МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
**«Сибирский государственный аэрокосмический университет
имени академика М.Ф. Решетнева»
(СибГАУ)**

В. С. Тынченко

Имитационное моделирование в среде GPSS World

*Методические указания к лабораторным работам
по курсу «Теория массового обслуживания» для студентов, обучающихся
по направлению 09.03.01 «Информатика и вычислительная техника»,
направленность (профиль) подготовки «Автоматизированные системы обработки информации и управления» всех форм обучения*

Красноярск 2015

Содержание

Введение

Программа GPSS (*General Purpose Systems Simulator*) является классической программой имитационного моделирования.

GPSS предназначена для моделирования систем массового обслуживания (систем с очередями) а также другх аналогичных систем, и имеет для этих целей специальные операторы, синтаксис, вспомогательные инструменты (статистическая обработка результатов, их накопление, графическое отображение).

GPSS была создана в 1961 году в фирме IBM *Джеффри Гордоном* (*Geoffrey Gordon*) почти в то же время, что и языки FORTRAN и ALGOL. Наряду с ними GPSS имелась на всех компьютерах того времени. Фирма *Minuteman Software* в 1982 году адаптировала программу к персональным компьютерам для работы в DOS (версия GPSS/PC), в 1993 году создала первую версию GPSS World для OS/2, а в 2000 году — усовершенствованную версию программы GPSS World для Windows (может применяться с Windows 98, 2000 и XP).

В настоящее время разработана версия GPSS World 5.0.3, которая поставляется в трёх вариантах:

GPSS World Commercial Version 5.0.3 — неограниченное количество блоков (стоимость — по договорённости с фирмой);

GPSS World Personal Version 5.0.3 — 2000 блоков ($695);

GPSS World Student Version 5.0.3 — 180 блоков ($9,95).

В России с декабря 2004 года распространяется версия GPSS World 5.0.2 также в трёх вариантах:

Однопользовательская лицензия студенческой версии GPSS World 5.0.2 — 500 блоков (1 500 руб.);

Однопользовательская лицензия персональной версии GPSS World 5.0.2 — 5000 блоков (23 000 руб.);

Однопользовательская лицензия академической версии GPSS World 5.0.2 — количество блоков не ограничено, поставляется для вузов (35 000 руб.);

Однопользовательская лицензия коммерческой версии GPSS World 5.0.2 — количество блоков не ограничено, поставляется для коммерческих фирм (120 000 руб.).

Эксплуатационные документы можно купить отдельно за 700 руб. плюс почтовые расходы.

Фирма *Minuteman Software* предлагает также бесплатный студенческий вариант программы GPSS World 4.3.5, который обладает всеми функциями соответствующей полной версии, но имеет ограничение на максимальное количество блоков (не более 150). Эта версия практически не отличается от последней продаваемой в России версии GPSS World 5.0.2 (исправлены некоторые мелкие недочёты). Таким образом, студенческая бесплатная версия GPSS

World 4.3.5 позволяет изучить все особенности программы, но не может быть использована для коммерческих целей при разработке больших имитационных моделей. Эксплуатационные документы также не очень нужны, так как вместе с версией GPSS World 4.3.5 предлагается бесплатный электронный вариант книг *Reference Manual* (справочник по системе с перечислением всех операторов, правил синтаксиса и встроенных инструментов), а также *Tutorial Manual* (примеры использования программы с соответствующими файлами, включенными в инсталляционный файл GPSS World 4.3.5).

Материалы о программе GPSS World размещены на сайтах:

`http://www.minutemansoftware.com/` — оригинальный сайт фирмы-разработчика;

`http://www.gpss.ru/index-h.html` — российский сайт по имитационному моделированию.

1 Установка и работа программы GPSS World

1.1 Установка студенческой версии программы GPSS World

Программа GPSS устанавливается сама после запуска инсталляционного файла, который в оригинале называется **student.exe**. Программа размещает себя в каталоге на диске **C:\Program Files\Minuteman Software** в каталоге, носящем название фирмы-разработчика программы **Minuteman Software**. В этом каталоге размещаются каталоги более низкого уровня: **GPSS World Student Version** → **Sample Models**.

В каталоге **GPSS World Student Version** находится сама программа **GPSSW.exe**, а также вспомогательные файлы.

В каталоге **Sample Models** находятся образцы программных моделей на GPSS, которые можно запустить и на их примере познакомиться с работой программы GPSS.

К программе прилагаются две электронные фирменные книги на английском языке:

Reference Manuel — справочник по GPSS, в котором приводятся сведения об операторах GPSS и её функционировании;

Tutorial Manual — описание примеров программных моделей с текстами на GPSS и пояснениями.

При первом запуске программы GPSS появляется главное окно программы (Рисунок 1.1):

Рисунок 1.1 — Главное окно программы GPSS после её вызова

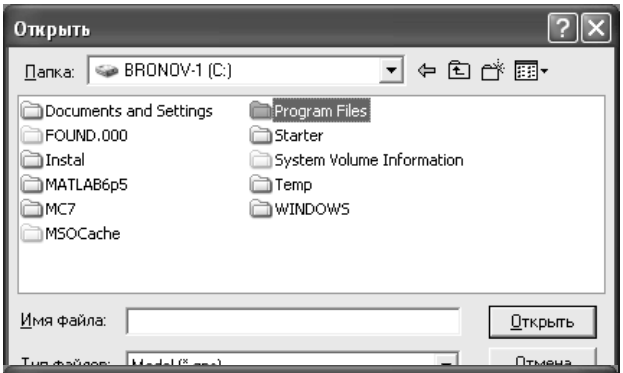
Далее можно вызвать готовую программную модель или создать новую.

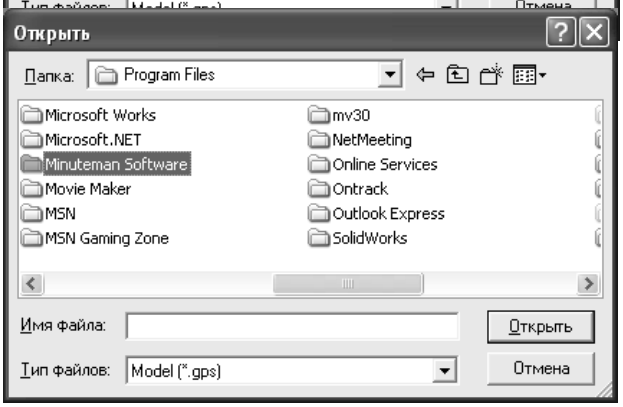
Вместе с программой GPSS поставляется набор готовых программных моделей, иллюстрирующих возможности GPSS. Эти модели размещены в каталоге **Sample Models** и подробно рассмотрены в электронной книге *Tutorial Manual*, хранящемся в файле **GPSSWorld-Tutorial_Manual.doc.**

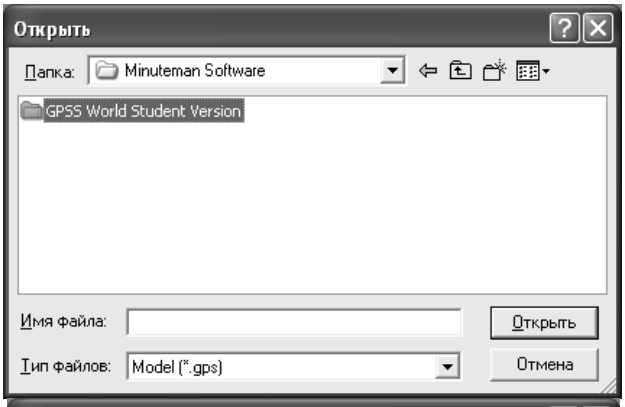
Вызов примера готовой программной модели описан ниже.

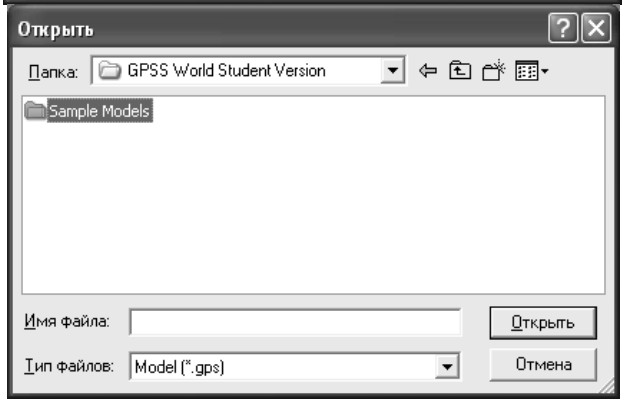
1.2 Вызов готовой программной модели

1 Через меню **File** → **Open** перейти в подменю выбора пути к файлу и дойти до каталога примеров программных моделей (Рисунок 1.2).


Открыть
Папка: BRONOV-1 (C:)
Documents and Settings
FOUND.000
Instal
MATLAB6p5
MC7
MSOCache
Program Files
Starter
System Volume Information
Temp
WINDOWS
Имя файла:
Открыть


Открыть
Папка: Program Files
Microsoft Works
Microsoft.NET
Minuteman Software
Movie Maker
MSN
MSN Gaming Zone
mv30
NetMeeting
Online Services
Ontrack
Outlook Express
SolidWorks
Имя файла:
Открыть
Тип файлов: Model (*.gps)
Отмена


Открыть
Папка: Minuteman Software
GPSS World Student Version
Имя файла:
Открыть
Тип файлов: Model (*.gps)
Отмена


Открыть
Папка: GPSS World Student Version
Sample Models
Имя файла:
Открыть
Тип файлов: Model (*.gps)
Отмена

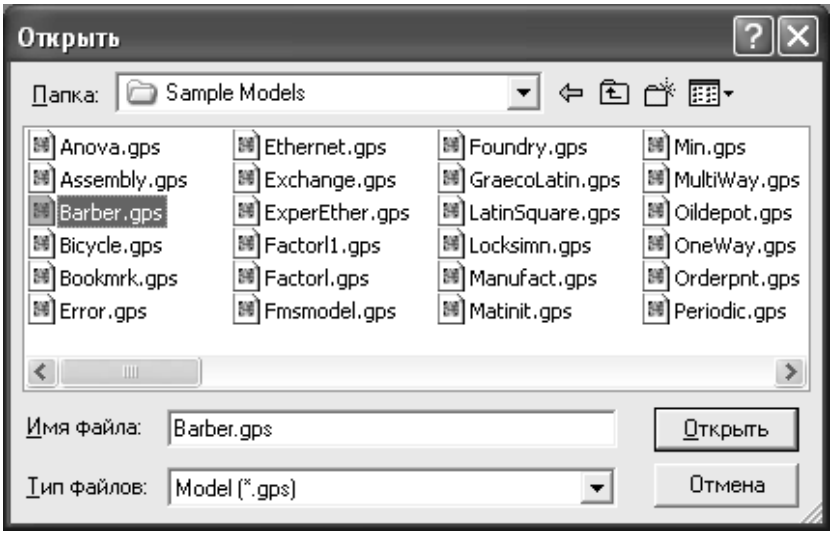


Рисунок 1.2 — Выбор каталога с примерами программных моделей на GPSS

2 Выбрать в качестве примера программной модели файл с именем **Barber.gps**, моделирующий работу парикмахера. На экране появится текст модели на языке GPSS (Рисунок 1.3).

