Московский авиационный институт
(Национальный исследовательский университет)

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

Лабораторная работа № 8

Тема: Асинхронное программирование

Студент: Попов Илья Павлович
Группа: 80-206
Преподаватель: Чернышов Л.Н.
Дата: 16.12
Оценка:

Москва, 2021

1. Постановка задачи

Создать приложение, которое будет считывать из стандартного ввода данные фигур, согласно варианту задания, выводить их характеристики на экран и записывать в файл.

Программа должна:

1. Осуществлять ввод из стандартного ввода данных фигур, согласно варианту задания;
2. Программа должна создавать классы, соответствующие введенным данных фигур;
3. Программа должна содержать внутренний буфер, в который помещаются фигуры. Для создания буфера допускается использовать стандартные контейнеры STL. Размер буфера задается параметром командной строки. Например, для буфера размером 10 фигур: **oop_exercise_08 10**;
4. При накоплении буфера они должны запускаться на асинхронную обработку, после чего буфер должен очищаться;
5. Обработка должна производиться в отдельном потоке;
6. Реализовать два обработчика, которые должны обрабатывать данные буфера:

a. Вывод информации о фигурах в буфере на экран;

b. Вывод информации о фигурах в буфере в файл. Для каждого буфера должен создаваться файл с уникальным именем.

7. Оба обработчика должны обрабатывать каждый введенный буфер. Т.е. после каждого заполнения буфера его содержимое должно выводиться как на экран, так и в файл.
8. Обработчики должны быть реализованы в виде лямбда-функций и должны хранится в специальном массиве обработчиков. Откуда и должны последовательно вызываться в потоке – обработчике;
9. В программе должно быть ровно два потока (thread). Один основной (main) и второй для обработчиков;
10. В программе должен явно прослеживаться шаблон Publish-Subscribe. Каждый обработчик должен быть реализован как отдельный подписчик;
11. Реализовать в основном потоке (main) ожидание обработки буфера в потоке-обработчике. Т.е. после отправки буфера на обработку основной поток должен ждать, пока поток обработчик выведет данные на экран и запишет в файл.

Вариант № 24
Фигуры: квадрат, треугольник, 8-угольник

2. Описание программы

Программа считывает из стандартного ввода данные фигур, согласно варианту задания, выводит их характеристики на экран и записывать в файл.

Программа состоит из 14 файлов:

1. figures.h - содержит реализацию родительского класса фигур;
   square.cpp и square.h — реализация класса квадрата;
   triangle.cpp и triangle.h — реализация класса треугольника;
   octagon.cpp и octagon.h — реализация класса 8-угольника;
2. factory.cpp и factory.h — класс для создания графических примитивов фигур ;
3. subscriber.cpp и subscriber.h — реализация класса, необходимого для передачи в поток ;как функтора, который необходим для выполнения обработки на отдельном потоке;
4. processor.cpp и processon.h — реализация класса, необходимого для вывода фигур на экран и в файл;
5. main.cpp - файл с взаимодействием с пользователем.

Пользователь при запуске указывает размер буфера фигур. В программе пользователь может полностью заполнить буфер различными фигурами, после чего будет показано содержимое буфера на экран и экспорт буфера в файл с уникальным именем.

После экспорта буфер очищается и пользователь может вновь его заполнить различными фигурами.

В программе имеются два потока, в поток sub_thread передаем функтор класса Subscriber, который после заполнения буфера будет выводить информацию о фигурах из буфера в файл и на экран.

При неверном вводе параметров фигуры будет происходить исключения и пользователю нужно снова выбрать фигуру с данными.

3. Набор тестов

Тест №1 (демонстрирует работу программы при корректно введенных входных данных)

```
_____________________________________
Available commands:
add <square, triangle, octagon>
menu - print this menu
exit

_____________________________________
add triangle
a = 1
h = 2

Figure has been successfuly added!

_____________________________________
add square
side = 3

Figure has been successfuly added!

_____________________________________
add octagon
```

side = 4

Figure has been successfuly added!

Buffer is full

Triangle
a = 1
h = 2
Area = 1

Square
Side = 3
Area = 9

Octagon
Side = 4
Area = 77.2548
//
В данный момент произошло заполнение буфера и данные введенных фигур выводятся на экран и выгружаются в файл с уникальным именем.

o.txt:

Triangle
a = 1
h = 2
Area = 1

Square
Side = 3
Area = 9

Octagon
Side = 4
Area = 77.2548

После этого буфер очищается и программа снова готова для ввода
//
menu

Available commands:
add <square, triangle, octagon>
menu - print this menu
exit

---

add octagon
side = 5

Figure has been successfuly added!

