Московский авиационный институт
(Национальный исследовательский университет)

Факультет: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»
Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа № 8

Тема: Асинхронное программирование

Студент: Павлова К.А.
Группа: М8О-306Б-18
Преподаватель:
Дата:
Оценка:

Москва, 2021

1. Постановка задачи

Вариант 16:

16.	8-угольник	Треугольник	Квадрат

2. Описание программы

При запуске программы мы указываем размер буфера фигур. В самой программе мы добавляем фигуры в буфер. При полном заполнении буфер автоматически экспортируется. При экспорте мы создаём отдельный поток, в котором экспортируем данные о фигурах на экран и в файл. После экспорта буфер очищается.

3. Набор тестов

№	Описание	Ввод
1	Демонстрация работы буфера и автоматического экспорта	.\oop_exercise_03.exe 4 2 0 0 1 0 y 3 100 100 20 1 y 1 0 0 2 2 y 2 -100 -100 100 0.75 n

4. Результаты выполнения тестов.

```
.\oop_exercise_03.exe 4
Choose type:
1 - Octagon
2 - Square
3 - Triangle
Type: 2
Coordinates of center: 0 0
Radius: 1
Angle: 0

Continue? (y/n)? y

Choose type:
1 - Octagon
2 - Square
3 - Triangle
Type: 3
Coordinates of center: 100 100
```

```
Radius: 20
Angle: 1

Continue? (y/n)? y

Choose type:
1 - Octagon
2 - Square
3 - Triangle
Type: 1
Coordinates of center: 0 0
Radius: 2
Angle: 2

Continue? (y/n)? y

Choose type:
1 - Octagon
2 - Square
3 - Triangle
Type: 2
Coordinates of center: -100 -100
Radius: 100
Angle: 0.75

Buffer filled. Exporting data:

Points: (1, 0), (-4.37114e-08, 1), (-1, -8.74228e-08), (1.19249e-08, -1)
Center: (-1.19199e-08, -1.49012e-08)
Size: 2

Points: (110.806, 116.829), (80.0223, 100.944), (109.172, 82.227)
Center: (100, 100)
Size: 519.615

Points: (-0.832294, 1.81859), (-1.87446, 0.69742), (-1.81859, -0.832294), (-0.69742, -1.87446), (0.832294, -
1.81859), (1.87446, -0.69742), (1.81859, 0.832294), (0.69742, 1.87446)
Center: (7.45058e-08, 7.45058e-08)
Size: 11.3137

Points: (-26.8311, -31.8361), (-168.164, -26.8311), (-173.169, -168.164), (-31.8361, -173.169)
Center: (-100, -100)
Size: 20000

Data exported
Continue? (y/n)? n
```