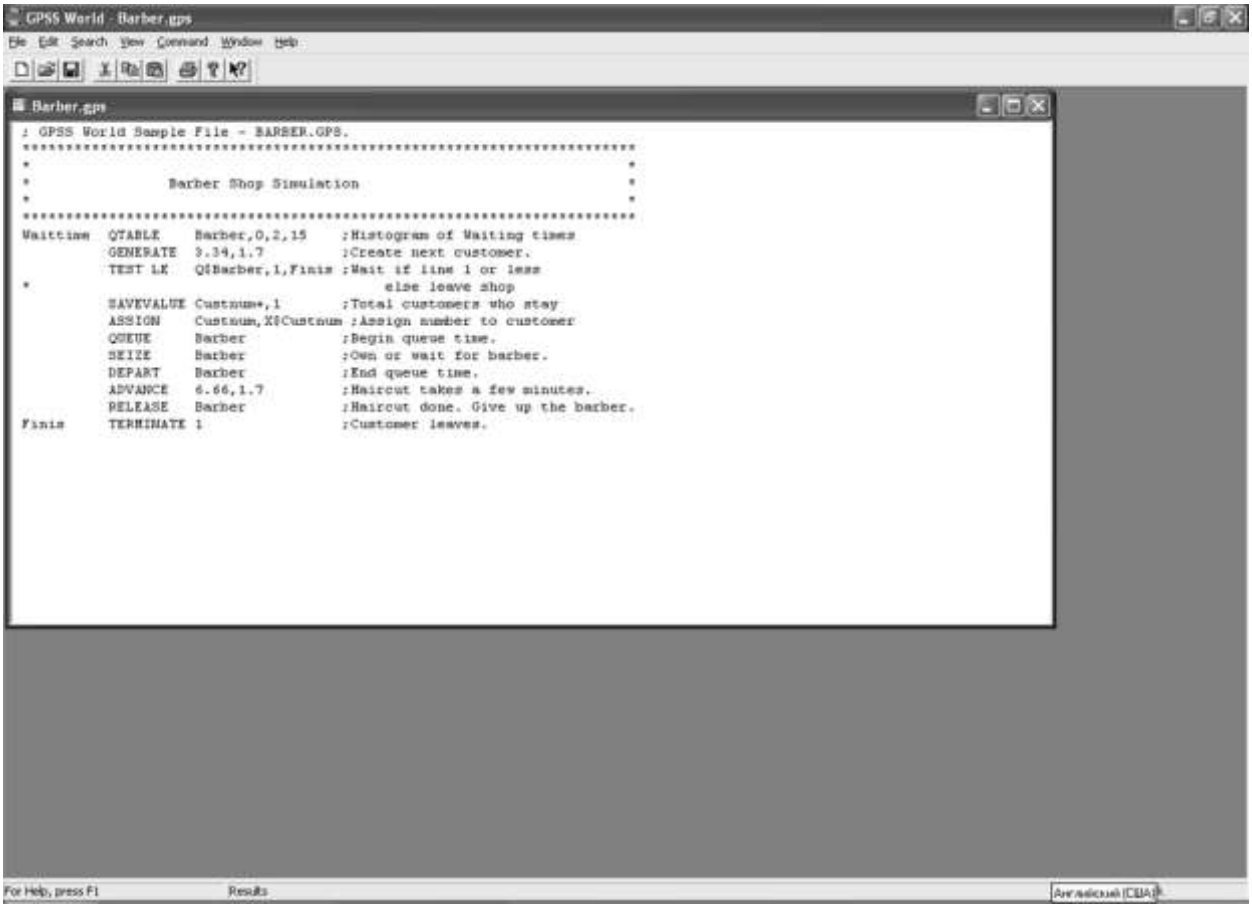


Рисунок 1.3 — Вызов готовой модели **Barber.gps**

3 Эту модель необходимо предварительно оттранслировать и лишь затем запустить на выполнение. Трансляцию осуществляют через меню

Command → Create Simulation. Эта команда — единственная, которая в данной ситуации после запуска GPSS доступна в меню **Command**, так как без трансляции модели другие операции невозможны.

В результате появляется ещё одно окно с названием JOURNAL — журнал регистрации процесса моделирования (Рисунок 1.4).

В этом журнале будут регистрироваться все задания программе, результаты её работы, возникающие ошибки. Сами значения результатов будут накапливаться в других окнах, а в окне JOURNAL будет лишь указание на нормальное или ненормальное завершение работы.

Многие предыдущие варианты программы GPSS были интерпретирующими и поэтому работали медленно, но GPSS World является программой транслирующей и поэтому работает существенно быстрее.

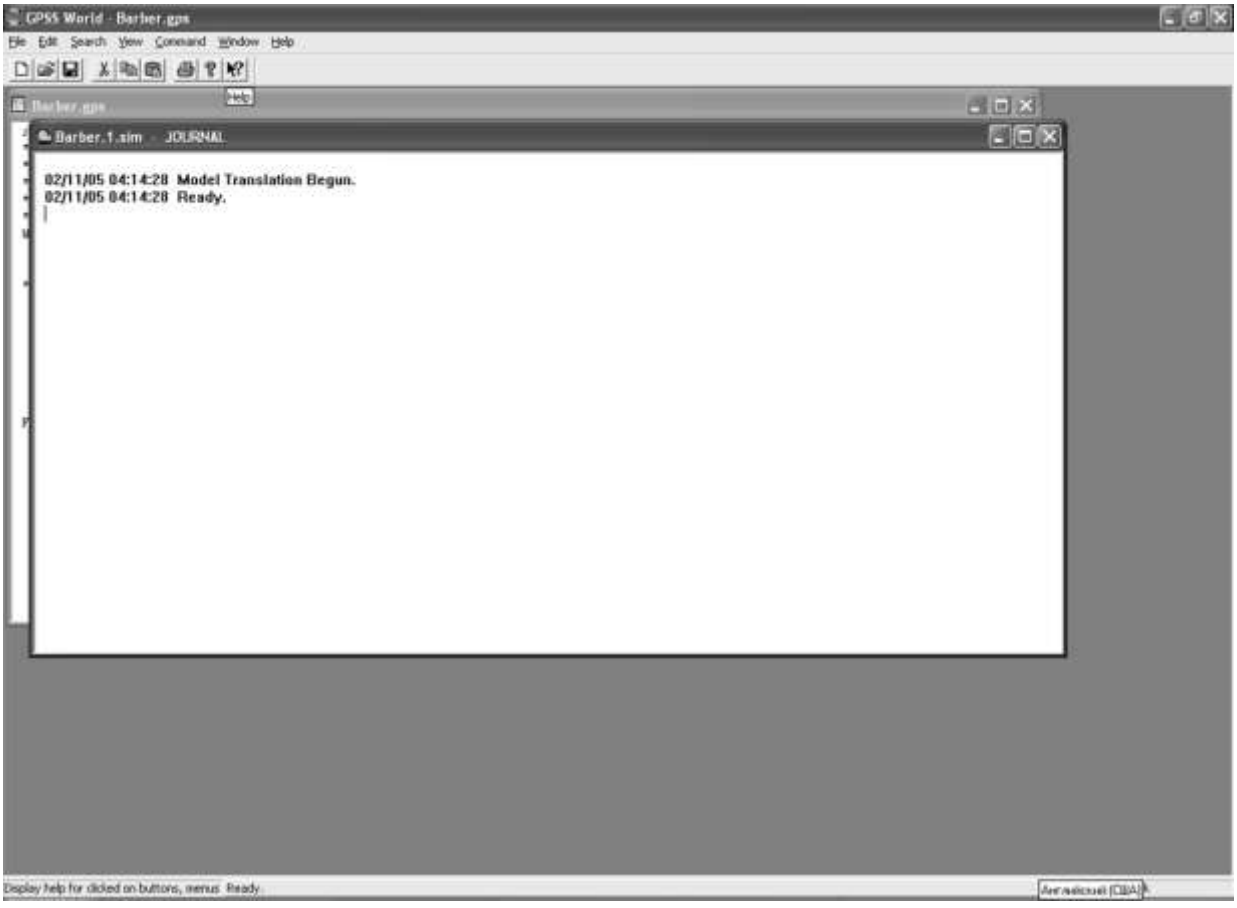


Рисунок 1.4 — Журнал регистрации процесса моделирования

Теперь можно запустить программу на счёт и посмотреть результаты. Но предварительно необходимо разобраться, что именно она моделирует.

1.3 Пояснения к тексту примера программной модели

Текст программной модели приведён ниже (Рисунок 1.5).

```
; GPSS World Sample File - BARBER.GPS.
************************************************************************
```

```
*                                                                     *
*                Barber Shop Simulation                               *
*                                                                     *
***********************************************************************
Waittime  QTABLE    Barber,0,2,15    ;Histogram of Waiting times
          GENERATE  3.34,1.7         ;Create next customer.
          TEST LE   Q$Barber,1,Finis ;Wait if line 1 or less
*                                         else leave shop
          SAVEVALUE Custnum+,1       ;Total customers who stay
          ASSIGN    Custnum,X$Custnum ;Assign number to customer
          QUEUE     Barber           ;Begin queue time.
          SEIZE     Barber           ;Own or wait for barber.
          DEPART    Barber           ;End queue time.
          ADVANCE   6.66,1.7         ;Haircut takes a few minutes.
          RELEASE   Barber           ;Haircut done. Give up the barber.
Finis     TERMINATE 1                ;Customer leaves.
```

Рисунок 1.5 — Текст примера программной модели **Barber.gps**

Имеется парикмахерская с одним креслом парикмахера. Через некоторые промежутки времени появляются клиенты, чтобы постричься. Если кресло пустое, то парикмахер начинает обслуживать очередного клиента. Если кресло занято, то клиент встаёт в очередь. Модель позволяет определить размер очереди (средний за время моделирования, максимальный и минимальный) и некоторые другие параметры.

Текст программы располагается по строкам — каждый оператор в собственной строке. Могут быть пустые строки, которые игнорируются. Данная программа начинается с комментариев. Строки с комментариями помечаются звёздочками "*" (в первом слева столбце) или точкой с запятой ";" (в любом месте строки). Содержание комментариев может быть любым, в том числе на русском языке, так как они игнорируются транслятором.

Первая строка с оператором имеет вид:

```
Waittime  QTABLE    Barber,0,2,15    ;Histogram of Waiting times
```

Слово **Waittime** является меткой оператора **QTABLE**. Обычно метки служат для организации перехода из других мест программы, если это требуется. Метка располагается в левых крайних позициях строки перед оператором. Но в данном случае метка привязана к таблице, которая служит для построения гистограммы. Оператор **QTABLE** означает "таблица очередей": первая буква **Q** — начальная буква английского слова queue = очередь (читается "кью"). Этот оператор применяется для фиксации данных об очередях. В данном случае речь идёт об очереди клиентов перед занятием ими кресла парикмахера. Таким образом метка превращается в имя таблицы для построения гистограммы после окончания моделирования (гистограммой называется специальный график, отображающий некоторые статистические параметры). Слово **Waittime** выбрано произвольно, а слово **QTABLE** — является оператором GPSS. Сочетание **Barber,0,2,15** является набором операндов оператора **QTABLE**. С точки зрения обычного языка программирования это примерно эквивалентно записи **QTABLE(Barber,0,2,15)**. Но в GPSS операнды отделяются от оператора пробелами и разделяются между собой запя-

тыми. Если какой-то операнд отсутствует, вместо него ставится запятая (могут быть две и более запятые подряд). Если отсутствуют все последние операнды, то их можно не помечать.

Операнд **Barber** означает имя очереди (задаётся произвольно и присутствует ниже), величину которой необходимо записывать в таблицу. Число **0** означает верхнюю границу первого частотного класса, число **2** означает размер частотного интервала — разницу между верхней и нижней границей каждого частотного класса, число **15** означает число частотных интервалов. Все эти параметры относятся к параметрам гистограмм.

