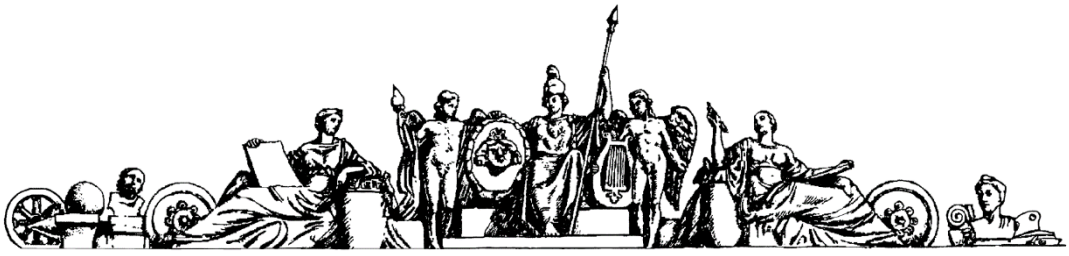

Министерство образования и науки Российской Федерации
Федеральное агентство по образованию
Государственное образовательное учреждение высшего профессионального образования
«Московский государственный технический университет имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)
Факультет «Робототехника и комплексная автоматизация» (РК)
Кафедра «Системы автоматизированного проектирования» (РК6)

Практикум по «Теории вероятности».

Студент: Сергеева Диана

Группа: РК6-36Б

Преподаватель: Берчун Ю.В

Проверил:

Дата:

2020

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».

В программе создадим классы:

1. Transact с полями:

```
class Transact
{
private:
    static int _currID;             //Идентификатор объекта класса
    int _ID;                        //Идентификатор экземпляра класса
    double _time;                   //Время следующей обработки
    int _state;                     //Номер текщего состояния
public:
    Transact();                         //Конструтор класса
    int getID();                        //Возвращает идентификатор
    double getTime();                   //Возвращает время
    int getState();                     //Возвращает текщее состяние
    void setTime(double time);          //Устанавливает время
    void setState(int state);           //Устанавливает номер текщего состояния
};
```

2. State с полями:

```
class State
{
protected:
    static int _currID;         //Идентификатор объекта класса
    int _ID;                    //Идентификатор экземпляра класса
    int _state;                 //Номер следующего состояния транзакта
public:
    State();                                //Конструтор класса
    int getID();                            //Возвращает идентификатор
    void setState(int state);               //Возвращает номер следующего состояния
    virtual void useTransact(Transact& transact, std::ofstream& file) = 0;  //Виртуальный метод
};
```

3. GENERATE с полями:

```
class GENERATE : public State
{
private:
    double _time;                   //Время появления события
    int _max;                       //Максимальное число для генератора
    int _min;                       //Минимальное число для генератора
public:
    GENERATE(int min, int max);         //Констркктор класса
    double getTime();                   //Возвращает время
    void useTransact(Transact& transact, std::ofstream& file); //Вывод сообщения
};
```

4. QUEUE с полями:

```
class QUEUE
{
private:
    std::list <Transact*> _transacts; //Список указателей на объекты Transact
public:
    QUEUE();                          //Конструктор
    int getSize();                    //Возвращает длину очереди
    void addTransact(Transact*);      //Добавляет в очередь указатель на новый объект
    Transact* popTransact();          //Возвращает 1 из очереди указатель
    bool hasTransact(Transact& transact); //Проверяет наличие транзакта в очереди

};
```

5. OPERATOR с полями:

```
class OPERATOR : public State
{
private:
    int _min;               //Минимальное число для генератора
    int _max;               //Максимальное число для генератора
    bool _isBusy;           //Флаг, показывает занят или нет оператор
    Transact* _served;      //Указатель на транзакт, который обслуживается
    QUEUE _queue;           //Очередь к оператору
public:
    OPERATOR(int min, int max); //Конструтор
    int getSize();              //Возвращаети длину очереди к оператору
    bool isBusy();              //Возвращает значение флага
    void useTransact(Transact& transact, std::ofstream& file); //Вывод сообщения
};
```

6. TERMINATE с полями:

```
class TERMINATE : public State
{
public:
    TERMINATE(); //Конструктор класса
    void useTransact(Transact& transact, std::ofstream& file); //Вывод сообщения
};
```

