == NULL)

{

db = new Database();

cout << "Database created." << endl;

}

else

{

cout << "Error: Database already exists." << endl;

}

break;

case 2:

if (db != NULL)

{

string filename;

cout << "Enter filename to save: ";

cin >> filename;

db->saveToFile(filename);

cout << "Data saved to database." << endl;

}

else

{

cout << "Error: No database exists. Please create a database first." << endl;

}

break;

case 3:

if (db != NULL)

{

string filename;

cout << "Enter filename to load: ";

cin >> filename;

db->loadFromFile(filename);

cout << "Data loaded from database." << endl;

}

else

{

cout << "Error: No database exists. Please create a database first." << endl;

}

break;

case 4:

if (db != NULL)

{

delete db;

db = NULL;

cout << "Database reset." << endl;

}

else

{

cout << "Error: No database exists. Please create a database first." << endl;

}

break;

case 5:

{

cout << "Back" << endl;

break;

}

default:

cout << "Invalid option!" << endl;

break;

}

} while (dbUserChoice != 5);

break;

}

case 2:

{

if (db == NULL)

{

cout << "Error: No database exists. Please create a database first." << endl;

break;

}

int subUserChoice;

Menu subMenu;

do

{

subMenu.DisplaySubMenu();

subUserChoice = subMenu.getUserChoice();

switch (subUserChoice)

{

case 1:

{

string name;

double height, base, a, b, area, perim;

cout << "Write name: ";

cin >> name;

cout << "Write height: ";

cin >> height;

cout << "Write base: ";

cin >> base;

cout << "Write side A: ";

cin >> a;

cout << "Write side B: ";

cin >> b;

area = (base \* height) / 2;

perim = a + b + base;

Triangle\* tri = new Triangle(name, area, perim, height, base, a, b);

db->addFigure(tri);

cout << "Triangle added to database." << endl;

break;

}

case 2:

{

string name;

double height, width, area, perim;

cout << "Write name: ";

cin >> name;

cout << "Write height: ";

cin >> height;

cout << "Write width: ";

cin >> width;

area = height \* width;

perim = 2 \* (height + width);

Rectangle\* rec = new Rectangle(name, area, perim, height, width);

db->addFigure(rec);

cout << "Rectangle added to database." << endl;

break;

}

case 3:

{

string name;

int sides;

double side\_length, area, perim;

cout << "Write name: ";

cin >> name;

cout << "Write number of sides: ";

cin >> sides;

cout << "Write side length: ";

cin >> side\_length;

area = (sides \* side\_length \* side\_length) / (4 \* tan(3.14 / sides));

perim = sides \* side\_length;

Polygon\* poly = new Polygon(name, area, perim, sides, side\_length);

db->addFigure(poly);

cout << "Polygon added to database." << endl;

break;

}

case 4:

{

string name;

double radius, area, perim;

cout << "Write name: ";

cin >> name;

cout << "Write radius: ";

cin >> radius;

area = 3.14 \* radius \* radius;

perim = 2 \* 3.14 \* radius;

Circle\* cir = new Circle(name, area, perim, radius);

db->addFigure(cir);

cout << "Circle added to database." << endl;

break;

}

case 5:

{

cout << "Back" << endl;

break;

}

default:

cout << "Invalid option!" << endl;

break;

}

} while (subUserChoice != 5);

break;

}

case 3:

{

if (db == NULL)

{

cout << "Error: No database exists. Please create a database first." << endl;

break;

}

db->printExistingFigures();

string name;

cout << "Enter the name of the figure to find: ";

cin >> name;

db->findFigure(name);

break;

}

case 4:

{

if (db == NULL)

{

cout << "Error: No database exists. Please create a database first." << endl;

break;

}

string type;

cout << "Enter the type of figure (Triangle, Rectangle, Polygon, Circle): ";

cin >> type;

db->getTotalArea(type);

break;

}

case 5:

{

if (db == NULL)

{

cout << "Error: No database exists. Please create a database first." << endl;

break;

}

double area;

cout << "Enter the area threshold: ";

cin >> area;

db->getTotalPerimeter(area);

break;

}

case 6:

{

if (db == NULL)

{

cout << "Error: No database exists. Please create a database first." << endl;

break;

}

db->printExistingFigures();

string name;

cout << "Enter the name of the figure to visualize: ";

cin >> name;

Geometric\_Figure\* figure = db->findFigureName(name);

if (figure != NULL)

{

if (figure->getType() == "Triangle")

{

Triangle\* tri = dynamic\_cast<Triangle\*>(figure);

tri->visualizeTri();

}

else if (figure->getType() == "Rectangle")

{

Rectangle\* rec = dynamic\_cast<Rectangle\*>(figure);

rec->visualizeRec();

}

else if (figure->getType() == "Polygon")

{

Polygon\* poly = dynamic\_cast<Polygon\*>(figure);

poly->visualizePoly();

}

else if (figure->getType() == "Circle")

{

Circle\* cir = dynamic\_cast<Circle\*>(figure);

cir->visualizeCir();

}

}

break;

}

case 7:

{

if (db == NULL)

{

cout << "Error: No database exists. Please create a database first." << endl;

break;

}

db->printExistingFigures();

string name;

cout << "Enter the name of the figure to remove: ";

cin >> name;

db->removeFigure(name);

cout << "Figure removed from database." << endl;

break;

}

case 8:

{

break;

}

default:

cout << "Invalid option!" << endl;

break;

}

} while (userChoice != 8);

return 0;

}