---

add octagon
side = 5

Figure has been successfuly added!

---

add octagon
side = 5

Figure has been successfuly added!

```
_____________________________________
Buffer is full

Octagon
Side = 5
Area = 120.711

Octagon
Side = 5
Area = 120.711

Octagon
Side = 5
Area = 120.711
//////////////////////////////////////////////////////////////////
Буфер снова заполнен, данные распечатаны и выгружены

1.txt:
Octagon
Side = 5
Area = 120.711

Octagon
Side = 5
Area = 120.711

Octagon
Side = 5
Area = 120.711

После этого буфер очищается и программа снова готова для ввода
//////////////////////////////////////////////////////////////////
exit
```

Тест №2 (демонстрирует работу программы при некорректно введенных входных данных)

```
_____________________________________
Available commands:
add <square, triangle, octagon>
menu - print this menu
exit
_____________________________________
adsgsdg
Wrong command
add square
side = dsag

Square input error!
Repeat input!
side = 3

Figure has been successfuly added!
_____________________________________
```

4. Результаты выполнения тестов

Представлены выше, с целью упростить прочтение.

5. Листинг программы

figures.h

```
#ifndef _D_FIGURE_H_
#define _D_FIGURE_H_

#include <iostream>
#include <cmath>
#include <string>

using namespace std;

struct figure {
   virtual ostream& print(ostream& os) const = 0;
   virtual double area() const = 0;
   virtual ~figure() {}

   virtual bool is_fig_num(const string& s) {
      bool point = false;
      for (int i = 0; i < s.length(); ++i) {
         if (s[i] == '.') {
            if ((i == 0 || i == s.length() - 1) || point) {
               return false;
            }
            else {
               point = true;
            }
         }
         else if (s[i] < '0' || s[i] > '9') { return false; }
      }
      return true;
   }
};

#endif
```

square.cpp

```
#include "square.h"

square::square(istream& is) {
   string str;
   cout << "side = ";  is >> str; cout << endl;

   while (true) {
      if (is_fig_num(str)) {
         side = stod(str);
         break;
      }
      cout << "Square input error!\n";
```

```
        cout << "Repeat input!\n";
        cout << "side = ";  is >> str; cout << endl;
    }
}

double square::area() const {
    long double res = pow(side, 2);
    return res;
}

ostream& square::print(ostream& os) const {
    os << endl << "Square" << endl;
    os << "Side = " << side << endl;
    os << "Area = " << area() << '\n';

    return os;
}
```

square.h

```
#ifndef _D_SQUARE_H_
#define _D_SQUARE_H_

#include "figure.h"

class square : public figure {
public:
    square() = default;
    square(istream& is);
    double area() const override;
    ostream& print(ostream&) const override;
private:
    double side;
};

#endif
```

triangle.cpp

```
#include "triangle.h"

triangle::triangle(istream &is) {
    string str1, str2;

    cout << "a = "; is >> str1;
    cout << "h = "; is >> str2; cout << endl;

    while(true) {
        if (is_fig_num(str1) && is_fig_num(str2)) {
            a = stod(str1);
            h = stod(str2);
            break;
        }
        cout << "Triangle input error!\n";
        cout << "Repeat input!\n";
```

```
        cout << "a = "; is >> str1;
        cout << "h = "; is >> str2; cout << endl;
    }
};

double triangle::area() const {
    double res = 0.5 * a * h;
    return res;
}

ostream& triangle::print(ostream& os) const {
    os << endl << "Triangle" << endl;
    os << "a = " << a << endl;
    os << "h = " << h << endl;
    os << "Area = " << area() << '\n';
    return os;
}
```

triangle.h

```
#ifndef _D_TRIANGLE_H_
#define _D_TRIANGLE_H_

#include "figure.h"

class triangle : public figure {
public:
    triangle() = default;
    triangle(istream& is);
    double area() const override;
    ostream& print(ostream& os) const override;
private:
    double a, h;
};

#endif
```

octagon.cpp

```
#include "octagon.h"

octagon::octagon(istream& is) {
    string str;
    cout << "side = ";  is >> str; cout << endl;

    while (true) {
        if (is_fig_num(str)) {
            side = stod(str);
            break;
        }
        cout << "Octagon input error!\n";
        cout << "Repeat input!\n";
        cout << "side = ";  is >> str; cout << endl;
    }
}
```

```
double octagon::area() const {
   long double res = (2 + 2 * sqrt(2)) * pow(side, 2);
   return res;
}

ostream& octagon::print(ostream& os) const {
   os << endl << "Octagon" << endl;
   os << "Side = " << side << endl;
   os << "Area = " << area() << '\n';

   return os;
}
```