Вторая строка программы:

```
        GENERATE  3.34,1.7          ;Create next customer.
```

Здесь нет метки. Слово **GENERATE** является оператором GPSS и используется для генерации (создания) транзактов. В разных случаях транзакты могут иметь разную природу. В данном случае это — клиенты парикмахерской. В других случаях транзакты могут представлять из себя детали для сборки, документы, минуты и др. У оператора **GENERATE** имеется два числовых операнда: **3.34** и **1.7**, разделённых запятой. Первое число означает интервал времени, через который появляется очередной транзакт (в данном случае через каждые 3,34 единицы времени). Чему равна единица времени, определяется разработчик модели, когда задаёт время в разных местах программы. В GPSS можно говорить лишь о некоторых временны□х пропорциях. Число **1.7** означает разброс времени. Поэтому данный оператор имитирует появления нового клиента через каждые 3,34±1,7 единицы времени. В этом интервале (1,64…5,04) единицы времени принято равномерное распределение.

Следующая строка программы:

```
        TEST LE   Q$Barber,1,Finis ;Wait if line 1 or less
*                                    else leave shop
```

Она содержит оператор **TEST**, аналогичный оператору **IF** в обычных языках программирования. Этот оператор тестирует (проверяет) выполнение условия и осуществляет переход на какую-либо метку (или без перехода на метку). Стоящий справа вспомогательный оператор **LE** как раз и означает, какой именно контроль осуществляет основной оператор **TEST**: **LE** означает "меньше или равно". Могут быть и другие вспомогательные операторы: **E** "равно", **L** "меньше", **G** "больше", **NE** "не равно", **LE** "меньше или равно", **GE** "больше или равно"). Сравниваются между собой два первых операнда, в данном случае: **Q$Barber** $\leq$ **1** ? Если условие выполняется, то осуществляется переход на следующий (нижестоящий) оператор. Если условие не выполняется, то осуществляется переход на метку, заданную в третьем операнде: **Finis**. Переменная **Q$Barber** имеет значение количества человек в очереди. Тогда смысл этого оператора **TEST** заключается в том, что просматривается длина очереди. Если есть очередь и в очереди более 1 человека, то

новый клиент не задерживается, а покидает парикмахерскую. Для этого и сделан переход на метку **Finis** в конце программы. Если же очереди нет или в ней всего лишь 1 человек, то клиент занимает очередь. Поэтому в данном случае очередь не может быть больше 2 человек.

Здесь можно варьировать максимально допустимую длину очереди и смотреть, сколько же всего может набраться в ней человек.

Следующая строка программы:

```
SAVEVALUE Custnum+,1       ;Total customers who stay
```

Оператор **SAVEVALUE** означает *Сохраняемая величина*. Имя этой переменной (любое) задаётся первым операндом: **Custnum**. Знак "**+**" не относится к имени переменной. В сравнении с обычными языками программирования это — обычная переменная, используемая как счётчик. Знак "**+**" означает, что к ней прибавляется следующий операнд (в данном случае 1). Можно использовать также знак "**–**" и тогда второй операнд будет отниматься от первого. Если не будет ни "**+**", ни "**–**", то произойдёт присваивание значения второго операнда первому. Второй операнд может быть как числом, так и переменной (а также строкой). С помощью переменной **Custnum** в данном случае подсчитывается число клиентов, посетивших парикмахерскую и оставшихся в очереди.

Следующая строка программы:

```
ASSIGN    Custnum,X$Custnum ;Assign number to customer
```

Оператор **ASSIGN** означает присвоение значения второго операнда первому. Второй операнд в данном случае имеет то же имя, что переменная в предыдущем операторе, а именно **X$Custnum**, но имя **Custnum** стои□т после специфического сочетания знаков **X$**. В данном случае имеется пример так называемых *Системных числовых атрибутов* (СЧА) — *Системным числовым атрибутом* является сочетание знаков **X$**. Смысл СЧА будет рассмотрен ниже, а пока достаточно принять, что выражение **X$Custnum** означает "содержимое переменной **Custnum**", которое ранее было подсчитано.

Смысл последних двух операторов следующий. Когда оператор **GENERATE** создаёт очередной транзакт, он присваивает ему очередной номер по порядку. Некоторые из транзактов (клиентов) покидают парикмахерскую из-за очереди. Последние два оператора занимаются тем, что присваивают транзактам новые номера (перенумеровывают их). Новый порядок номеров относится только к тем транзактам, которые решились встать в очередь. Например, пусть генерируются транзакты с номерами 1, 2, 3, 4, 5, 6, 7, 8, 9 и т. д. И пусть транзакты 2, 5 и 8 "испугались" очереди и покинули парикмахерскую. Тогда оставшиеся транзакты будут перенумерованы этими двумя операторами следующим образом:

Прежний номер транзакта	1	**2**	3	4	**5**	6	7	**8**	9	…
Новый номер транзакта	1		2	3		4	5		6	…

Жирным шрифтом выделены покинувшие парикмахерскую транзакты (клиенты).

Для чего нужно делать такую перенумерацию? В данном случае — это просто учебная демонстрация работы конкретных операторов. В некоторых практических случаях это может оказаться необходимым. Например, если каждому оставшемуся (и обслуженному впоследствии) транзакту выдаётся чек за оплаченную им услугу и номер чека соответствует номеру транзакта. Или когда требуется перенумеровать очередь. Возможны и другие реальные случаи (кто первый? кто последний?).

Следующая строка программы:

```
QUEUE     Barber          ;Begin queue time.
```

Здесь оператор **QUEUE** означает *Очередь*, а операнд **Barber** — её имя (в данном случае "парикмахер"). Работа этого оператора заключается в фиксации транзакта, который направляется к креслу парикмахера. Оператор **QUEUE** "знает", свободно ли кресло. Если оно свободно, то оператор **QUEUE** пропускает транзакт через себя без задержки. Если же кресло занято, то оператор **QUEUE** задерживает очередной транзакт и создаёт очередь. В функции оператора **QUEUE** входит также фиксировать время, когда очередной транзакт вошёл в очередь, а также время выхода его из очереди, время нахождения в очереди и др. параметры. Все эти данные можно использовать в программе, обращаясь к переменной **Barber** с помощью *Системных числовых атрибутов*.

Следующая строка программы:

```
SEIZE     Barber          ;Own or wait for barber.
```

Оператор **SEIZE** означает "захват транзактом *Обслуживающего устройства*", в данном случае — устройства по имени **Barber** "парикмахер". Операнд **Barber** является именем *Обслуживающего устройства*. В данном случае это имя совпадает с именем другого (предыдущего) блока — *Очереди* **QUEUE**. Это — именно совпадение. Имена *Очереди* и *Обслуживающего устройства* могут быть одинаковыми или разными — это не имеет значения (можно проверить в программе). Программа понимает, что если имя используется с оператором **SEIZE**, то речь может идти только об *Обслуживающем устройстве*. Захват возможен только в том случае, если *Обслуживающее устройство* свободно. Именно сигнал от оператора **SEIZE** учитывается оператором **QUEUE** при постановке транзакта в очередь. Если *Обслуживающее устройство* занято, то транзакт остаётся в очереди. Если *Обслуживающее устройство* освободилось, то транзакт перемещается в него и программа переходит к следующему оператору:

```
DEPART    Barber          ;End queue time.
```

Оператор **DEPART** означает "освободить *Очередь* с именем **Barber** на 1 транзакт".

Следующая строка программы:

```
ADVANCE   6.66,1.7        ;Haircut takes a few minutes.
```

Оператором **ADVANCE** задаётся время обработки в том *Обслуживающем устройстве*, имя которого было в ближайшем (предыдущем) операторе **SEIZE** (в данном случае это устройство с именем **Barber**). Первый и второй операнды аналогичны таковым у оператора **GENERATE**, т. е. означают, что время обработки составляет 6,66±1,7 единиц времени (с равномерным распределением в заданном диапазоне). Когда оператор **SEIZE** "впихивает" транзакт в *Обслуживающее устройство*, начинается отсчёт времени. Когда время заканчивается (через 6,66±1,7 единиц), транзакт "выпихивается" из *Обслуживающего устройства*:

```
RELEASE   Barber          ;Haircut done. Give up the barber.
```

Оператор **RELEASE** осуществляет это "выпихивание" из устройства с именем, заданным в его операнде.

Последний оператор программы:

```
Finis     TERMINATE 1               ;Customer leaves.
```

Здесь **Finis** — метка (на неё осуществляется переход оператором **TEST LE**). Оператор **TERMINATE** означает, что при его выполнении уничтожается 1 транзакт. "Уничтожение" транзакта означает, что он покидает систему (модель), т. е. парикмахерскую. На этот оператор выводят два пути: "испугавшиеся очереди" транзакты попадают на выход через оператор **TEST LE**, а оставшиеся и дождавшиеся обслуживания — естественным путём после последовательного перемещения через все операторы.

Анализируя эти фрагменты, можно заметить, что существуют пары операторов:

QUEUE "включить в *Очередь*" — **DEPART** "освободить *Очередь*"

SEIZE "захватить *Обслуживающее устройство*" — **RELEASE** "освободить *Обслуживающее устройство*".

Именно такая жёсткая парность этих операторов позволяет GPSS не перепутать операции, даже если *Очередь* и *Обслуживающее устройство* имеют одинаковое (синонимичное) название. Одинаковые названия бывают полезны в больших моделях для их лучшей читаемости.

1.4 Запуск программы на счёт

Для запуска на счёт выполняется следующая последовательность действий (Рисунок 1.6) через меню **Command → START**

Рисунок 1.6 — Задание продолжительности работы программы числом транзактов

В данном окне необходимо ввести количество транзактов, которые должны пройти через систему до окончания счёта. По умолчанию стоит 1, но может потребоваться большее число транзактов. В данном случае можно ввести 100. Слово **START** является командой GPSS и его можно стереть и заново ввести в этом окне. Его можно набить и в конце самой программы, указав там число задаваемых транзактов. Тогда программа после трансляции сразу начнёт считать. Использование окна позволяет постепенно просматривать работу системы, задавая порциями число вводимых транзактов. После окончания начавшегося счёта появится новое окно поверх прежних (Рисунок 1.7):

Это окно представляет собой так называемый *Стандартный отчёт*, в котором показываются конечные результаты моделирования. Результатами являются различные данные, связанные с *Очередью*, *Обслуживающим устройством*, транзактами.

```
              GPSS World Simulation Report - Barber.1.1

                    Friday, February 11, 2005 06:40:58
           START TIME           END TIME  BLOCKS  FACILITIES  STORAGES
                0.000            353.895    10        1          0

              NAME                      VALUE
          BARBER                     10001.000
          CUSTNUM                    10002.000
          FINIS                         10.000
          WAITTIME                   10000.000

 LABEL              LOC  BLOCK TYPE     ENTRY COUNT CURRENT COUNT RETRY
                    1    GENERATE           102             0       0
                    2    TEST               102             0       0
                    3    SAVEVALUE           55             0       0
                    4    ASSIGN              55             0       0
                    5    QUEUE               55             1       0
                    6    SEIZE               54             1       0
                    7    DEPART              53             0       0
                    8    ADVANCE             53             0       0
                    9    RELEASE             53             0       0
FINIS               10    TERMINATE          100             0       0

FACILITY          ENTRIES  UTIL.   AVE. TIME AVAIL. OWNER PEND INTER RETRY DELAY
 BARBER              54    0.987       6.470  1        98    0     0     0      1

QUEUE               MAX CONT. ENTRY ENTRY(0) AVE.CONT. AVE.TIME   AVE.(-0) RETRY
 BARBER               2    2     55      1     1.652     10.628     10.824   0

TABLE               MEAN    STD.DEV.       RANGE            RETRY FREQUENCY CUM.%
 WAITTIME         10.709    2.702                             0
                                          _      -        0.000          1     1.89
                                      0.000      -        2.000          0     1.89
                                      2.000      -        4.000          1     3.77
                                      4.000      -        6.000          0     3.77
                                      6.000      -        8.000          4    11.32
                                      8.000      -       10.000         12    33.96
                                     10.000      -       12.000         17    66.04
                                     12.000      -       14.000         14    92.45
                                     14.000      -       16.000          4   100.00

SAVEVALUE               RETRY       VALUE
 CUSTNUM                  0         55.000

CEC XN    PRI          M1       ASSEM  CURRENT  NEXT  PARAMETER    VALUE
    98     0         341.236      98      6      7

                                                    CUSTNUM      54.000
FEC XN    PRI          BDT      ASSEM  CURRENT  NEXT  PARAMETER    VALUE
   103     0         356.553     103      0      1
```

Рисунок 1.7 — Окно *Стандартного отчёта*

Вверху — некоторые данные о моделировании: имя файла (**Barber.1.1**), дата и время моделирования, время старта (**START TIME** — в данном случае **0.000** единицы), время окончания счёта (**END TIME** — в данном случае **353.895** *единицы времени*), общее число блоков (**BLOCKS** — в данном случае **10**), число обслуживающих устройств (**FACILITIES** — в данном случае **1**, парикмахер), число многоканальных накопителей (**STORAGES** — данном случае их нет). Ниже располагаются имена переменных (и меток) и их номера, присвоенные им программой при трансляции. Это — вспомогательная информация.

