Министерство образования и науки Российской Федерации
Московский государственный университет им. Н. Э. Баумана
Кафедра «РК6» Группа 36Б

Отчет по практикуму по языку программирования C++ №3

Вариант 16

Выполнил: Самойлов А. А.

Проверил: Берчун Ю.В.

Дата: 29.12.2020

Подпись: _____

Москва, 2020 г.

Задание

Требуется разработать программу, реализующую дискретно-событийное моделирование системы, рассмотренной в задании 2 домашнего задания №4. Обратите внимание, что все интервалы времени подчиняются законам распределений, носящим непрерывный характер. Поэтому категорически неверными является выбор целочисленных типов данных для моментов и интервалов времени, и тем более инкремент модельного времени с единичным шагом. Нужно реализовать именно переход от события к событию, как это сделано в GPSS и других проблемно-ориентированных системах. Для упрощения можно ограничиться использованием единственного потока случайных чисел для генерации всех необходимых случайных величин. Результатом работы программы должен быть лог-файл, содержащий записи типа: «В момент времени 12.345 транзакт с идентификатором 1 вошёл в модель», «В момент времени 123.456 транзакт с идентификатором 123 встал в очередь 1», «В момент времени 234.567 транзакт с идентификатором 234 занял устройство 2», «В момент времени 345.678 транзакт с идентификатором 345 освободил устройство 1», «В момент времени 456.789 транзакт с идентификатором 456 вышел из модели».

Алгоритм

Создадим класс Transact, имитирующий транзакт в нашей системе. У него должны быть приватные поля:

1) Статическая целочисленная переменная CURRENT_ID, отвечающая за идентификатор каждого объекта класса. При получении каждого последующего номера идентификации следует увеличивать значение переменной на один, чтобы избежать повторения значения;
2) Целочисленная переменная _ID, содержащая идентификационное значение каждого экземпляра класса;
3) Число с плавающей запятой, обозначающее время следующего события, связанного с данным транзактом;
4) Целочисленная переменная, соответствующая номеру текущего состояния.

Публичные поля класса Transact:

1) Конструктор, устанавливающий идентификатор через CURRENT_ID, и при этом увеличивающий его для того, чтобы у каждого следующего объекта идентификатор отличался;
2) Методы GetID, GetTime и GetCurrentState, возвращающие идентификатор, время и номер текущего состояния, и метод SetTime и SetCurrentState, устанавливающие время и номер текущего состояния;
3) Дружественная перегрузка оператора ==, нужная для возможности сравнения двух транзактов.

Создадим абстрактный класс State, представляющий собой состояние. Его защищённые поля:

1) CURRENT_ID и _ID для реализации идентификации состояний;
2) Номер следующего состояния, куда должен отправиться транзакт по завершению работы в данном состоянии.

Публичные методы этого класса:

1) Конструктор, устанавливающий идентификатор через CURRENT_ID, и при этом увеличивающий его для того, чтобы у каждого следующего объекта идентификатор отличался;
2) Метод GetID, возвращающий идентификатор и SetNextState, номер следующего состояния;
3) Абстрактный метод UseTransact, который должен быть реализован в каждой конкретной реализации состояния.

Первым публичным наследником класса State будет класс Generate, который отвечает за создание транзактов в системе (является аналогом GENERATE из GPSS). В нём будут храниться приватные переменные:

1) Число с плавающей запятой _time, изначально устанавливаемое на 0.0 и отвечающее за время появления каждого последующего транзакта;
2) Целочисленные переменные _MIN и _MAX для генератора случайных чисел, являющиеся минимальным и, соответственно, максимальным числом.

Публичные методы:

1) Конструктор, вызывающий конструктор базового класса State и устанавливающий значение _time в ноль;
2) Метод GetTime, возвращающий время, к которому сразу прибавляется случайное число в диапазоне от _MIN до _MAX, и SetMinMax, устанавливающий минимальное и максимальное значения;
3) Реализация абстрактного метода UseTransact, который меняет текущее состояние у появившегося транзакта и выводит соответствующее информационное уведомление.