octagon.h

```
#ifndef _D_octagon_H_
#define _D_octagon_H_

#include "figure.h"

class octagon : public figure {
public:
   octagon() = default;
   octagon(istream& is);
   double area() const override;
   ostream& print(ostream& os) const override;
private:
   double side;
};

#endif
```

factory.cpp

```
#include "factory.h"

shared_ptr<figure> factory::new_figure(istream &is) {
   string name;
   is >> name;
   if (name == "square") {
      return shared_ptr<figure> ( new square(is));
   } else if ( name == "triangle") {
      return shared_ptr<figure> ( new triangle(is));
   } else if ( name == "octagon") {
      return shared_ptr<figure> ( new octagon(is));
   } else {
      throw logic_error("Error figure type!\n\n");
   }
}
```

factory.h

```
#ifndef _D_FACTORY_H_
#define _D_FACTORY_H_
```

```
#include <memory>
#include <fstream>
#include <string>
#include "figure.h"

#include "square.h"
#include "octagon.h"
#include "triangle.h"

struct factory {
   shared_ptr<figure> new_figure(istream& is);
};

#endif
```

subscriber.cpp

```
#include "subscriber.h"

void subscriber::operator()() {
   while(!stop){
      unique_lock<mutex>lock(mtx);
      cond_var.wait(lock,[&]{ return (buffer != nullptr || stop);});

      for (auto elem: processors) {
         elem->process(buffer);
      }

      buffer = nullptr;
      cond_var.notify_all();
   }
}
```

subscriber.h

```
#ifndef _D_SUBSCTIBER_H_
#define _D_SUBSCTIBER_H_

#include "processor.h"

struct subscriber {
   void operator()();
   vector<shared_ptr<processor>> processors;
   shared_ptr<vector<shared_ptr<figure>>> buffer;
   mutex mtx;
   condition_variable cond_var;
   bool stop = false;
};

#endif
```

processor.cpp

```
#include "processor.h"

void stream_processor::process(shared_ptr<vector<shared_ptr<figure>>> buffer) {
    for (auto figure : *buffer) {
        figure->print(cout);
    }
}

void file_processor::process(shared_ptr<vector<shared_ptr<figure>>> buffer) {
    ofstream fout;
    fout.open(to_string(cnt) + ".txt");
    cnt++;
    if (!fout.is_open()) {
        cout << "can't open\n";
        return;
    }
    for (auto figure : *buffer) {
        figure->print(fout);
    }
}
```

processon.h

```
#ifndef _D_PROCESSOR_H_
#define _D_PROCESSOR_H_

#include <condition_variable>
#include <thread>
#include <vector>
#include <mutex>

#include "figure.h"
#include "factory.h"

struct processor {
    virtual void process(shared_ptr<vector<shared_ptr<figure>>> buffer) = 0;
};

struct stream_processor : processor {
    void process(shared_ptr<vector<shared_ptr<figure>>> buffer) override;
};

struct file_processor : processor {
    void process(shared_ptr<vector<shared_ptr<figure>>> buffer) override;
private:
    int cnt = 0;
};

#endif // _D_PROCESSOR_H_
```

