МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ

«Национальный исследовательский университет «ИТМО» (НИУ ИТМО)

Лабораторная работа №5
«C++ OOP/Parallel»
по курсу «C++ и UNIX системы»

Выполнила студентка 3 курса группы К3333: Киреева М.С.

Проверил: Маслов И.Д.

Цель

Познакомить студента с принципами параллельных вычислений. Составить несколько программ в простейшими вычислительными действиями, чтобы освоить принципы параллельных вычислений (когда одни алгоритмы зависят / не зависят от других).

Задача

1. [C++ PARALLEL LANG] Создать параллельный язык программирования

Требуется создать язык программирования, в котором будет доступна установка следующих команд:

Установка счетного цикла

Вывод в консоль

Вывод в файл в режиме добавления

Арифметические операции +, -, *, /

Счетный цикл должен поддерживать дальнейшую установку всех остальных поддерживаемых команд.

Для реализации задачи использовать технологию объектно-ориентированного программирования в части реализации поддерживаемых команд языка.

В программе должны быть отражены следующие шаги:

1.1. Текстовый ввод команд. Каждая новая строка – это новый набор команд.

1.2. Ожидание команды на окончание ввода

1.3. Параллельное исполнение введенных строк (наборов команд). Наборы команд

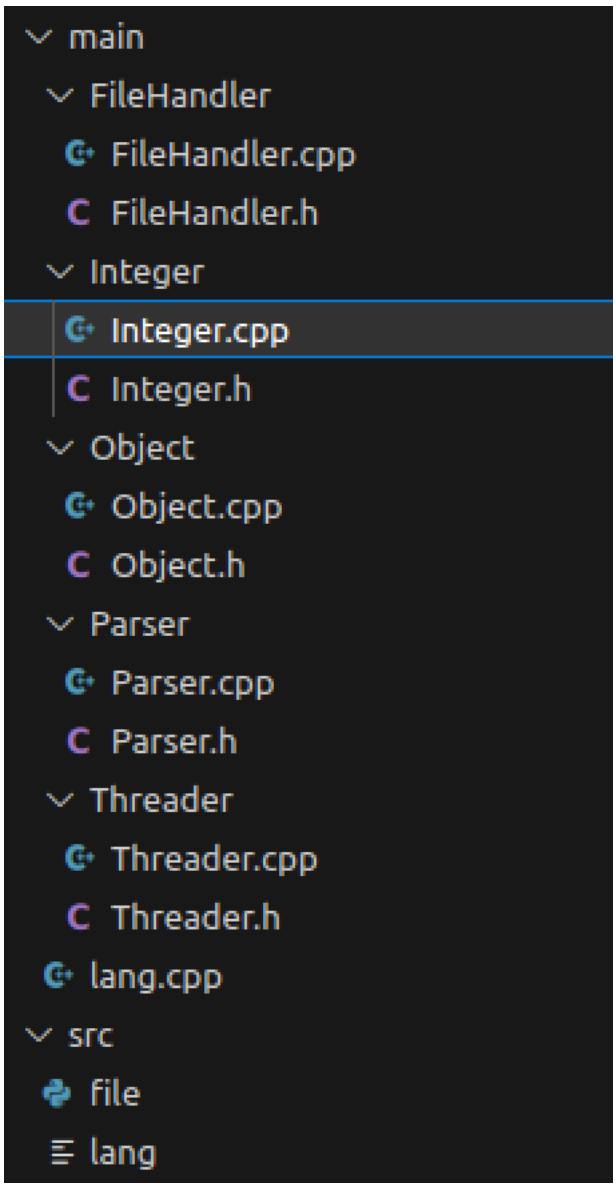
должны исполняться параллельно. В консоли фиксировать время запуска / завершения каждого потока. При выводе информации о времени указывать принадлежность потока к строке (набору команд)

2. [SAVE] Результат всех вышеперечисленных шагов сохранить в репозиторий (+ отчет по данной ЛР в папку doc)

Фиксацию ревизий производить строго через ветку dev. С помощью скриптов

накатить ревизии на stg и на prd. Скрипты разместить в корне репозитория. Также создать скрипты по возврату к виду текущей ревизии (даже если в папке имеются несохраненные изменения + новые файлы).