В центре отчёта имеется текст программы (без комментариев). Текст программы расположен в виде таблицы (без прочерченных строк и столбцов). Самая левая колонка — **LABEL** (метка). Единственная метка в программе — **FINIS**.

Название таблицы **WAITTIME** не является меткой и здесь не отражено. Колонка **LOC** содержит номера блоков, присвоенные им программой. В колонке **BLOCK TYPE** располагаются имена блоков. В колонке **ENTRY COUNT** помещено количество "вошедших" транзактов, а в колонке **CURRENT COUNT** — оставшихся в них на данный момент. Видно, что на момент остановки программы в системе осталось два транзакта — один сидит в кресле парикмахера (оператор **SEIZE**) и один — в очереди (оператор **QUEUE**). Колонка **RETRY** содержит информацию о числе транзактов, которые пытались дважды (и более раз) войти в эти блоки (неудачи могли быть связаны, например, если бы их обошли транзакты с более высоким приоритетом). Таковых в данном случае не было.

Ниже идёт информация об обслуживающих устройствах (в GPSS они называются **FACILITY**). В данном случае имеется только одно обслуживающее устройство (был один оператор **SEIZE**) по имени **BARBER**. По нему выдаётся следующая информация:

ENTRIES — число входов в обслуживающее устройство, равно 54 (столько обслужено транзактов);

UTIL — коэффициент использования (отношение времени работы к полному времени моделирования), составил 0,987 (максимально возможное число 1,000; вообще-то парикмахер работал всё время, так как к нему непрерывно была очередь, и поэтому коэффициент использования должен был бы быть равным 1,000, но в самом начале работы он немного подождал первого клиента и из-за этого коэффициент использования оказался всё же меньше 1,000);

AVE. TIME — среднее время обслуживания, оказалось равным 6,470 *ед. времени*;

AVAIL. — доступность обслуживающего устройства для очередных транзактов на момент окончания счёта характеризуется как "доступно" (стоит 1, если бы был 0, то было бы "недоступно"). Доступность или недоступность задаются программно: некоторые из устройств могут быть "выключенными" из работы (например, когда имитируется поломка);

OWNER — порядковый номер транзакта, обслуживаемого в данный момент, равен 98. Транзакты различаются в основном номерами (это их имена) и к транзактам обращаются по номерам;

PEND — число транзактов, ожидающих поступления на обслуживание и помещённых в специальную *Цепь ожидания*, равно 0 (их нет);

INTER — число транзактов, движение которых было принудительно прервано, равно 0 (их нет);

DELAY — число транзактов, ожидающих обслуживания и имеющих для этого необходимый приоритет, равно 1 (имеется 1 транзакт, очевидно, находящийся сейчас в очереди). Приоритет присваивается транзакту по сравнению с другими транзактами, и имеющие более высокий приоритет обслуживаются раньше других (вне очереди). Обычно изначально у всех транзактов одинаковый приоритет и они обслуживаются по очереди. Но приоритет можно изменять программно.

Аналогичные параметры рассчитаны и приведены для очереди. В программе имелась только одна очередь с именем **BARBER**. Одинаковые имена очереди и *Обслуживающего устройства* — совпадение. Имена могут быть разными. В то же время, программа никогда не спутает *Очередь* и *Обслуживающее устройство* с одинаковыми именами, так как эти элементы обрабатываются по-разному и имеют совершенно разные параметры. Иногда оказывается удобным давать *Очереди* и *Обслуживающему устройству* одинаковые имена, чтобы указать на их взаимосвязь. Для *Очереди* в отчёте указаны следующие параметры:

MAX — наибольшее (максимальное) число клиентов в очереди (выше в программе осуществляется проверка длины очереди и клиенты встают в очередь только при условии, что её длина не более 1 человека, поэтому длина очереди не могла превышать 2);

CONT. — текущее содержание очереди на момент окончания сеанса моделирования (в данном случае 2);

ENTRY — общее число вошедших в очередь клиентов с начала моделирования (счётчик клиентов), в данном случае через очередь прошло 55 клиентов (некоторые не стали становиться в очередь, когда там уже был клиент);

ENTRY(0) — общее число вошедших в очередь клиентов, не считая тех, кто не ждал в очереди, а прошёл через неё сразу (например, первый клиент всегда проходит сразу, так как в очереди вначале никого нет);

AVE.CONT. — среднее число клиентов в очереди, определяется как число клиентов, делённое на общее число вошедших в очередь клиентов;

AVE.TIME — среднее время нахождения клиента в очереди (в данном случае 10,628 *мин*), определяется как суммарное время нахождения в очереди всех клиентов (по каждому из них собирается такая информация), делённая на число вошедших клиентов;

AVE.(-0) — то же, что и **AVE.TIME**, но без учёта тех клиентов, которые не ждали в очереди и прошли её "насквозь";

RETRY — число попыток повторных входов, если бы были неудачи (в данном случае их нет).

Ниже находится информация о *Таблицах гистограмм* **TABLE**. В программе имеется одна такая таблица с именем **WAITTIME**. По ней приводятся следующие данные:

MEAN — среднее значение (в данном случае — среднее значение времени нахождения транзактов в очереди);

STD.DEV. — среднеквадратическое отклонение (в данном случае времени нахождения транзактов в очереди);

RANGE — область (первая колонка нижней таблицы — *от* и вторая колонка нижней таблицы — *до*);

RETRY — число попыток повторных входов, если бы были неудачи (в данном случае их нет).

FREQUENCY — число транзактов, время нахождения которых в очереди в рамках указанного диапазона времени;

CUM.% — процент транзактов, время нахождения которых в очереди в лежит внутри указанного диапазона времени, нарастающим итогом.

Ниже в *Стандартном отчёте* приводится содержимое *Таблицы гистограмм* **TABLE**, рассчитанной в ходе имитационного эксперимента. Эту таблицы можно переписать для удобства восприятия иначе, поменяв столбцы и строки (Таблица 1.1).

Таблица 1.1 — Таблица для гистограммы программы-примера

Время в очереди, *мин*	от 0 до 2	от 2 до 4	от 4 до 6	от 6 до 8	от 8 до 10	от 10 до 12	от 12 до 14	от 14 до 16	от 16 до 18	от 18 до 20	от 20 до 22	от 22 до 24	от 24 до 26	от 26 до 28	от 28 до 30
Число транзактов	0	1	0	4	12	17	14	4	0	0	0	0	0	0	0
%	1,89	3,77	3,77	11,32	33,96	66,04	92,45	100,00							

В первой строке переписанной таблицы (Таблица 1.1) приводятся значения диапазона времени с шагом 2 минуты.

Во второй строке переписанной таблицы (Таблица 1.1) приводится число транзактов (клиентов), находившихся в очереди это время.

В третьей строке переписанной таблицы (Таблица 1.1) приводится процентное значение всех рассмотренных значений числа клиентов нарастающим итогом.

В частности, Таблица 1.1 означает, что 1 транзакт находился в очереди 2…4 минуты, 4 транзакта — 6…8 минут, 12 транзактов — 8…10 минут, 17 транзактов — 10…12 минут, 14 транзактов — 12…14 минут, 4 транзакта — 14…16 минут. Дольше 16 минут ни один транзакт в очереди не задерживался. Проценты можно трактовать следующим образом: 3,77% транзактов находились в очереди до 4 минут, 11,32% транзактов — до 8 минут, 33,96% транзактов — до 10 минут, 66,04% транзактов — до 12 минут, 92,45% транзактов — до 14 минут, все транзакты находились в очереди не более 16 минут.

Внешний вид гистограммы можно посмотреть, пройдя по меню: **Window → Simulation window → Table Window** (Рисунок 1.8).