5. Листинг программы

```
#include <iostream>
#include <vector>
#include <cmath>
#include <fstream>
#include <string>
#include <thread>

#define PI 3.14159265f

//точка многоугольника
using point = std::pair<float, float>;

//родительский класс для всех фигур
class Figure
{
public:
      //вычисление геометрического центра фигуры
```

```
	point getCenter()
	{
		point center = std::make_pair(0.0f, 0.0f);
		for (point p : m_points)
		{
			center.first += p.first;
			center.second += p.second;
		}
		center.first /= m_points.size();
		center.second /= m_points.size();
		return center;
	}

	//вывод координат вершин фигуры
	void print(std::ostream& output)
	{
		bool comma = false; //печатать запятую перед точкой или нет
		for (point p : m_points)
		{
			if (comma) output << ", ";
			comma = true;

			output << "(" << p.first << ", " << p.second << ")";
		}
	}

	//вычисление площади
	float size()
	{
		float S = 0.0f;
		for (int i = 0; i < m_points.size() - 1; i++)
			S += triag(m_points[0], m_points[i], m_points[i + 1]);
		return S;
	}

	//запись в поток
	void write(std::ostream& output)
	{
		output << "Points: ";
		print(output);

		point center = getCenter();
		output << std::endl << "Center: (" << center.first << ", " << center.second << ")";
		output << std::endl << "Size: " << size() << std::endl << std::endl;
	}

protected:
	//точки многоугольника
	std::vector<point> m_points;

	//площадь треугольника по координатам вершин
	//S = 1/2 * abs(det(x1 - x3, y1 - y3; x2 - x3, y2 - y3))
	float triag(point& a, point& b, point& c)
	{
		return 0.5f * abs((a.first - c.first) * (b.second - c.second) -
			(b.first - c.first) * (a.second - c.second));
	}
};

//Любую фигуру вращения можно задать координатами центра, радиусом описанной окружности и углом
	поворота
//8-угольник
class Octagon : public Figure
{
public:
	//заполняем вектор вершин
	Octagon(float x, float y, float r, float a)
```

```
	{
		for (int i = 0; i < 8; i++)
		{
			float phi = a + i * PI / 4.0f;
			m_points.push_back(std::make_pair(r * cosf(phi) + x, r * sinf(phi) + y));
		}
	}
};
//Квадрат
class Square : public Figure
{
public:
	//заполняем вектор вершин
	Square(float x, float y, float r, float a)
	{
		for (int i = 0; i < 4; i++)
		{
			float phi = a + i * PI / 2.0f;
			m_points.push_back(std::make_pair(r * cosf(phi) + x, r * sinf(phi) + y));
		}
	}
};
//Треугольник
class Triangle : public Figure
{
public:
	//заполняем вектор вершин
	Triangle(float x, float y, float r, float a)
	{
		for (int i = 0; i < 3; i++)
		{
			float phi = a + i * PI / 1.5f;
			m_points.push_back(std::make_pair(r * cosf(phi) + x, r * sinf(phi) + y));
		}
	}
};
//генератор фигур
class Factory
{
public:
	enum class Type : int
	{
		T_None,
		T_Octagon,
		T_Square,
		T_Triangle
	};

	static std::shared_ptr<Figure> Make(Type type, float x, float y, float r, float a)
	{
		switch (type)
		{
		case Factory::Type::T_Octagon:
			return std::make_shared<Figure>(Octagon(x, y, r, a));

		case Factory::Type::T_Square:
			return std::make_shared<Figure>(Square(x, y, r, a));

		case Factory::Type::T_Triangle:
			return std::make_shared<Figure>(Triangle(x, y, r, a));
		}
		return std::shared_ptr<Figure>(nullptr);
	}
};

//publisher-subscriber экспортер
class Exporter
```

```
{
public:
	Exporter(std::vector<std::shared_ptr<Figure>> &figs)
	{
		//генерируем уникальное имя файла
		int nameI = 0;
		std::string name = "0.txt";
		while (true)
		{
			std::ifstream test(name);
			if (test.good())
			{
				nameI++;
				name = std::to_string(nameI) + ".txt";
			}
			else break;
		}

		//создаём подписчиков
		WriterScreen wscreen;
		WriterFile wfile(name);

		//создаём поток, в котором отправляем информацию подписчикам
		std::thread write_thread([&wscreen, &wfile, &figs]()
			{
				for (auto& fig : figs)
				{
					wscreen.Write(fig);
					wfile.Write(fig);
				}
			});

		//ждём завершения потока
		write_thread.join();
	}

private:
	//родительский класс подписчиков
	class Writer
	{
	public:
		Writer(std::ostream& o) : m_output(o) {}

		//запись в поток
		void Write(std::shared_ptr<Figure> fig)
		{
			fig->write(m_output);
		}

	protected:
		std::ostream& m_output;
	};
	//экспортирующий на экран
	class WriterScreen : public Writer
	{
	public:
		WriterScreen() : Writer(std::cout) {}
	};
	//экспортирующий в файл
	class WriterFile : public Writer
	{
	public:
		WriterFile(std::string fname) : file(fname), Writer(file) {}

		~WriterFile()
		{
			file.close();
```

```
		}

	private:
		std::ofstream file;
	};
};

//документ
class Buffer
{
public:
	Buffer(int maxSz) : m_maxSz(maxSz)
	{
		m_figures.reserve(maxSz);
	}
	~Buffer() {}

	//добавление фигур
	void AddFigure(std::shared_ptr<Figure>&fig)
	{
		m_figures.push_back(fig);

		//экспорт
		if (m_figures.size() == m_maxSz)
		{
			//экспортируем фигуры
			std::cout << "Buffer filled. Exporting data:" << std::endl << std::endl;
			Exporter exporter(m_figures);
			std::cout << "Data exported" << std::endl;

			//очищаем буффер
			m_figures.clear();
		}
	}

private:
	std::vector<std::shared_ptr<Figure>> m_figures;
	int m_maxSz;
};

int main(int argc, char* argv[])
{
	if (argc != 2)
	{
		std::cout << "Usage: oop_exercise_08 buffer_size" << std::endl;
		return 0;
	}

	int buf_sz = std::atoi(argv[1]);
	if (buf_sz <= 0)
	{
		std::cout << "Buffer size must be positive" << std::endl;
		return 0;
	}

	//буффер
	Buffer buffer(buf_sz);

	//цикл программы
	while (true)
	{
		std::cout << "Choose type:" << std::endl <<
			"1 - Octagon" << std::endl <<
			"2 - Square" << std::endl <<
			"3 - Triangle" << std::endl <<
			"Type: ";
```

```
            int type;
            std::cin >> type;

            Factory::Type ftype = Factory::Type::T_None;
            switch (type)
            {
            case 1:
                    ftype = Factory::Type::T_Octagon;
                    break;
            case 2:
                    ftype = Factory::Type::T_Square;
                    break;
            case 3:
                    ftype = Factory::Type::T_Triangle;
                    break;

            default:
                    std::cout << "Unknown type" << std::endl;
                    break;
            }
            if (ftype == Factory::Type::T_None) break;

            float x, y, r, a;
            std::cout << "Coordinates of center: ";
            std::cin >> x >> y;

            std::cout << "Radius: ";
            std::cin >> r;

            std::cout << "Angle: ";
            std::cin >> a;

            std::cout << std::endl;
            buffer.AddFigure(Factory::Make(ftype, x, y, r, a));

            //выход
            std::string yes;

            std::cout << "Continue? (y/n)? ";
            std::cin >> yes;

            if (yes != "y" && yes != "Y") break;
            std::cout << std::endl;
      }
    return 0;
}
```

6. Выводы:

Изучено асинхронное программирование. Получены практические навыки в параллельной обработке данных. Также получены практические навыки в синхронизации потоков.

СПИСОК ЛИТЕРАТУРЫ

1. std::thread [электронный ресурс]. URL: https://ru.cppreference.com/w/cpp/thread/thread
2. Потоки, блокировки и условные переменные в C++11 [электронный ресурс]. URL: https://habr.com/ru/post/182610/