Определяем 2 оператора (2 объекта класса OPERATOR). Создаем список указателей на наши транзакты. Создаём последний транзакт, при котором завершим систему, далее создаём первый транзакт, вносим его в список, создавая при этом новый транзакт. Далее проходим по нашему списку с транзактами и ищем с наименьшем временем, если такой нашёлся меняем его состояние: 0 – транзакт включаем в нашу систему или при необходимости завершаем работу системы; 1 – выбираем оператор помещаем его либо в очередь, либо к оператору; 2 – обработка у 1 оператора; 3 - обработка у 2 оператора; 4 – обработка транзакта закончилась, удаляем его из нашей модели.

Результат работы программы:

```
В момент времни 8.79589 транзакт с индентификатором 1 вошел в модель.
В момент времени 8.79589 транзакт с идентификатором 1 занял устройство 1
В момент времни 13.5574 транзакт с индентификатором 2 вошел в модель.
В момент времени 13.5574 транзакт с идентификатором 2 занял устройство 2
В момент времни 18.756 транзакт с индентификатором 3 вошел в модель.
В момент времени 18.756 транзакт с идентификатором 3 встал в очередь 1
В момент времни 26.6404 транзакт с индентификатором 4 вошел в модель.
В момент времени 26.6404 транзакт с идентификатором 4 встал в очередь 2
В момент времни 26.7391 транзакт с индентификатором 2 освободил устройство 2
В момент времени 26.7391 транзакт с индентификатором 4 занял устройство 2
В момент времени 26.7391 тразакт с индентификатором 2 вышел из модели.
В момент времни 26.7481 транзакт с индентификатором 5 вошел в модель.
В момент времени 26.7481 транзакт с идентификатором 5 встал в очередь 2
В момент времни 28.7961 транзакт с индентификатором 6 вошел в модель.
В момент времени 28.7961 транзакт с идентификатором 6 встал в очередь 1
В момент времни 30.5526 транзакт с индентификатором 1 освободил устройство 1
В момент времени 30.5526 транзакт с индентификатором 3 занял устройство 1
В момент времени 30.5526 тразакт с индентификатором 1 вышел из модели.
В момент времни 32.6024 транзакт с индентификатором 7 вошел в модель.
В момент времени 32.6024 транзакт с идентификатором 7 встал в очередь 1
В момент времни 35.6744 транзакт с индентификатором 8 вошел в модель.
В момент времени 35.6744 транзакт с идентификатором 8 встал в очередь 2
В момент времни 37.8073 транзакт с индентификатором 4 освободил устройство 2
В момент времени 37.8073 транзакт с индентификатором 5 занял устройство 2
В момент времени 37.8073 тразакт с индентификатором 4 вышел из модели.
В момент времни 42.4645 транзакт с индентификатором 9 вошел в модель.
В момент времени 42.4645 транзакт с идентификатором 9 встал в очередь 2
В момент времни 55.9747 транзакт с индентификатором 3 освободил устройство 1
В момент времени 55.9747 транзакт с индентификатором 6 занял устройство 1
В момент времени 55.9747 тразакт с индентификатором 3 вышел из модели.
```

