ЛР 5: [C++ UNIX]: C++ OOP / PARALLEL

Гротов Константин
Z33431

Цель работы

Познакомить студента с принципами объектно-ориентированного программирования на примере создания сложной синтаксической структуры. Придумать синтаксис своего персонального мини-языка параллельного программирования, а также реализовать его разбор и вычисление.

1 Задача

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

1. Установка счетного цикла
2. Вывод в консоль
3. Вывод в файл в режиме добавления
4. Арифметические операции +, -, *, /
5. 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. Текстовый ввод команд. Каждая новая строка – это новый набор команд.
2. Ожидание команды на окончание ввода
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 Решение

Изначально были созданы два абстрактных класса, для последующей удобной реализации бинарных и унарных операторов.

```

class BinaryAction {
   public:
      virtual string getOutput() = 0;

    protected:
        double a, b;
```

```
        string c, d;
};


class UnaryAction {
   public:
      virtual string getOutput() = 0;

    protected:
        string a;
};


class PlusAction: public BinaryAction {
    public:
        PlusAction(double a_in, double b_in) { a = a_in; b = b_in; }
        string getOutput() {
            return to_string(a + b);
        }
};


class MinusAction: public BinaryAction {
   public:
           MinusAction(double a_in, double b_in) { a = a_in; b = b_in; }

      string getOutput() {
         return to_string(a - b);
      }
};


class MulAction: public BinaryAction {
   public:
   MulAction(double a_in, double b_in) { a = a_in; b = b_in; }

      string getOutput() {
         return to_string(a * b);
      }
};


class DivAction: public BinaryAction {
   public:
   DivAction(double a_in, double b_in) { a = a_in; b = b_in; }
      string getOutput() {

         return (b != 0) ? to_string(a * b) : "Division by zero" ;
      }
};


class ToFileAction: public BinaryAction {
   public:
   ToFileAction(string c_in, string d_in) { c = c_in; d = d_in; }
      string getOutput() {
         write_to_file(c, d);
         return "String has written to file" ;
      }

```

```
    private:
        void write_to_file(string filename, string s) {
            ofstream file;

            file.open(filename);
            file << s << endl;
            file.close();
        }
};


class CliAction: public UnaryAction {
   public:
   CliAction(string a_in) { a = a_in; }
      string getOutput() {
        cout << "--- " << a << " --- ";
        return "String has written to CLI";
      }
};
```

Листинг 1: Вспомогательные структуры данных

Вместо того, чтобы составлять очередь напрямую из командной строки (хотя такая функциональность тоже присутствует) было решено читать код из файла. Далее вся работа будет происходить со следующим кодом:

```
+ -> 1 2.12411234
- -> 1 3
* -> 2 3.14
/ -> 15 3
to_cli -> hello world!
to_file -> outputs/heyhey.txt <== hey LOOOOL hey hey
loop 5 -> to_cli -> hello repeated!
```