Структура проекта



Точка входа в программу

main > lang.cpp

```cpp
#include "FileHandler/FileHandler.h"
#include "Threader/Threader.h"

using namespace std;

int main(int argc, char *argv[]) {
    if (argc == 2){
        FileHandler file_handler(argv[1]);
        file_handler.handle();
        Threader threader(file_handler.getCommands());
        threader.executeThread();
    }
    else {
        cout << "Error! Wrong number of arguments" << "\n";
    }

    return 0;
}
```

Считывание файла с программным кодом

```cpp
#include "FileHandler.h"

FileHandler::FileHandler(char *name) {
    file_name = name;
}

void FileHandler::handle() {
    open();
    if (isOpen()) {
        read();
    }
    close();
}

void FileHandler::open(){
    file.open(file_name);
}

void FileHandler::close() {
    file.close();
}

bool FileHandler::isOpen() {
    if(!file.is_open()) {
        std::cout << "Error! Couldn't open file." << "\n";
        return false;
    }
    return true;
}

std::vector<std::string> FileHandler::getCommands() {
    return commands;
}
```

```
void Fil  typedef std::__cxx11::basic_stri
    std::string line;
    while(!file.eof()) {
        line="";
        std::getline(file,line);
        commands.push_back(line);
    }
}
```

Выделение языковых конструкций и команд

```
#include "Parser.h"


void Parser::parse(std::string command, int id) {
    auto start = std::chrono::steady_clock::now();
    Parser::executeCommands(command);
    auto end = std::chrono::steady_clock::now();
    std::cout << "Thread " << id << " execution time: " <<
    std::fixed << std::setprecision(3) << std::chrono::duration<double>(end - start).count() * 1000 <
}


void Parser::executeCommands(std::string line) {
    std::vector<std::string> commands = Parser::split(line, ',');
            Object obj
    Object obj;
    Integer int_obj;

    for(std::string command : commands) {
        Parser::distributeCommand(command, obj, int_obj);
    }

}


void Parser::distributeCommand(std::string command, Object &obj, Integer &int_obj) {
    if (Parser::findFirstSubstring(command, "loop")) {
        std::vector <int> loop_params = Parser::getLoopParams(command);

        Parser::getLoopBody(command);

        std::vector<std::string> loop_commands = Parser::split(command, ';');

        for(std::string com : loop_commands)
            Parser::dropSpaces(com);
```

```
            for(int i=loop_params[0]; i < loop_params[1]; i+= loop_params[2])
                for(std::string com : loop_commands) {
                    Parser::caseCommands(com, obj, int_obj);
                }
        }
        else {
            Parser::dropSpaces(command);
            Parser::caseCommands(command, obj, int_obj);
        }
    }


    std::vector<int> Parser::getLoopParams(std::string command) {
        std::vector<int> a;
        std::string line = Parser::getLineInBound(command, "(", ")");
        std::vector<std::string> command_elements = Parser::split(line, ':');

        for(std::string element : command_elements)
            a.push_back(std::stoi(element));

        return a;
    }
```

```
void Parser::caseCommands(std::string command, Object &obj, Integer &int_obj) {
        class Parser
    if(Parser::findFirstSubstring(command, "print")){

        Parser::print(command, obj, int_obj);
    }
    else if (Parser::findFirstSubstring(command, "file"))
    {
        Parser::writeFile(command);
    }
    else if (Parser::findFirstSubstring(command, "int"))
    {
        Parser::initType(command, int_obj);
    }
    else if (Parser::findFirstSubstring(command, "str"))
    {
        Parser::initType(command, obj);
    }
    else if (Parser::findFirstSubstring(command,"=")){
        Parser::calculations(command, int_obj);
    }
}
```

```
void Parser::writeFile(std::string command){
    std::string line = Parser::getLineInBound(command, "(", ")");
    std::vector<std::string> command_elements = Parser::split(line, ':');

    Parser::dropBounds(command_elements[0]);

loads/lab5/main/Integer e(command_elements[0], std::ios::app);

    if(file.is_open()) {
        Parser::dropBounds(command_elements[1]);

        file << command_elements[1];

        file.close();

        std::cout << "Data is written successful." << "\n";
    }
    else
        std::cout << "Error of openning file!" << "\n";
}


void Parser::dropBounds(std::string &str) {
    str.erase(0, 1);
    str.pop_back();
}
```

```
std::string Parser::getLineInBound(std::string value, std::string first, std::string last) {
    std::string line = Parser::findBetweenSymbols(value, first, last);
    Parser::dropBounds(line);
    return line;
}


void Parser::print(std::string command, Object obj, Integer int_obj) {
    std::string print_line = Parser::getLineInBound(command, "(", ")");

    if (Parser::findFirstSubstring(print_line, "'")){
        Parser::dropBounds(print_line);
        std::cout << print_line << "\n";
    }
    else {
        if (obj.findKey(print_line)){
            std::cout << print_line << "\n";
        }
        else if (int_obj.findKey(print_line))
        {
            std::cout << int_obj.getValue(print_line) << "\n";
        }
        else
            std::cout << "Error! Variable isn't found" << "\n";
    }
}
```

```cpp
void Parser::calculations(std::string command, Integer &int_obj) {
    Parser::dropSpaces(command);
    std::vector<std::string> command_elements = Parser::split(command, ' ');

    if(command_elements.size() == 5) {
        int valueA = Parser::getNumber(command_elements[2], int_obj);
        int valueB = Parser::getNumber(command_elements[4], int_obj);

        if (Parser::findFirstSubstring(command, "+"))
        {
            int_obj.setValue(command_elements[0], valueA + valueB);
        }
        else if (Parser::findFirstSubstring(command, "-"))
        {
            int_obj.setValue(command_elements[0], valueA - valueB);
        }
        else if (Parser::findFirstSubstring(command, "*"))
        {
            int_obj.setValue(command_elements[0], valueA * valueB);
        }
        else if (Parser::findFirstSubstring(command, "/"))
        {
            if (valueB != 0)
                int_obj.setValue(command_elements[0], valueA / valueB);
            else
                std::cout << "Error! Division by zero!" << "\n";
        }
    }
}
```

```cpp
int Parser::getNumber(std::string str, Integer obj) {
    int value;
    if(Parser::isNumeric(str))
        value = std::stoi(str);
    else
        value = obj.getValue(str);
    return value;
}


bool Parser::isNumeric(std::string value) {
    for(int i=0;i<value.length();i++) {
        if(!isdigit(value[i])) {
            return false;
        }
    }
    return true;
}


template<typename Type>
void Parser::initType(std::string command, Type &obj){
    std::vector<std::string> command_elements = Parser::split(command, '

    if (command_elements.size() == 2) {
        obj.insert(command_elements[1], "");
    }

    if (command_elements.size() == 4) {
        obj.insert(command_elements[1], command_elements[3]);
    }
}
```

```
std::vector<std::string> Parser::split(std::string line, char separator) {
    namespace std
    ::string> result;
    std::stringstream s(line);
    std::string str;
    while(std::getline(s, str, separator)) {
        result.push_back(str);
    }
    return result;
}


bool Parser::findFirstSubstring(std::string str, std::string substr){
    std::size_t found = str.find(substr);
    return (found != std::string::npos) ? true : false;
}


void Parser::dropSpaces(std::string &line) {
    std::size_t first = line.find_first_not_of(" ");
    std::size_t last = line.find_last_not_of(" ");
    line = line.substr(first, last - first + 1);
}

std::string Parser::findBetweenSymbols(std::string line,std::string first_symbol, std::string last_symb
    std::size_t first = line.find(first_symbol);
    std::size_t last = line.find(last_symbol);
    return line.substr(first, last - first + 1);
}


void Parser::getLoopBody(std::string &loop){
    loop = Parser::getLineInBound(loop, "{", "}");
    Parser::dropSpaces(loop);
}
```