Текст программы:

```
class Transact
{
private:
    static int _currID;             //Идентификатор объекта класса
    int _ID;                        //Идентификатор экземпляра класса
    double _time;                   //Время следующей обработки
    int _state;                     //Номер текщего состояния
public:
    Transact();                         //Конструтор класса
    int getID();                        //Возвращает идентификатор
    double getTime();                   //Возвращает время
    int getState();                     //Возвращает текщее состяние
    void setTime(double time);          //Устанавливает время
    void setState(int state);           //Устанавливает номер текщего состояния
};
class State
{
protected:
    static int _currID;         //Идентификатор объекта класса
    int _ID;                    //Идентификатор экземпляра класса
    int _state;                 //Номер следющего состояния транзакта
public:
    State();                                //Конструтор класса
    int getID();                            //Возвращает идентификатор
    void setState(int state);               //Возвращает номер следующего состояния
    virtual void useTransact(Transact& transact, std::ofstream& file) = 0;  //Виртуальный метод
};
```

```cpp
class GENERATE : public State
{
private:
    double _time;                   //Время появления события
    int _max;                       //Максимальное число для генератора
    int _min;                       //Минимальное число для генератора
public:
    GENERATE(int min, int max);         //Констрктор класса
    double getTime();                   //Возвращает время
    void useTransact(Transact& transact, std::ofstream& file); //Вывод сообщения
};
class QUEUE
{
private:
    std::list <Transact*> _transacts; //Список указателей на объекты Transact
public:
    QUEUE();                            //Конструктор
    int getSize();                      //Возвращает длину очереди
    void addTransact(Transact*);        //Добавляет в очередь указатель на новый объект
    Transact* popTransact();            //Возвращает 1 из очереди указатель
    bool hasTransact(Transact& transact); //Проверяет наличие транзакта в очереди

};
class OPERATOR : public State
{
private:
    int _min;               //Минимальное число для генератора
    int _max;               //Максимальное число для генератора
    bool _isBusy;           //Флаг, показывает занят или нет оператор
    Transact* _served;      //Указатель на транзакт, который обслуживается
    QUEUE _queue;           //Очередь к оператору
public:
    OPERATOR(int min, int max); //Конструтор
    int getSize();              //Возвращаети длину очереди к оператору
    bool isBusy();              //Возвращает значение флага
    void useTransact(Transact& transact, std::ofstream& file); //Вывод сообщения
};
class TERMINATE : public State
{
public:
    TERMINATE(); //Конструктор класса
    void useTransact(Transact& transact, std::ofstream& file); //Вывод сообщения
};
```

```cpp
int Transact::_currID = 0;

Transact::Transact()
{
    _ID = _currID;
    _currID++;
    _state = 0;
    _time = 0;
}

int Transact::getID()
{
    return _ID;
}

double Transact::getTime()
{
    return _time;
}

int Transact::getState()
{
    return _state;
}

void Transact::setTime(double time)
{
    _time = time;
}

void Transact::setState(int state)
{
    _state = state;
}
```

```cpp
int State::_currID = 0;

State::State()
{
    _ID = _currID;
    _currID++;
    _state = 0;
}

int State::getID()
{
    return _ID;
}

void State::setState(int state)
{
    _state = state;
}
```

```cpp
int QUEUE::getSize()
{
    return _transacts.size();
}

void QUEUE::addTransact(Transact* transact)
{
    _transacts.push_back(transact);
}

Transact* QUEUE::popTransact()
{
    Transact* element = _transacts.front();
    _transacts.pop_front();
    return element;
}

bool QUEUE::hasTransact(Transact& transact)
{
    std::list <Transact*> ::iterator iter = _transacts.begin();
    for (; iter != _transacts.end(); iter++)
    {
        if ((**iter).getID() == transact.getID())
            return true;
    }
    return false;
}
```

```cpp
GENERATE::GENERATE(int min, int max):State()
{
    _time = 0;
    _min = min;
    _max = max;
}

double GENERATE::getTime()
{
    _time += (double)(_min + (rand() % (_max - 1)) + (double)rand() / RAND_MAX);
    return _time;
}

void GENERATE::useTransact(Transact& transact, std::ofstream& file)
{
    transact.setState(_state);
    file << "В момент времни " << transact.getTime() << " транзакт с индентификатором " << transact.getID() << " вошел в модель." << endl;
}
```

```cpp
OPERATOR::OPERATOR(int min, int max) : State(), _queue()
{
    _served = NULL;
    _isBusy = false;
    _min = min;
    _max = max;
}

int OPERATOR::getSize()
{
    return _queue.getSize();
}

bool OPERATOR::isBusy()
{
    return _isBusy;
}

void OPERATOR::useTransact(Transact& transact, std::ofstream& file)
{
    if (!_isBusy)
    {