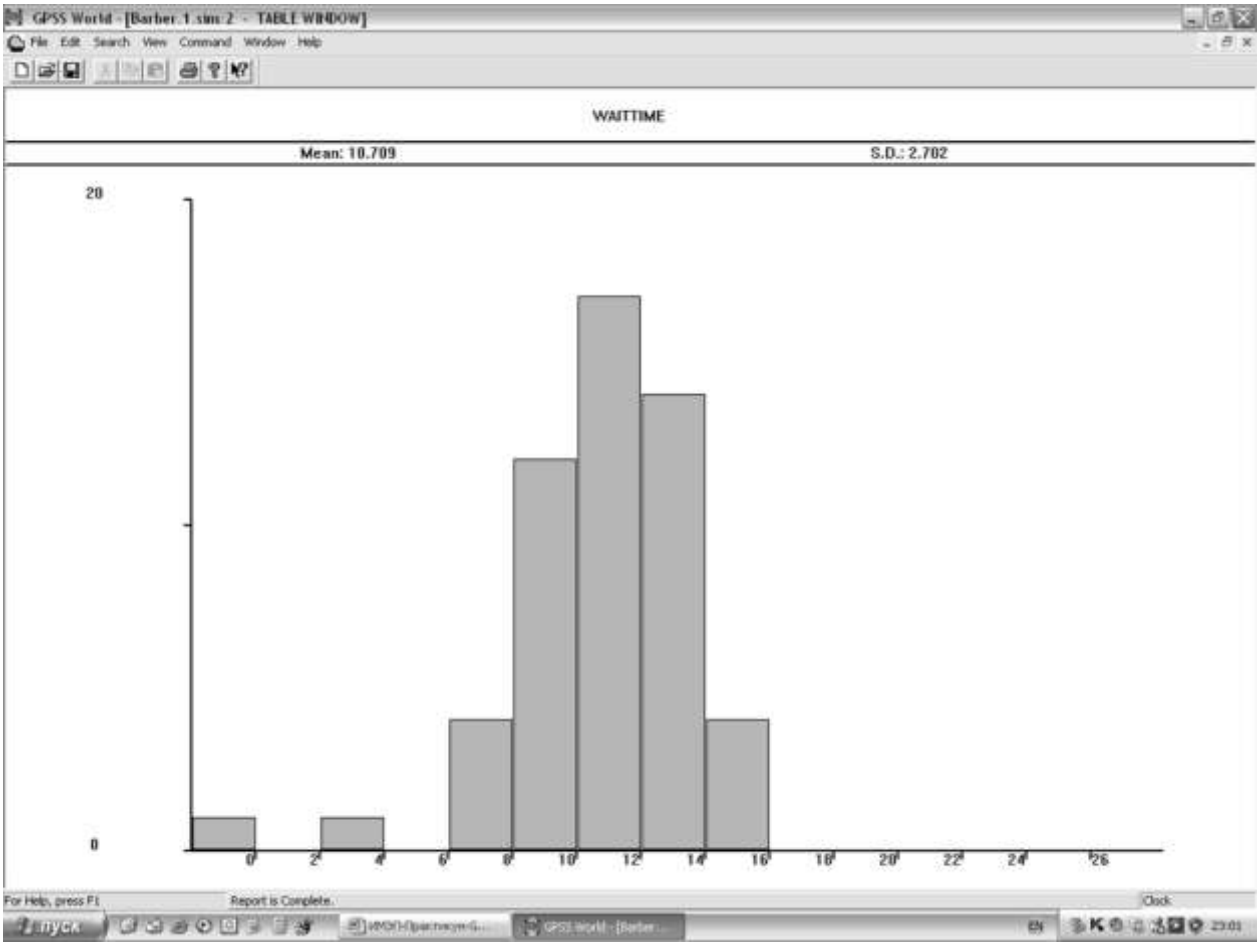


Рисунок 1.8 — Гистограмма в *Окне гистограмм* ***Table Window***

Из строки в тексте программы

```
Waittime  QTABLE    Barber,0,2,15
```

следует, что в *Таблице* с именем **Waittime** собирается информация об *Очереди* **Barber**. Эта информация представляет собой данные о том, сколько времени провели транзакты в данной *Очереди*. Всего необходимо собрать 15 интервалов времени с шагом 2 минуты. Смысл полученной гистограммы (Рисунок 1.8) рассмотрен выше с применением таблицы (Таблица 1.1).

По горизонтальной оси (оси абсцисс) отложены значения времени (в минутах с шагом 2), а по вертикальной оси (оси ординат) отложено число транзактов, находившихся в течение этого времени в очереди.

Гистограммы являются весьма распространённым инструментом визуального анализа результатов. В частности, по ним можно видеть, какие именно значения (в данном случае времени нахождения в очереди) встречались чаще всего. По графику (Рисунок 1.8) видно, что больше всего транзактов находилось в очереди от 10 до 12 минут (наибольший столбец графика), на втором месте — транзакты, находившиеся в очереди от 12 до 14 минут (правее наибольшего столбца графика), на третьем месте — транзакты, находившиеся в очереди от 8 до 10 минут (левее наибольшего столбца) и т. д.

Работа программы фиксируется в *Журнале* ***Journal***, который автоматически появляется в специальном окне. В данном случае *Журнал* содержит следующие записи:

11/27/06 23:00:35 Model Translation Begun.
11/27/06 23:00:35 Ready.
11/27/06 23:00:40 START 100
11/27/06 23:00:40 Simulation in Progress.
11/27/06 23:00:40 The Simulation has ended. Clock is 353.895450.
11/27/06 23:00:40 Reporting in Barber.1.1 - REPORT Window.

Их можно расшифровать следующим образом:

27 ноября 2006 года в 23 часа 00 минут 35 секунд местного времени началась трансляция модели (перед запуском модели на выполнение она транслируется).

27 ноября 2006 года в 23 часа 00 минут 35 секунд местного времени трансляция модели закончилась.

27 ноября 2006 года в 23 часа 00 минут 40 секунд местного времени модель была запущена на выполнение с записью в *Счётчике завершения* числа внешних циклов 100.

27 ноября 2006 года в 23 часа 00 минут 40 секунд местного времени модель начала выполняться.

27 ноября 2006 года в 23 часа 00 минут 40 секунд местного времени работа модели закончилась. Общее модельное время составило 353,895459 единиц модельного времени (например, минут). (Модельное время — это то время, которое фигурирует в программе, оно никак не связано с временем работы программы.)

27 ноября 2006 года в 23 часа 00 минут 40 секунд местного времени сформирован Стандартный отчёт, помещённый в окне **Barber.1.1 - REPORT Window** (затем этот отчёт можно записать в файл).

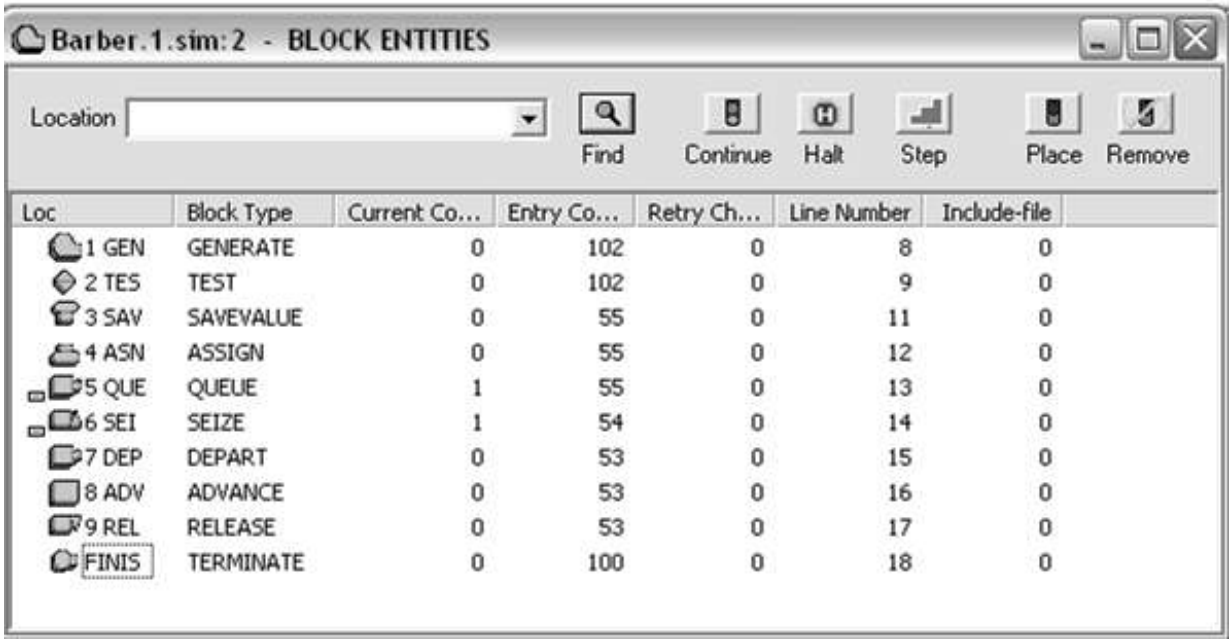
Поскольку программа работала очень быстро, времена некоторых операций практически совпадают.

Дополнительную информацию можно получить с помощью *Окон*. Для этого следует пройти через меню:

Window → Simulation Window → Blocks Window;
Window → Simulation Window → Facilities Window;
Window → Simulation Window → Queues Window;
Window → Simulation Window → Savevalues Window.

В результате будут вызваны соответствующие *Окна*.

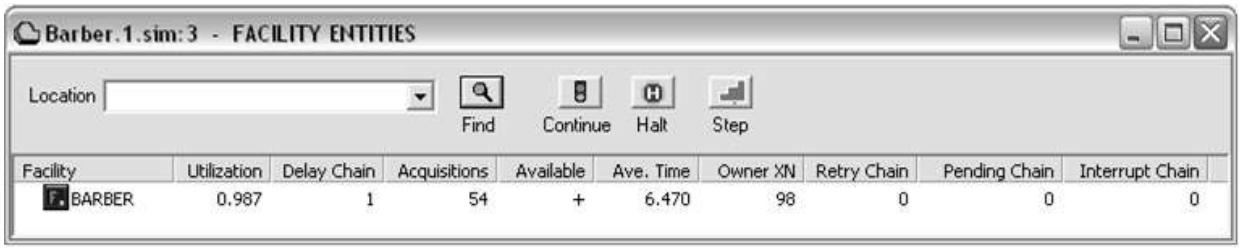
Окно блоков ***Blocks Window*** (Рисунок 1.9) предназначено для анимации программы. В нём приводится текст программы со всеми блоками. В самой левой колонке показаны транзакты в виде прямоугольников. *Окно блоков* позволяет проследить движение транзактов через блоки и логику работы программы.



Loc	Block Type	Current Co...	Entry Co...	Retry Ch...	Line Number	Include-file
1 GEN	GENERATE	0	102	0	8	0
2 TES	TEST	0	102	0	9	0
3 SAV	SAVEVALUE	0	55	0	11	0
4 ASN	ASSIGN	0	55	0	12	0
5 QUE	QUEUE	1	55	0	13	0
6 SEI	SEIZE	1	54	0	14	0
7 DEP	DEPART	0	53	0	15	0
8 ADV	ADVANCE	0	53	0	16	0
9 REL	RELEASE	0	53	0	17	0
FINIS	TERMINATE	0	100	0	18	0

Рисунок 1.9 — *Окно блоков* ***Blocks Window***

Окно блоков широко применяется при отладке программ, а также при исследовании некоторых особенностей алгоритмов. Для этого следует запускать программу в шаговом режиме, нажимая функциональную клавишу **F5** или кнопку **Step** на панели *Окна блоков*.



Facility	Utilization	Delay Chain	Acquisitions	Available	Ave. Time	Owner XN	Retry Chain	Pending Chain	Interrupt Chain
BARBER	0.987	1	54	+	6.470	98	0	0	0

Рисунок 1.10 — *Окно обслуживающих устройств* ***Facilities Window***

В *Окне обслуживающих устройств* (Рисунок 1.10) приводится список всех *Обслуживающих устройств*, в данном случае оно одно — **Barber**. В нём по колонкам приводится информация, которая частично совпадает с информацией об *Обслуживающих устройствах* в *Стандартном отчёте*, но есть различия в названиях параметров. Например, параметр *Стандартного отчёта* **ENTRIES** в *Окне* назван **Acquisitions**, параметр **UTIL.** — **Utilization**, **OWNER** — **Owner XN** (аббревиатура XN в документации по GPSS World означает слово "транзакт") и т. д. В целом данные *Стандартного отчёта* и *Окна* совпадают, но их назначение — разное. *Стандартный отчёт* появляется после завершения автоматического выполнения программы и приводимые в нём параметры относятся к последнему моменту расчётов. В *Окне* можно наблюдать текущие изменения параметров — как при автоматическом выполнении программы, так и в шаговом режиме. Если задано автоматическое выполне-

ние программы, то её можно остановить в любой момент нажатием кнопки **Halt** на панели *Окна*□, а затем вновь запустить программу нажатием кнопки **Continue**. Шаговый режим обеспечивается нажатием функциональной клавиши **F5** или кнопки **Step** на панели *Окна*□.