Параллельные вычисления

```
#include "Threader.h"

Threader::Threader(std::vector<std::string> commands) {
    thread_commands = commands;
}

void Threader::executeThread() {
    int size = thread_commands.size();
    for(int i=0;i<size; i+=N) {
        int start = i;
        int end = std::min(i+N, size);

        std::vector <std::thread> threads;

        for(int j=start; j<end; j++) {

            threads.push_back(std::thread(Parser::parse, thread_commands[j], j));
        }

        for(std::thread& thread : threads) {
            thread.join();
        }
    }
}
```

Объект

```
#include "Object.h"

void Object::insert(std::string key, std::string value = std::string()) {
    std::string newValue;
    if (isEmpty(value))
        newValue = value;
    if (isValidKey(key))
        data[key] = newValue;
}


std::string Object::getValue(std::string name){
    return data[name];
}

bool Object::isValidKey(std::string key) {
    if (key.length() > 64)
        return false;
    if (isString(key))
        return false;
    return true;
}


bool Object::isEmpty(std::string value) {
    return (value.length() == 0) ? false : true;
}

bool Object::isString(std::string value) {
    for(int i=0; i<value.length(); i++) {
        if(isdigit(value[i]) == true)
            return true;
    }
    return false;
}
```

```
bool Object::findKey(std::string key) {
    return (data.find(key) == data.end()) ? false : true;
}
```

Объекты типа Integer

```cpp
#include "Integer.h"


void Integer::insert(std::string key, std::string value= std::string()) {
    try {
        int newValue;
        if (isEmpty(value)) {
            newValue = std::stoi(value);
        }
        if (isValidKey(key))
            data[key] = newValue;
    } catch (const std::exception&) {
        std::cout << "Error of type convertion " << "\n";
        return;
    }
}

int Integer::getValue(std::string name){
    return data[name];
}

void Integer::setValue(std::string name, int value) {
    data[name] = value;
}

bool Integer::findKey(std::string key) {
    return (data.find(key) == data.end()) ? false : true;
}
```

Вывод:

В ходе работы был реализован новый язык программирования, построенный на принципах ООП и параллельных вычислений.