        file << "В момент времени " << transact.getTime() << " транзакт с идентификатором " << transact.getID() << " занял устройство " << _ID << endl;
        transact.setTime(transact.getTime() + (double)(_min + (rand() % (_max - 1)) + (double)rand() / RAND_MAX));
        _served = &transact;
        _isBusy = true;
    }
    else if (&transact == _served)
    {
        file << "В момент времни " << transact.getTime() << " транзакт с индентификатором " << transact.getID() << " освободил устройство " << _ID << endl;
        transact.setState(_state);
        if (_queue.getSize() > 0)
        {
            Transact* next = _queue.popTransact();
            file << "В момент времени " << next->getTime() << " транзакт с индентификатором " << next->getID() << " занял устройство " << _ID << endl;
            next->setTime(next->getTime() + (double)(_min + ((unsigned long long int)rand() % ((unsigned long long int)_max - 1)) + (double)rand() / RAND_MAX));
            _served = next;
        }
        else
        {
            _isBusy = false;
        }
    }
    else
    {
        if (!_queue.hasTransact(transact))
        {
            _queue.addTransact(&transact);
            file << "В момент времени " << transact.getTime() << " транзакт с идентификатором " << transact.getID() << " встал в очередь " << _ID << endl;
        }
        transact.setTime(_served->getTime());
    }
}
```

```cpp
void TERMINATE::useTransact(Transact& transact, std::ofstream& file)
{
    file << "В момент времени " << transact.getTime() << " тразакт с индентификатором " << transact.getID() << " вышел из модели." << endl;
    delete &transact;
}
```

```cpp
int main()
{
    int r1 = 8;
    int g1 = 7;
    int b1 = 5;
    int rnd = 20;
    int endTime = 60;

    std::ofstream file;
    file.open("result.txt");
    if (!file.is_open())
    {
        std::cout << "Cannot use file" << std::endl;
        return 1;
    }
    srand(rnd);

    GENERATE generate(0, r1 + g1 + b1);
    generate.setState(1);

    OPERATOR* operators[2];
    operators[0] = new OPERATOR(r1, r1 + g1 + b1);
    operators[0]->setState(4);
    operators[1] = new OPERATOR(g1, r1 + g1 + b1);
    operators[1]->setState(4);

    TERMINATE Terminator;

    list <Transact*> transacts;
    Transact* finish = new Transact();
    finish->setState(0);
    finish->setTime(endTime);
    transacts.push_front(finish);
    Transact* start = new Transact();
    start->setState(0);
    start->setTime(generate.getTime());
    transacts.push_front(start);
    list <Transact*> ::iterator iter;
bool isFinish = false;
while (!isFinish)
{
    double tMin = (*transacts.begin())->getTime();
    for (iter = transacts.begin(); iter != transacts.end(); ++iter)
    {
        if (tMin > (*iter)->getTime())
            tMin = (*iter)->getTime();
    }
    for (iter = transacts.begin(); iter != transacts.end(); ++iter)
    {
        if ((*iter)->getTime() == tMin)
        {
            bool flag;
            switch ((*iter)->getState())
            {
            case 0:
                if ((**iter).getID() == (*finish).getID())
                {
                    isFinish = true;
                }
                else
                {
                    generate.useTransact(**iter, file);
                    Transact* newTrans = new Transact();
                    newTrans->setTime(generate.getTime());
                    transacts.push_back(newTrans);
                }
                break;
```

```
                case 1:
                    flag = false;
                    for (int i = 0; i < sizeof(operators) / sizeof(*operators); ++i) {
                        if (!(operators[i]->isBusy())) {
                            (*iter)->setState(i + 2);
                            operators[i]->useTransact(**iter, file);
                            flag = true;
                            break;
                        }
                    }
                    if (!flag)
                    {
                        int min = operators[0]->getSize();
                        int minIndex = 0;
                        for (int i = 0; i < sizeof(operators) / sizeof(*operators); ++i)
                        {
                            if (operators[i]->getSize() < min) {
                                minIndex = i;
                                min = operators[i]->getSize();
                            }
                        }
                        (*iter)->setState(minIndex + 2);
                        operators[minIndex]->useTransact(**iter, file);
                    }
                    break;
                case 2:
                    operators[0]->useTransact(**iter, file);
                    break;
                case 3:
                    operators[1]->useTransact(**iter, file);
                    break;
                case 4:
                    Terminator.useTransact(**iter, file);
                    iter = transacts.erase(iter);
                    break;
                }
            }
        }
    }
    file.close();
    return 0;
}
```