Создадим класс Querry, который реализует очередь (аналог для QUERRY из GPSS). Так как по своей сути он является «обёрткой» для класса list библиотеки “list”, то для реализации достаточно всего двух приватных полей:

1) Сам лист для указателей на объекты класса Transfer;
2) Целочисленная переменная _capacity для возможности ограничения длины очереди.

Публичные методы класса также дополняют возможности взаимодействия с листом:

1) Конструктор, по умолчанию устанавливающий _capacity равным -1 для создания бесконечной очереди;
2) Метод Add, добавляющий (если возможно) указатель на трансфер в список;

3) Метод Has, проверяющий нахождение трансфера в списке;
4) Метод Size, возвращающий число находящихся в списке указателей на трансферы;
5) Метод Get, возвращающий первый по списку указатель на трансфер, при этом удалив его из списка.

Теперь создадим класс Operator (аналог для OPERATE из GPSS), являющийся наследником State, который, непосредственно, обрабатывает поступающие в него транзакты. Так как оператор не может обслуживать одновременно более одного транзакта, воспользуемся созданным классом Querry и создадим очередь. Класс оператора должен содержать следующие приватные поля:

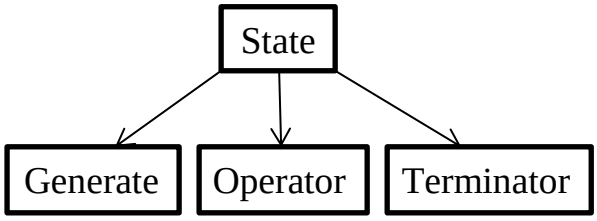
1) Целочисленные переменные _MIN и _MAX для генератора случайных чисел, являющиеся минимальным и, соответственно, максимальным числом;
2) Булевый флаг _isOccuped, значение в котором соответствует тому, обслуживается ли в настоящее время транзакт;
3) Объект _querry, представляющий собой очередь к опреатору;
4) Указатель на транзакт _occuped, который в настоящее время находится в обработке оператором.

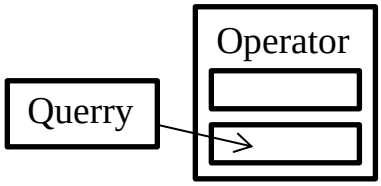
Публичные методы класса оператора:

1) Конструктор, по умолчанию вызывающий конструктор базового класса, определяющий бесконечную очередь, и устанавливающий флаг _isOccuped в значение false;
2) Метод SetMinMax, устанавливающий минимальное и максимальное значения, IsOccuped, возвращающий значение флага _isOccuped и метод GetQuerrySize, возвращающий текущую длину очереди;
3) Реализацию метода UseTransact, которая, в зависимости от текущего состояния оператора (загруженность, наличие очереди, освобождение оператора) совершает действие обработки транзакта.

Наконец, создадим класс Terminator (аналог для TERMINATE из GPSS), также наследующий абстрактный класс State, отвечающий за уничтожение покидающих систему транзактов (это необходимо для исключения проблемы нехватки памяти). Класс также является наследником State, и содержит только два публичных метода:

1) Конструктор, вызывающий конструктор базового класса;
2) Реализацию метода UseTransact, который и занимается очищением памяти от транзакта посредством delete.





Реализация главной функции нашей программы main состоит, во-первых, из проверки введённых пользователем данных, а во-вторых, из создания системы обработки транзактов (два объекта класса operator, по одному объекту классов generate и terminator) и соответствующей настройки порядка проверки посредством switch для текущего состояния каждого из транзактов.