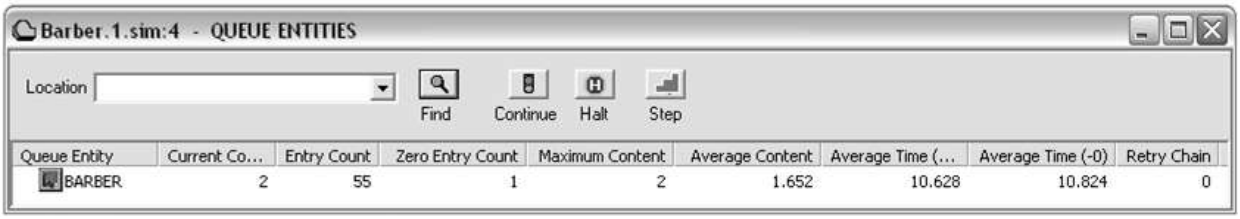


Рисунок 1.11 — *Окно очередей* ***Queues Window***

Окно очередей (Рисунок 1.11) позволяет просмотреть изменения параметров *Очередей* в автоматическом или шаговом режимах работы. Информация об *Очередях* в *Окне* в целом совпадает с той, что приводится в *Стандартном отчёте*.

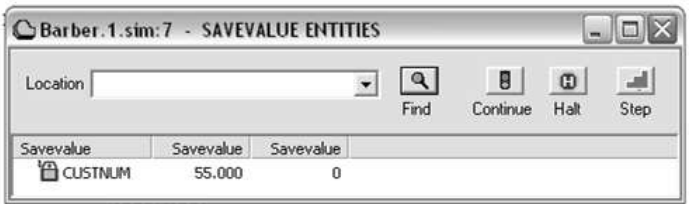


Рисунок 1.12 — *Окно сохраняемых величин* ***Savevalues Window***

Окно сохраняемых величин (Рисунок 1.12) содержит список *Сохраняемых величин* с единственным параметром — значением соответствующей *Сохраняемой величины*.

Таким образом, в целом работа с готовым примером модели включала следующие этапы:

– вызов файла с готовой моделью и возможная её правка;

– трансляция с помощью команд **Command → Create Simulation**;

– запуск программы на счёт в автоматическом режиме с помощью команд **Command → Start**;

– получение и просмотр *Стандартного отчёта*;

– просмотр *Окон* и, возможно, выполнение отдельных шагов в шаговом режиме;

– сохранение, если нужно, открытых окон (файлов в этих окнах).

Возможны также другие операции, например, перетрансляция программы, последовательное задание командой **Start** новых сеансов и т. д.

Окна можно расположить на экране рядом и тогда появляется возможность одновременно наблюдать за изменением всех величин для отладки программы или выявления особенностей её функционирования (Рисунок 1.13).

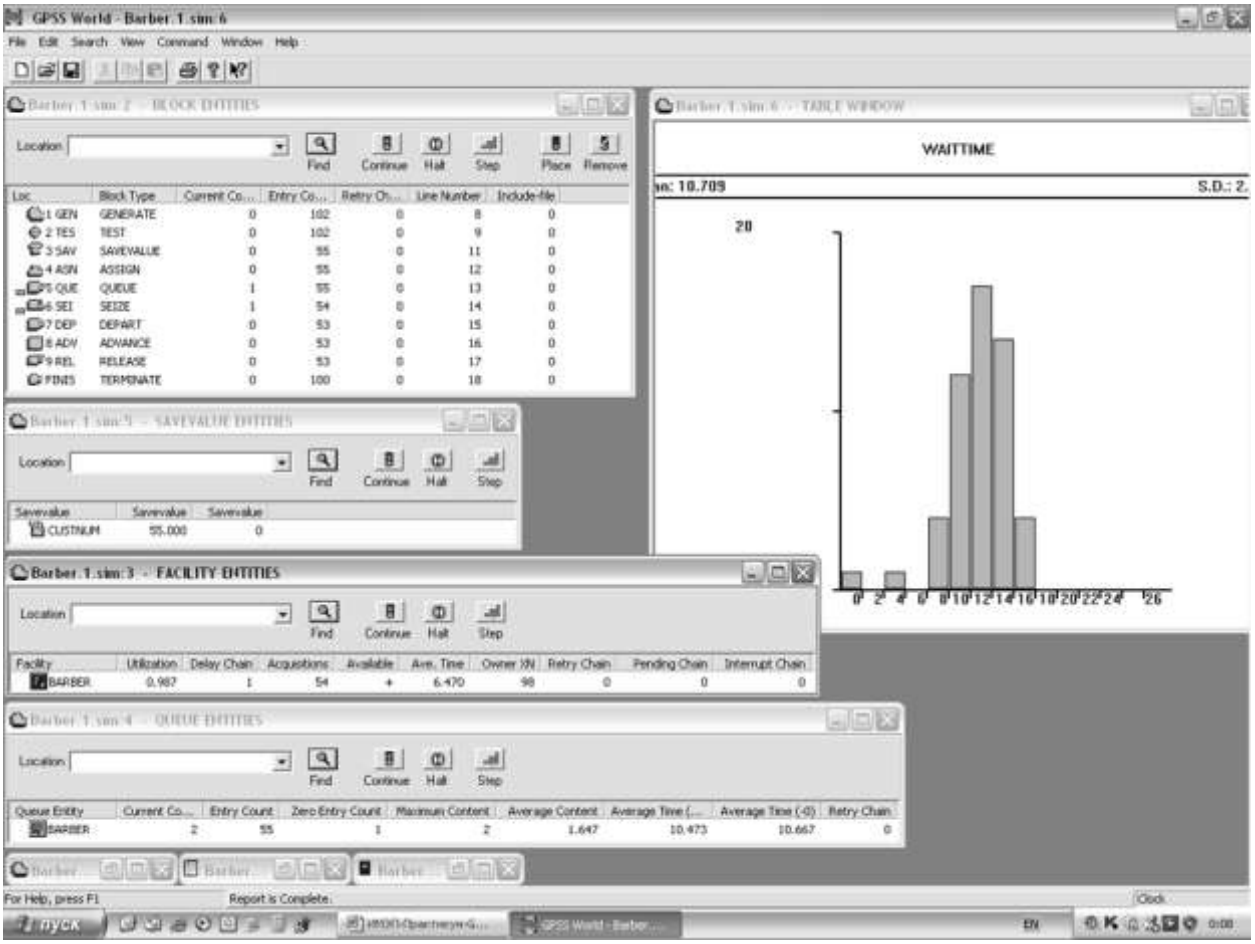


Рисунок 1.13 — Одновременное использование нескольких *Окон*

Если задать в этом режиме новое количество шагов через команду **START** (меню **Command → START**), то можно наблюдать динамическое изменение всех параметров одновременно во всех окнах. Для временного останова программы нажимают кнопку **Halt** в любом окне, а для продолжения работы — кнопку **Continue**.

Можно также использовать шаговый режим, нажимая функциональную клавишу **F5** или кнопку **Step** в любом окне.

2 Примеры разработки моделей организационных объектов

2.1 Модель предприятия обслуживания

Целью данной разработки является демонстрация процесса постепенного наращивания модели объекта при усложнении условий его функционирования. При этом вводятся и изучаются новые понятия языка GPSS World, операторы и приёмы программирования. Для примера выбрана модель парикмахерской. Ниже приводятся условия моделируемой ситуации (задачи моделирования), текст соответствующей программы и пояснения к ней. Первоначально используется одноканальное *Обслуживающее устройство*, т. е. одно кресло парикмахера и для этого случая представляется последовательность программных моделей. Затем рассматривается *Многоканальное устройство*, соответствующее нескольким креслам в парикмахерской, и для этого случая также рассматривается несколько последовательно усложняющихся моделей.

Модель 2.1

Условия задачи: Клиент заходит в парикмахерскую через каждые 25±20 минут. За это время он успевает войти, обслужиться и выйти.

```
*=============================================================================
* GPSS World - barber01.gps
*-----------------------------------------------------------------------------
* Парикмахерская с 1 креслом
*-----------------------------------------------------------------------------
* Клиент входит и выходит
*=============================================================================
      GENERATE     25,20 ; Клиент пришёл
      TERMINATE    1     ; Клиент ушёл
```

Пояснения. Программа является минимально возможной по составу операторов — в ней присутствует только два оператора — для создания **GENERATE** и уничтожения **TERMINATE** транзактов. В данном случае новый транзакт (т. е. новый клиент) появляется через каждые 25±20 условных единиц времени (например, минут), т. е. интервал между двумя транзактами лежит в диапазоне 5….45 условных единиц времени, причём конкретное значение интервала определяется случайным образом с использованием равномерного закона распределения. Здесь отсутствуют какие-либо устройства (*Обслуживающие устройства*, *Очереди* и др.), а также задержки времени. Несмотря на это, программа транслируется и работает. С помощью *Окна блоков* ***Blocks Window*** можно увидеть процесс рождения и уничтожения транзактов, а с помощью *Окна выражений* ***Expressons Window*** можно смотреть, какие именно транзакты появляются и уничтожаются (с помощью СЧА **XN1**). При этом можно использовать, например, пошаговый режим, нажимая клавишу **F5**. Если программа не содержит указанных операторов, то в ней

нечему будет работать, так как все другие операторы только тогда производят действия, если "оживляются" транзактами.

Модель 2.2

Условия задачи: Клиент заходит в парикмахерскую через каждые 25±20 минут. Время обслуживания составляет 25±5 минут. Статистика очереди не ведётся.

```
*=============================================================================
* GPSS World - barber02.gps
*-----------------------------------------------------------------------------
* Парикмахерская с 1 креслом
*-----------------------------------------------------------------------------
* Клиент входит, обслуживается и выходит
*=============================================================================
      GENERATE     25,20  ; Клиент пришёл
      SEIZE        Kreslo ; Пошёл обслуживаться
      ADVANCE      25,5   ; Обслуживается
      RELEASE      Kreslo ; Закончил обслуживание
      TERMINATE    1      ; Клиент ушёл
```

Пояснения. По сравнению с предыдущей программой (Модель 2.1), данная версия программы дополнена операторами занятия транзактом *Обслуживающего устройства* **SEIZE**, задания времени обслуживания **ADVANCE** и освобождения *Обслуживающего устройства* **RELEASE**. В этой версии программа является минимально возможной для выполнения конкретного действия: новый транзакт (т. е. новый клиент) появляется через каждые 25±20 ед. времени (например, минут), занимает *Обслуживающее устройство* ***Kreslo***, сидит в нём в течение 25±5 единиц времени, освобождает ***Kreslo*** и уходит. Интервалы появления клиента и продолжительность его обслуживания подобраны таким образом, что они в среднем одинаковы (25 ед. времени), но из-за разброса могут случайным образом не совпадать. Время обслуживания имеет небольшой разброс (от 20 до 30 ед. времени), а интервалы прихода клиентов — от 5 до 45 ед. времени. В результате возможно образование очереди. Но в данной программе нет операторов, которые контролируют наличие очереди и её статистику (количество находящихся в ней клиентов, среднее время нахождения в очереди и др.). Поэтому, хотя очередь и может образовываться, но о ней из работы такой программы никакой информации получить нельзя.

Модель 2.3

Условия задачи: Клиент заходит в парикмахерскую через каждые 25±20 минут. Время обслуживания составляет 25±5 минут. Ведётся статистика очереди.

```
*=============================================================================
* GPSS World - barber03.gps
*-----------------------------------------------------------------------------
* Парикмахерская с 1 креслом
*-----------------------------------------------------------------------------
* Клиент входит, занимает кресло, а если оно уже занято, то встаёт в очередь
*=============================================================================
      GENERATE     25,20    ; Клиент пришёл
```

```
      QUEUE        Ochered  ; Встал в очередь
      SEIZE        Kreslo   ; Пошёл обслуживаться
      DEPART       Ochered  ; Покинул очередь
      ADVANCE      25,5     ; Обслуживается
      RELEASE      Kreslo   ; Закончил обслуживание
      TERMINATE    1        ; Клиент ушёл
```

Пояснения. По сравнению с предыдущей программой (Модель 2.3), данная версия программы дополнена операторами занятия **QUEUE** и освобождения **DEPART** *Очереди* ***Ochered*** перед *Обслуживающим устройством* ***Kreslo***. В этой версии программа обеспечивает контроль очереди и её статистику (количество находящихся в ней клиентов, среднее время нахождения в очереди и др.). Наличие операторов, связанных с очередью, не является необходимым условием работы — сами по себе они не обеспечивают наличие или отсутствие очередей. Включение операторов **QUEUE** и **DEPART** позволяет лишь собирать статистику конкретной очереди в конкретном месте программы, при этом они должны быть всегда в паре. Эти операторы должны располагаться таким образом, чтобы точно фиксировать вхождение транзакта в конкретную очередь и покидание её. Например, если оператор **DEPART** поставить после оператора **ADVANCE**, то время выхода из очереди будет искажено временем обслуживания.