main.cpp

```
//Попов Илья Павлович
//М80-206Б-20
```

```
//Лабораторная работа №8
//Вариант № 24
//Фигуры - квадрат, треугольник, 8-угольник

/*
Программа считывает из стандартного ввода данные фигур, согласно варианту задания, выводит их характеристики на экран и записывать в файл.
*/

/*
Программа должна:
1. Осуществлять ввод из стандартного ввода данных фигур, согласно варианту задания;
2. Программа должна создавать классы, соответствующие введенным данных фигур;
3. Программа должна содержать внутренний буфер, в который помещаются фигуры. Для создания буфера допускается использовать стандартные контейнеры STL. Размер буфера задается параметром командной строки. Например, для буфера размером 10 фигур: oop_exercise_08 10
4. При накоплении буфера они должны запускаться на асинхронную обработку, после чего буфер должен очищаться;
5. Обработка должна производиться в отдельном потоке;
6. Реализовать два обработчика, которые должны обрабатывать данные буфера:
a. Вывод информации о фигурах в буфере на экран;
b. Вывод информации о фигурах в буфере в файл. Для каждого буфера должен создаваться файл с уникальным именем.
7. Оба обработчика должны обрабатывать каждый введенный буфер. Т.е. после каждого заполнения буфера его содержимое должно выводиться как на экран, так и в файл.
8. Обработчики должны быть реализованы в виде лямбда-функций и должны хранится в специальном массиве обработчиков.
   Откуда и должны последовательно вызываться в потоке – обработчике.
9. В программе должно быть ровно два потока (thread). Один основной (main) и второй для обработчиков;
10. В программе должен явно прослеживаться шаблон Publish-Subscribe. Каждый обработчик должен быть реализован как отдельный подписчик.
11. Реализовать в основном потоке (main) ожидание обработки буфера в потоке-обработчике. Т.е. после отправки буфера на обработку основной поток должен ждать,
    пока поток обработчик выведет данные на экран и запишет в файл.
*/

#include <vector>
#include "subscriber.h"


void usage() {
    cout << "_____" << endl;
    cout << "Available commands:\n";
    cout << "add <square, triangle, octagon>\n";
    cout << "menu - print this menu\n";
    cout << "exit\n";
    cout << "_____" << endl;
}

bool is_number(const string& s) {
    bool point = false;
    for (int i = 0; i < s.length(); ++i) {
        if (s[i] == '-' && i == 0) {
            continue;
        }
        else if (s[i] == '.') {
            if ((i == 0 || i == s.length() - 1) || point) {
                return false;
```

```
            }
            else {
                point = true;
            }
        }
        else if (s[i] < '0' || s[i] > '9') { return false; }
    }
    return true;
}

int main(int argc,char* argv[]) {
    try {
        if (argc != 2) {
            throw logic_error("2 arguments needed!\n\n");
            return -1;
        }
        usage();

        if (!is_number(argv[1])) {
            throw logic_error("Wrong argument!\n\n");
            return -2;
        }

        int cur = stoi(argv[1]);

        if (cur <= 0) {
            throw logic_error("Wrong argument!\n\n");
            return -3;
        }
    }
    catch (invalid_argument& err) {
        cout << err.what() << '\n\n';
    }

    int  buffer_size = stoi(argv[1]);

    shared_ptr<vector<shared_ptr<figure>>> buffer = make_shared<vector<shared_ptr<figure>>>();
    buffer->reserve(buffer_size);

    factory factory;

    subscriber sub;
    sub.processors.push_back(make_shared<stream_processor>());
    sub.processors.push_back(make_shared<file_processor>());
    thread sub_thread(ref(sub));

    bool quit = false;
    string cmd;
    try {
        while (!quit) {
            unique_lock<mutex> locker(sub.mtx);

            cin >> cmd;
            if (cmd == "add") {
                buffer->push_back(factory.new_figure(cin));
                cout << "Figure has been successfuly added!";
                cout << endl << "_____________________________________________" << endl;


                if (buffer->size() == buffer_size) {
```

```
                cout << "Buffer is full\n";
                sub.buffer = buffer;
                sub.cond_var.notify_all();
                sub.cond_var.wait(locker, [&](){ return sub.buffer == nullptr;});
                buffer->clear();
            }
        }
        else if (cmd == "menu") {
            usage();
        }
        else if (cmd == "exit")  {
            quit = true;
        }
        else {
            cout << "Wrong command\n";
        }
    }
  }
  catch (logic_error &err) {
     cout << err.what() << "\n\n";
  }


  sub.stop = true;
  sub.cond_var.notify_all();
  sub_thread.join();

  return 0;
}
```

6. Вывод

Асинхронное программирование состоит в поддержке множества потоков внутри одного процесса и позволяет:

- Разделить ответственность за разные задачи между разными потоками
- Повысить быстродействие программы

Кроме того, часто различным составляющим программы необходимо обмениваться данными, использовать общие данные или результаты друг друга. Такую возможность предоставляют потоки внутри процесса, так как они используют адресное пространство процесса, которому принадлежат.

Асинхронное программирование очень полезно при создании сложных проектов, и любому программисту необходимо владеть навыками работы с ним.

ЛИТЕРАТУРА

1. Лямбда-выражения(анонимные функции) в C++ [Электронный ресурс].
URL: https://ravesli.com/lyambda-vyrazheniya-anonimnye-funktsii-v-s/(дата обращения: 15.12.2021).

2. Лямбда-захваты в C++ [Электронный ресурс].
URL: https://ravesli.com/lyambda-zahvaty-v-s/(дата обращения: 15.12.2021).
3. Написание многопоточных приложений на C++ [Электронный ресурс].
URL: https://eax.me/cpp-multithreading/(дата обращения: 15.12.2021).