Для начала, создадим транзакт, время появления которого задаётся пользователем как время моделирование системы. Его появление будем рассматривать как сигнал о том, что пора завершать работу системы. Затем создадим стартовый транзакт – присвоим ему время, полученное от Generate. По сути, транзакт ещё не создан в системе, ведь время его создания ещё не наступило, так что данный транзакт представляет собой «резервирование» следующего создаваемого транзакта. Каждый раз, когда наступает время зарезервированного транзакта, он появляется в системе (метод UseTransact у Generate), и при этом создаётся следующий зарезервированный транзакт, время появления которого также определяется методом GetTime у Generate. Это, в первую очередь, нужно для экономии памяти.

Наконец, создадим список указателей на транзакты – в него будут заноситься все созданные или зарезервированные транзакты. Для прохождения по списку нам понадобится итератор, а для завершения моделирования – флаг endFlag.

Теперь осталось создать основной цикл, реализующий моделирование системы. Используем while с условием инверсии значения флага endFlag. В начале цикла определяем наименьшее время среди всех существующих транзактов и запоминаем его. Далее, снова проходим по списку и сравниваем время транзакта с наименьшим. Если они совпадают, то вызываем switch для значения текущего состояния транзакта:

Состоянию 0 соответствует зарезервированность транзакта. В этом случае его следует включить в систему посредством Generate и зарезервировать следующий транзакт для системы. При обнаружении завершающего транзакта, установим флаг endFlag в значение true.

Состоянию 1 соответствует выбор оператора для последующей обработки. Если кто-то из операторов свободен, то транзакт направляется к нему, если оба оператора заняты, то длины очередей операторов сравниваются в поиске наименьшей величины, и транзакт отправляется к оператору с наименьшей очередью, если операторы идентичны по занятости, выбирается первый оператор.

Состояниям 2 и 3 соответствует обработка в первом или во втором операторах.

Состоянию 4 соответствует прохождение оператором обработки и необходимость удалить его из списка транзактов (при этом важно также очистить использованную им память, во избежание переполнения). Для этого воспользуемся объектом класса Terminator.

Код

“transact.h”

```
#ifndef TRANSACT_H
#define TRANSACT_H

class Transact {
   public:
      Transact() { _ID = CURRENT_ID++; _currentState = 0; }
      unsigned int GetID() const { return _ID; }
      unsigned int GetCurrentState() { return _currentState; }
      void SetCurrentState(unsigned int state) { _currentState = state; }
      float GetTime() { return _timeNextEvent; }
      void SetTime(float _time) { _timeNextEvent = _time; }
      friend bool operator == (Transact& one, Transact& another);
   private:
      static unsigned int CURRENT_ID;
      unsigned int _ID;
      float _timeNextEvent;
      unsigned int _currentState;
};

#endif
```

“transact.cpp”

```
#include "transact.h"

unsigned int Transact::CURRENT_ID = 0;
```

```
bool operator == (Transact& one, Transact& another) {
   return one._ID == another._ID;
}
```

“state.h”

```
#ifndef STATE_H
#define STATE_H

#include <iostream>
#include <fstream>
#include <list>
#include "transact.h"

using namespace std;

class Querry {
   public:
      Querry(int capacity = -1) : _capacity(capacity) { }
      int Size() { return _transacts.size(); }
      int Add(Transact* transact);
      Transact* Get() { Transact* ret = _transacts.front(); _transacts.pop_front();
return ret;}
      bool Has(Transact& transact);
   private:
      list <Transact*> _transacts;
      int _capacity;
};

class State {
```

```
    public:
        State() { _ID = CURRENT_ID++; }
        unsigned int GetID() const { return _ID; }
        void SetNextState(unsigned int nextState, ofstream& file) { _nextState =
nextState; }
        virtual int UseTransact(Transact& transact) = 0;
    protected:
        static unsigned int CURRENT_ID;
        unsigned int _ID;
        unsigned int _nextState;
};

class Generate : public State {
    public:
        Generate() : State() { _time = 0.0; }
        float GetTime() { return _time += (_MIN + (rand() % _MAX) +
(float)rand()/RAND_MAX); }
        int UseTransact(Transact& transact, ofstream& file);
        void SetMinMax(int min, int max) { _MIN = min; _MAX = max; }
    private:
        float _time;
        int _MIN;
        int _MAX;
};

class Operator : public State {
    public:
        Operator(int querryCapacity = -1) : State(), _querry(querryCapacity) {
_isOccuped = false; }
```

```
    void SetMinMax(int min, int max) { _MIN = min; _MAX = max; }
    int UseTransact(Transact& transact, ofstream& file);
    int GetQuerrySize() { return _querry.Size(); }
    bool IsOccuped() { return _isOccuped; }
  private:
    int _MIN;
    int _MAX;
    bool _isOccuped;
    Transact* _occuped;
    Querry _querry;
};

class Terminator : public State {
  public:
    Terminator() : State() { }
    int UseTransact(Transact& transact, ofstream& file);
};

#endif
```