Модель 2.4

Условия задачи: Клиент заходит в парикмахерскую через каждые 20±15 минут. Время обслуживания составляет 25±10 минут. Ведётся статистика очереди. Моделируется рабочий день (480 минут).

```
*=============================================================================
* GPSS World - barber04.gps
*-----------------------------------------------------------------------------
* Парикмахерская с 1 креслом
*-----------------------------------------------------------------------------
* Клиент входит, занимает кресло, а если оно уже занято, то встаёт в очередь
* Особенность: использование таймера с отдельным транзактом
*=============================================================================
      GENERATE     20,15    ; Клиент пришёл
      QUEUE        Ochered  ; Встал в очередь
      SEIZE        Kreslo   ; Пошёл обслуживаться
      DEPART       Ochered  ; Покинул очередь
      ADVANCE      25,10    ; Обслуживается
      RELEASE      Kreslo   ; Закончил обслуживание
      TERMINATE    0        ; Клиент ушёл
; Таймер
      GENERATE     480      ; Рабочий день окончился
      TERMINATE    1        ; Моделирование окончилось
      START        1        ; Число дней моделирования
```

Пояснения. По сравнению с предыдущей программой (Модель 2.3), данная версия программы имеет специальный таймер для задания времени её работы. В предыдущей версии можно было задать количество транзактов (клиентов), которых обслуживает парикмахерская, и затем определить время её работы и другие показатели. Но в реальности необходимо задавать время работы парикмахерской и определять количество прошедших через неё клиентов. Поэтому в нижней части программы присутствует специальный фраг-

мент под названием "**Таймер**", в котором имеется лишь два оператора — **GENERATE** и **TERMINATE**. Оператор **GENERATE** генерирует через 480 ед. времени один транзакт (число генерируемых транзактов не обозначено, но в этом случае по умолчанию понимается 1). Число 480 определяется как 60 минут, умноженные на 8 часов, т. е. время оказывается задано в минутах. Перед генерацией этого транзакта через каждые 20±15 минут в верхней части программы генерируются другие транзакты, означающие появление нового клиента. В течение 25±10 минут этот клиент обслуживается парикмахером. Постепенно модельное время приближается к 480 минутам и когда достигает его, генерируется одиночный транзакт, физически соответствующий, например, звонку об окончании рабочего дня. Самый последний оператор **START 1** задаёт число для *Счётчика завершений*, поэтому после срабатывания оператора **TERMINATE 1** в нижней части программы из *Счётчика завершений* вычитается 1, он обнуляется и программа останавливается. Фактически, с помощью оператора **START** задаётся число моделируемых дней. Необходимо обратить внимание, что в верхнем основном фрагменте программы оператор **TERMINATE 0** не влияет на содержимое счётчика завершений, так как содержит в качестве операнда ***0***, хотя уничтожает каждый транзакт, который входит в него. Это необходимо сделать, так как в следующем нижнем фрагменте "**Таймер**" программы имеется новый оператор **GENERATE**, в который транзакты входить не могут. В программе может быть любое количество операторов **GENERATE**, но в конце действия каждого из них перед следующим оператором **GENERATE** обязательно должен стоять оператор **TERMINATE**. Если этого не сделать, то при трансляции ошибка не будет обнаружена, но при работе программы при попытке транзакта войти в оператор **GENERATE** будет диагностирована ошибка и программа аварийно завершится.

Модель 2.5

Условия задачи: Клиенты делятся на два типа — клиенты, желающие только постричься и желающие постричься и побриться. Каждый клиент, желающий постричься, заходит в парикмахерскую через каждые 30±10 минут. Каждый клиент, желающий постричься и побриться заходит через каждые 60±20 минут. Таким образом, число клиентов, желающих одновременно постричься и побриться, в два раза меньше тех, кто хочет только постричься. Время стрижки составляет 18±6 минут, время бритья составляет 10±2 минут. Ведётся статистика очереди для тех, кто только стрижётся и для тех, кто стрижётся и бреется. Моделируется рабочий день (480 минут).

```
*=============================================================================
* GPSS World - barber05.gps
*-----------------------------------------------------------------------------
* Парикмахерская с 1 креслом
*-----------------------------------------------------------------------------
* Два потока клиентов: те, кто только стрижётся и те, кто стрижётся и бреется
*=============================================================================
```

```
; Стрижка
      GENERATE      30,10    ; Пришёл клиент для стрижки
      QUEUE         Ochered  ; Клиент для стрижки встал в очередь
      SEIZE         Kreslo   ; Клиент для стрижки пошёл стричься
      DEPART        Ochered  ; Клиент для стрижки покинул очередь
      ADVANCE       18,6     ; Клиент стрижётся
      RELEASE       Kreslo   ; Клиент закончил стрижку
      TERMINATE     0        ; Клиент ушёл
; Стрижка и бритьё
      GENERATE      60,20    ; Пришёл клиент для стрижки и бритья
      QUEUE         Ochered  ; Клиент встал в очередь
      SEIZE         Kreslo   ; Клиент пошёл обслуживаться
      DEPART        Ochered  ; Клиент покинул очередь
      ADVANCE       18,6     ; Клиент стрижётся
      ADVANCE       10,2     ; Клиент бреется
      RELEASE       Kreslo   ; Клиент закончил стрижку и бритьё
      TERMINATE     0        ; Клиент ушёл
; Таймер
      GENERATE      480      ; Рабочий день окончился
      TERMINATE     1        ; Моделирование окончилось
      START         1        ; Число дней моделирования
```

Пояснения. По сравнению с предыдущей программой (Модель 2.4), в данной версии программы моделируется два раздельных потока транзактов — клиенты для стрижки и клиенты для стрижки и бритья. Первый верхний фрагмент программы "**Стрижка**" повторяет предыдущую версию программы. Второй верхний фрагмент программы "**Стрижка и бритьё**" также повторяет предыдущую версию программы, но с добавлением ещё одного оператора **ADVANCE** ***10,2***, задающего продолжительность бритья. Поскольку в парикмахерской только одно кресло и одна очередь, то операторы **QUEUE** (**DEPART**) и **SEIZE** (**RELEASE**) в обоих верхних фрагментах программы оперируют с одними и теми же *Обслуживающим устройством* ***Kreslo*** и *Очередью* ***Ochered***. Два раздельных потока транзактов является одним из возможных вариантов задания клиентов с разными целями обслуживания, но видно, что программа дважды содержит схожие операторы, что является недостатком такой реализации. Ниже рассмотрен другой вариант реализации программы для моделирования тех же условий.

Модель 2.6

Условия задачи: Условия задачи те же, что и в предыдущей версии модели (модель 5). Клиенты делятся на два типа — клиенты, желающие только постричься и желающие постричься и побриться. Клиенты появляются через каждые 25±5 минут. Все желают постричься. Желающих постричься и побриться 40% от общего числа клиентов. Ведётся статистика очереди для всех клиентов. Моделируется рабочий день (480 минут).

```
*=============================================================================
* GPSS World - barber06.gps
*-----------------------------------------------------------------------------
* Парикмахерская с 1 креслом
*-----------------------------------------------------------------------------
* Один поток клиентов, но некоторые из них только стригутся, а некоторые
* стригутся и бреются
*=============================================================================
      GENERATE      25,5             ; Клиент пришёл
      QUEUE         Ochered          ; Встал в очередь
```

```
       SEIZE        Kreslo          ; Пошёл обслуживаться
       DEPART       Ochered         ; Покинул очередь
       ADVANCE      20,5            ; Стрижётся
       TRANSFER     0.4,Met1,Met2   ; 40% выбирают бритьё (Met2),60% стрижку (Met1)
Met2   ADVANCE      15,3            ; Бреется
Met1   RELEASE      Kreslo          ; Закончил обслуживание
       TERMINATE    0               ; Клиент ушёл
; Таймер
       GENERATE     480             ; Рабочий день окончился
       TERMINATE    1               ; Моделирование окончилось
       START        1               ; Число дней моделирования
```

Пояснения. По сравнению с предыдущей программой (Модель 2.5), в данной версии программы моделируется один поток транзактов — клиентов для стрижки и клиентов для стрижки и бритья. Т. е. все клиенты собираются стричься, но часть из них — также бриться. Для разделения потока клиентов используется оператор **TRANSFER *0.4,Met1,Met2***, обеспечивающий переход входящего в него потока транзактов на метки ***Met2*** (с вероятностью 0.4, т. е. для 40% транзактов) или ***Met1*** (с вероятностью 1 – 0.4 = 0.6, т. е. для оставшихся 60% транзактов). Метка ***Met1*** связана с оператором освобождения *Обслуживающего устройства* **RELEASE *Kreslo*** и приводит к выходу транзакта из модели. Метка ***Met2*** связана с оператором **ADVANCE *15,3***, задающим продолжительность бритья 15±3 минуты.

Модель 2.7

Условия задачи: Условия задачи немного изменены по сравнению с предыдущей версией модели (Модель 2.6). Теперь клиенты делятся на два типа — клиенты, желающие только постричься и желающие только побриться. Клиентов, которые хотят того и другого — нет. Клиенты появляются через каждые 20±5 минут. Желающих постричься 60% от общего числа клиентов, желающих побриться 40% от общего числа клиентов. Продолжительность стрижки 25±10 минут, продолжительность бритья 15±3 минуты. Необходима статистика очереди для всех клиентов. Моделируется рабочий день (480 минут).

```
*=========================================================================
* GPSS World - barber07.gps
*-------------------------------------------------------------------------
* Парикмахерская с 1 креслом
*-------------------------------------------------------------------------
* Часть клиентов только стрижётся, часть только бреется
*=========================================================================
       GENERATE     20,5            ; Клиент пришёл
       QUEUE        Ochered         ; Встал в очередь
       SEIZE        Kreslo          ; Пошёл обслуживаться
       DEPART       Ochered         ; Покинул очередь
       TRANSFER     0.4,Met1,Met2   ; 40% выбирают бритьё (Met2),60% стрижку (Met1)
Met1   ADVANCE      25,10           ; Стрижётся
       TRANSFER     ,Met3           ; Закончил стрижку
Met2   ADVANCE      15,3            ; Бреется
Met3   RELEASE      Kreslo          ; Закончил обслуживание
       TERMINATE    0               ; Клиент ушёл
; Таймер
       GENERATE     480             ; Рабочий день кончился
       TERMINATE    1               ; Моделирование окончилось
       START        1               ; Число дней моделирования
```

Пояснения. По сравнению с предыдущей программой (Модель 2.6), в данной версии программы моделируется поток транзактов — клиентов только для стрижки и клиентов только для бритья. При этом. 40% клиентов собираются только стричься, 60% только бриться. В целом программа построена также, как и в предыдущем случае, т. е. для разделения потока клиентов используется оператор **TRANSFER *0.4,Met1,Met2***, обеспечивающий переход входящего в него потока транзактов на метки ***Met2*** (с вероятностью 0.4, т. е. для 40% транзактов) или ***Met1*** (с вероятностью 1 – 0.4=0.6, т. е. для оставшихся 60% транзактов). Метка ***Met1*** связана с оператором освобождения *Обслуживающего устройства* **RELEASE *Kreslo*** и приводит к выходу транзакта из модели. Метка ***Met2*** связана с оператором **ADVANCE *15,3***, задающим продолжительность бритья 15±3 минуты. В данную версию программы введён дополнительный оператор безусловного перехода **TRANSFER *,Met3***, отправляющий тех, кто постригся, на выход.