На вход основному классу подается путь к данному файлу и при инициализации происходит формирования очереди. После инициализации можно вызвать либо последовательный вызов команд, либо параллельный.

```
class LoLanguage {
public:
    string source_code = "";
    queue <string> actions;

    LoLanguage(string filename) {
        string line_tmp;
        ifstream file(filename);

        if (file.is_open()) {
            while (getline(file, line_tmp)) {
                source_code += line_tmp + "\n";
                if (line_tmp != "")
                    actions.push(line_tmp);
            }
            file.close();
        }
    }

    pair<string, string> splitAction(string s, string delimeter = " -> ") {
        string act, input;

        act = s.substr(0, s.find(delimeter));
```

```
        input = s.substr(s.find(delimeter) + delimeter.length(), s.length())
    ;

        return make_pair(act, input);
    }

    void do_all_actions() {
        pair<string, string> action_pair;

        while (!actions.empty()) {
            action_pair = splitAction(actions.front());
            if (action_pair.first.find("loop") == 0) {
                // cout << stoi(splitAction(action_pair.first, " ").second)
    << endl;
                for (int i = 0; i < stoi(splitAction(action_pair.first, " ")
    .second); i++) {
                    do_action(action_pair.second);
                }
            }
            else
                do_action(actions.front());
            actions.pop();
        }
    }

    void threadFunction(string s, int thread_num) {
        cout << "[" << thread_num << "]" << " started" << endl;
        auto begin = chrono::high_resolution_clock::now();

        do_action(s, thread_num);

        auto end = chrono::high_resolution_clock::now();
        auto elapsed = chrono::duration_cast<chrono::nanoseconds>(end -
    begin);
        cout << "[" << thread_num << "]" << " ended in " << elapsed.count()
    * 1e-9 << "s" << endl;
    }

    void loopThreadFunction(string s, int thread_num) {
        pair<string, string> action_pair;

        cout << "[" << thread_num << "]" << " started" << endl;
        auto begin = chrono::high_resolution_clock::now();

        action_pair = splitAction(s);
        for (int i = 0; i < stoi(splitAction(action_pair.first, " ").second)
    ; i++) {
            do_action(action_pair.second, thread_num);
        }

        auto end = chrono::high_resolution_clock::now();
        auto elapsed = chrono::duration_cast<chrono::nanoseconds>(end -
    begin);
        cout << "[" << thread_num << "]" << " ended in " << elapsed.count()
    * 1e-9 << "s" << endl;
    }

    void do_all_actions_parallel() {
        vector<thread> threads;
        pair<string, string> action_pair;
```

```
76
77          void (LoLanguage::*func)(string, int);
78
79          int cnt = 0;
80          while (!actions.empty()) {
81              func = &LoLanguage::threadFunction;
82
83              action_pair = splitAction(actions.front());
84              if (action_pair.first.find("loop") == 0) {
85                  func = &LoLanguage::loopThreadFunction;
86                  threads.push_back(thread(func, *this, actions.front(), cnt))
    ; // do_action(actions.front());
87              }
88              else
89                  threads.push_back(thread(func, *this, actions.front(), cnt))
    ; // do_action(actions.front());
90
91              cnt += 1;
92              actions.pop();
93          }
94
95          for (int t = 0; t < threads.size(); t++) {
96              threads[t].join();
97          }
98      }
99
100     void do_action(string action, int thread=0) {
101         pair<string, string> action_pair = splitAction(action);
102         string flag = action_pair.first;
103         string input = action_pair.second;
104         string a, b;
105
106         if (thread)
107             cout << "[" << thread << "]" << endl;
108
109         switch (hashstring(flag.c_str())) {
110             case hashstring("+"): {
111                 action_pair = splitAction(input, " ");
112                 a = action_pair.first;
113                 b = action_pair.second;
114                 PlusAction A(stod(a), stod(b));
115                 cout << A.getOutput() << endl;
116                 break;
117             }
118             case hashstring("-"): {
119                 action_pair = splitAction(input, " ");
120                 a = action_pair.first;
121                 b = action_pair.second;
122                 MinusAction A(stod(a), stod(b));
123                 cout << A.getOutput() << endl;
124                 break;
125             }
126             case hashstring("*"): {
127                 action_pair = splitAction(input, " ");
128                 a = action_pair.first;
129                 b = action_pair.second;
130                 MulAction A(stod(a), stod(b));
131                 cout << A.getOutput() << endl;
132                 break;
133             }
```

```
            case hashstring("/"): {
                action_pair = splitAction(input, " ");
                a = action_pair.first;
                b = action_pair.second;
                DivAction A(stod(a), stod(b));
                cout << A.getOutput() << endl;
                break;
            }
            case hashstring("to_file"): {
                action_pair = splitAction(input, "<==");
                a = action_pair.first;
                b = action_pair.second;
                ToFileAction A(a, b);
                cout << A.getOutput() << endl;
                break;
            }
            case hashstring("to_cli"): {
                CliAction A(input);
                cout << A.getOutput() << endl;
                break;
            }
        }
    }
};
```

Листинг 2: Основной класс разбора языка программирования

3 Выводы

В ходе лабораторной работы был получен опыт работы с ООП, а также закреплен опыт с параллельными вычисления на языке C++. Был придуман синтаксис языка прогарммирвоания и осуществлен его разбор в последовательном и параллельном варианте.