“state.cpp”

```
#include <iterator>
#include "state.h"

int Querry::Add(Transact* transact) {
  if((_capacity == -1) || (_transacts.size() < _capacity)) {
    _transacts.push_back(transact);
  } else {
```

```
        return 1;
    }
}

bool Querry::Has(Transact& transact) {
    list <Transact*> :: iterator it;
    for (it = _transacts.begin(); it != _transacts.end(); ++it) {
        if((**it) == transact) { return true; }
    }
    return false;
}

unsigned int State::CURRENT_ID = 0;

int Generate::UseTransact(Transact& transact, ofstream& file) {
    transact.SetCurrentState(_nextState);
    file << "At " << transact.GetTime() << "s transact " << transact.GetID() << " has
appeared"  << endl;
}

int Operator::UseTransact(Transact& transact, ofstream& file) {
    if(!_isOccuped) {
        file << "At " <<  transact.GetTime() << "s transact " << transact.GetID() << " has
begun operating at " << _ID  << " operator" << endl;
        transact.SetTime(transact.GetTime() + (_MIN + (rand() % _MAX) +
(float)rand()/RAND_MAX));
        _occuped = &transact;
        _isOccuped = true;
    } else if (&transact == _occuped) {
```

```
        file << "At " << transact.GetTime() << "s transact " << transact.GetID() << " has ended operating at " << _ID << " operator" << endl;

        transact.SetCurrentState(_nextState);

        if (_querry.Size() > 0) {

            Transact* next = _querry.Get();

            file << "At " << next->GetTime() << "s transact " << next->GetID() << " has begun operating at " << _ID << " operator" << endl;

            next->SetTime(next->GetTime() + (_MIN + (rand() % _MAX) + (float)rand()/RAND_MAX));

            _occuped = next;

        } else {

            _isOccuped = false;

        }

    } else {

        if (!_querry.Has(transact)) {

            _querry.Add(&transact);

            file << "At " << transact.GetTime() << "s transact " << transact.GetID() << " has gone to querry at " << _ID << " operator on " << _querry.Size() << " place" << endl;

        }

        transact.SetTime(_occuped->GetTime());

    }

}


int Terminator::UseTransact(Transact& transact, ofstream& file) {

    file << "At " << transact.GetTime() << "s transact " << transact.GetID() << " has left" << endl;

    delete &transact;

}
```

“main.cpp”

```
#include "transact.h"
#include "state.h"

int main(int argc, char** argv)
{
   if (argc < 6) { cout << "Not enough input data" << endl; return -1; }
   int r1 = atoi(argv[1]);
   int g1 = atoi(argv[2]);
   int b1 = atoi(argv[3]);
   int rnd = atoi(argv[4]);
   int endTime = atoi(argv[5]);

   ofstream file;
   file.open("debug.txt");
   if(!file.is_open()) { cout << "Cannot use file" << endl; return 2; }

   srand(rnd);

   Generate generate;
   generate.SetNextState(1);
   generate.SetMinMax(0, r1+g1+b1);

   Operator* operators[2];
   operators[0] = new Operator();
   operators[0]->SetNextState(4);
   operators[0]->SetMinMax(r1, r1+g1+b1);
   operators[1] = new Operator();
   operators[1]->SetNextState(4);
```

```
operators[1]->SetMinMax(g1, r1+g1+b1);

Terminator terminator;

list <Transact*> transacts;

Transact* end = new Transact();
end->SetCurrentState(0);
end->SetTime(endTime);
transacts.push_front(end);

Transact* start = new Transact();
start->SetCurrentState(0);
start->SetTime(generate.GetTime());
transacts.push_front(start);

list <Transact*> :: iterator it;
bool endFlag = false;
while (!endFlag) {
   float minTime = (*transacts.begin())->GetTime();
   for (it = transacts.begin(); it != transacts.end(); ++it) {
      minTime = minTime > (*it)->GetTime() ? (*it)->GetTime() : minTime;
   }
   for (it = transacts.begin(); it != transacts.end(); ++it) {
      if ((*it)->GetTime() == minTime) {
         switch((*it)->GetCurrentState()) {
            case 0: {
               if(**it == *end) {
```

```
            endFlag = true;
        } else {
            generate.UseTransact(**it, file);
            Transact* newTrans = new Transact();
            newTrans->SetTime(generate.GetTime());
            transacts.push_back(newTrans);
        }
        break;
    }
    case 1: {
        bool flag = false;
        for (int i = 0; i < sizeof(operators)/sizeof(*operators); ++i) {
            if (!(operators[i]->IsOccuped())) {
                (*it)->SetCurrentState(i+2);
                operators[i]->UseTransact(**it, file);
                flag = true;
                break;
            }
        }
        if(!flag) {
            int min = operators[0]->GetQuerrySize();
            int minIndex = 0;
            for (int i = 0; i < sizeof(operators)/sizeof(*operators); ++i) {
                if (operators[i]->GetQuerrySize() < min) {
                    minIndex = i;
                    min = operators[i]->GetQuerrySize();
                }
            }
```

```
                    (*it)->SetCurrentState(minIndex+2);
                    operators[minIndex]->UseTransact(**it, file);
                }
                break;
            }
            case 2:
                operators[0]->UseTransact(**it, file);
                break;
            case 3:
                operators[1]->UseTransact(**it, file);
                break;
            case 4:
                terminator.UseTransact(**it, file);
                it = transacts.erase(it);
                break;
        }
    }
  }

  file.close();
  return 0;
}
```

Тест (результаты записи в файл)

При вводе 6 5 8 1 30:

At 2.39438s transact 1 has appeared

At 2.39438s transact 1 has begun operating at 1 operator

At 6.19282s transact 2 has appeared

At 6.19282s transact 2 has begun operating at 2 operator

At 13.7468s transact 2 has ended operating at 2 operator

At 13.7468s transact 2 has left

At 16.9611s transact 3 has appeared

At 16.9611s transact 3 has begun operating at 2 operator

At 18.5899s transact 4 has appeared

At 18.5899s transact 4 has gone to querry at 1 operator on 1 place

At 18.5919s transact 1 has ended operating at 1 operator

At 18.5919s transact 4 has begun operating at 1 operator

At 18.5919s transact 1 has left

At 20.5061s transact 5 has appeared

At 20.5061s transact 5 has gone to querry at 1 operator on 1 place

At 23.4745s transact 3 has ended operating at 2 operator

At 23.4745s transact 3 has left

At 26.1131s transact 6 has appeared

At 26.1131s transact 6 has begun operating at 2 operator

Литература

1. http://bigor.bmstu.ru/
2. Конспекты лекций
3. «Язык программирования C++», Бьёрн Страуструп
4. Стивен Прата. «Язык программирования C++. Лекции и упражнения»
5. Роберт Лафоре. «Объектно-ориентированное программирование в C++»