Модель 2.8

Условия задачи: Условия задачи дополнительно изменены по сравнению с предыдущей версией модели (Модель 2.7). Теперь клиенты делятся на три типа — клиенты, желающие только постричься, желающие только побриться и желающие постричься и побриться. Клиенты появляются через каждые 20±5 минут. Желающих постричься 60% от общего числа клиентов и из них 20% желают также побриться, желающих только побриться 40% от общего числа клиентов. Продолжительность стрижки 25±10 минут, продолжительность бритья 15±3 минуты. Необходима статистика очереди для всех клиентов. Моделируется рабочий день (480 минут).

```
*=============================================================================
* GPSS World - barber08.gps
*-----------------------------------------------------------------------------
* Парикмахерская с 1 креслом
*-----------------------------------------------------------------------------
* Часть клиентов только стрижётся, часть только бреется, часть стрижётся и бреется
*=============================================================================
       GENERATE     20,5              ; Клиент пришёл
       QUEUE        Ochered           ; Встал в очередь
       SEIZE        Kreslo            ; Пошёл обслуживаться
       DEPART       Ochered           ; Покинул очередь
       TRANSFER     0.4,Met1,Met2     ; 40% выбирают бритьё (Met2),60% стрижку (Met1)
Met1   ADVANCE      25,10             ; Стрижётся
       TRANSFER     0.2,Met3,Met2     ; Выбирает бритьё Met2 дополнительно к стрижке
Met2   ADVANCE      15,3              ; Бреется
Met3   RELEASE      Kreslo            ; Закончил обслуживание
       TERMINATE    0                 ; Клиент ушёл
; Таймер
       GENERATE     480               ; Рабочий день окончился
       TERMINATE    1                 ; Моделирование окончилось
       START        1                 ; Число дней моделирования
```

Пояснения. По сравнению с предыдущей программой (Модель 2.7), в данной версии моделируется один поток транзактов для трёх типов клиентов — только для стрижки, только для бритья, для стрижки и бритья. При этом 40% клиентов собираются только стричься, 60% только бриться, 20% от

постригшихся — хотят также и побриться. В целом программа построена так же, как и в предыдущем случае, т. е. для разделения потока клиентов используется оператор **TRANSFER** ***0.4,Met1,Met2***, обеспечивающий переход входящего в него потока транзактов на метки ***Met2*** (с вероятностью 0.4, т. е. для 40% клиентов, желающих только бриться) или ***Met1*** (с вероятностью 1 – 0.4 = 0.6, т. е. для оставшихся 60% клиентов, желающих стричься и, возможно, также бриться). Метка ***Met1*** связана с оператором **ADVANCE** ***25,10***, задающим время стрижки 25±10 минут. После чего стоит оператор **TRANSFER** ***0.2,Met3,Met2***, обеспечивающий дополнительное разделение потока постригшихся клиентов на тех, кто будет (20%, метка ***Met2***) и не будет (1 – 0.2 = 0.8 = 80%, метка ***Met3***) бриться. Метка ***Met3*** связана с оператором освобождения *Обслуживающего устройства* **RELEASE** ***Kreslo*** и приводит к выходу затем транзакта из модели через оператор **TERMINATE 0**. Метка ***Met2*** связана с оператором **ADVANCE** ***15,3***, задающим продолжительность бритья 15±3 минуты, после чего транзакт естественным образом попадает на следующий оператор **RELEASE** ***Kreslo*** и далее — на выход через оператор **TERMINATE 0**.

Модель 2.9

Условия задачи: Условия задачи дополнительно изменены по сравнению с предыдущей версией модели (Модель 2.8) введением контроля длины очереди: каждый вошедший клиент оценивает длину очереди и если она больше 2 человек, уходит, а если меньше — встаёт в очередь. В остальном условия — те же, т. е. клиенты делятся на три типа — клиенты, желающие только постричься, желающие только побриться и желающие постричься и побриться. Клиенты появляются через каждые 20±5 минут. Желающих постричься 60% от общего числа клиентов и из них 20% желают также побриться, желающих только побриться 40% от общего числа клиентов. Продолжительность стрижки 35±10 минут, продолжительность бритья 25±3 минуты. Необходима статистика очереди для всех клиентов. Моделируется рабочий день (480 минут).

```
*=============================================================================
* GPSS World - barber09.gps
*-----------------------------------------------------------------------------
* Парикмахерская с 1 креслом
*-----------------------------------------------------------------------------
* Часть клиентов только стрижётся, часть только бреется, часть стрижётся и бреется.
* Новый клиент оценивает длину очереди и уходит, если она велика
*=============================================================================
       GENERATE     20,5              ; Пришёл клиент
       TEST LE      Q$Ochered,1,Met4  ; Проверка длины очереди и уход, если она >1
       QUEUE        Ochered           ; Встал в очередь
       SEIZE        Kreslo            ; Пошёл обслуживаться
       DEPART       Ochered           ; Покинул очередь
       TRANSFER     0.4,Met1,Met2     ; Выбирает бритьё Met2 или стрижку Met1
Met1   ADVANCE      35,10             ; Стрижётся
       TRANSFER     0.2,Met3,Met2     ; Выбирает бритьё Met2 дополнительно к стрижке
Met2   ADVANCE      25,3              ; Бреется
Met3   RELEASE      Kreslo            ; Закончил обслуживание
```

```
       TRANSFER     ,Met5             ; Пошёл к выходу
Met4   SAVEVALUE    Klient_No+,1      ; Подсчёт клиентов, не ставших ждать
Met5   TERMINATE    0                 ; Клиент ушёл
; Таймер
       GENERATE     480               ; Рабочий день окончился
       TERMINATE    1                 ; Моделирование окончилось
       START        1                 ; Число дней моделирования
```

Пояснения. По сравнению с предыдущей программой (Модель 2.8), в данной версии моделируется возможность покидания клиентом парикмахерской без занятия очереди, если длина её оказывается больше 2 человек. Для оценки длины очереди используется оператор **TEST LE *Q$Ochered,1,Met4***, смысл которого в том, что выполняется сравнение длины очереди (определяемой с помощью СЧА ***Q$Ochered***) с максимальной её длиной (***1***), когда клиент ещё готов в неё встать. Если условие "меньше или равно" (**LE**) выполняется, то транзакт проходит к следующему по порядку оператору, если не выполняется — на метку ***Met4***. С этой меткой связан оператор **SAVEVALUE *Klient_No+,1***, который занимается подсчётом ушедших без обслуживания клиентов (переменная ***Klient_No***). Знак "плюс" в конце этой переменной означает, что к содержимому этой переменной каждый раз прибавляется второй операнд (***1***). Начальное значение переменной ***Klient_No*** устанавливается программой в 0. Эту величину можно посмотреть в динамике в *Окне сохраняемых величин* ***Savevalues Window***.

Модель 2.10

Условия задачи: Условия задачи дополнительно изменены по сравнению с предыдущей версией модели (Модель 2.9). Так же, как и в предыдущей модели, новый клиент оценивает длину очереди и если она больше 2 человек, уходит, а если меньше — встаёт в очередь. Но некоторые клиенты имеют право на обслуживание вне очереди и длина очереди их не волнует, таких предполагается 10% от общего числа клиентов. В остальном условия — те же, т. е. клиенты делятся на три типа — клиенты, желающие только постричься, желающие только побриться и желающие постричься и побриться. Клиенты появляются через каждые 20±5 минут. Желающих постричься 60% от общего числа клиентов и из них 20% желают также побриться, желающих только побриться 40% от общего числа клиентов. Продолжительность стрижки 35±10 минут, продолжительность бритья 25±3 минуты. Необходима статистика очереди для всех клиентов. Моделируется рабочий день (480 минут).

```
*=========================================================================
* GPSS World - barber10.gps
*-------------------------------------------------------------------------
* Парикмахерская с 1 креслом
*-------------------------------------------------------------------------
* Часть клиентов только стрижётся, часть только бреется, часть стрижётся и бреется.
* Новый клиент оценивает длину очереди и уходит, если она велика
* Часть клиентов имеют льготы по обслуживанию вне очереди и они остаются, не
* зависимо от длины очереди и проходят, минуя её
```

```
*=========================================================================
       GENERATE     20,5              ; Клиент пришёл
       TRANSFER     0.1,Met6,Met7     ; 10% клиентов с правом пройти без очереди
Met7   PRIORITY     10                ; Получает право внеочередного обслуживания
       SAVEVALUE    Klient_Lgot+,1    ; Подсчёт клиентов, имеющих льготу
       TRANSFER     ,Met8             ; Пошёл в начало очереди
Met6   TEST L       Q$Ochered,2,Met4  ; Проверка очереди и уход, если она >2 чел.
Met8   QUEUE        Ochered           ; Встал в очередь
       SEIZE        Kreslo            ; Пошёл обслуживаться
       DEPART       Ochered           ; Покинул очередь
       TRANSFER     0.4,Met1,Met2     ; 40% выбирают бритьё (Met2),60% стрижку (Met1)
Met1   ADVANCE      35,10             ; Стрижётся
       TRANSFER     0.2,Met3,Met2     ; 20% выбирают дополнительно бритьё (Met2)
Met2   ADVANCE      25,3              ; Бреется
Met3   RELEASE      Kreslo            ; Закончил обслуживание
       TRANSFER     ,Met5             ; Пошёл к выходу
Met4   SAVEVALUE    Klient_No+,1      ; Подсчёт клиентов, не ставших ждать
Met5   TERMINATE    0                 ; Клиент ушёл
; Таймер
       GENERATE     480               ; Рабочий день окончился
       TERMINATE    1                 ; Моделирование окончилось
       START        1                 ; Число дней моделирования
```

Пояснения. По сравнению с предыдущей программой (Модель 2.9), в данной версии моделируется возможность обслуживания некоторых клиентов вне очереди. Для этого используется оператор задания приоритета транзакта **PRIORITY**, на который направляется 10% вошедших клиентов (такая задана пропорция льготников от общего числа клиентов) с помощью оператора **TRANSFER** ***0.1,Met6,Met7***. Сразу после оператора ***Met7*** **PRIORITY** ***10*** стоит оператор **SAVEVALUE** ***Klient_Lgot+,1***, подсчитывающий в переменной ***Klient_Lgot*** число таких льготников. Затем с помощью безусловного оператора **TRANSFER**,***Met8*** льготник обходит оператор оценки длины очереди ***Met6*** **TEST L** ***Q$Ochered,2,Met4*** и сразу попадает в очередь (оператор **QUEUE** ***Ochered***), но он пройдёт через неё перед другими. Если в *Очереди* ***Ochered*** окажется несколько льготников, то автоматически будет образована своя внутренняя очередь льготников и они будут пропускаться в зависимости от времени прихода. В *Окне сохраняемых величин* ***Savevalues Window*** в динамике можно наблюдать *Сохраняемые величины* ***Klient_Lgot*** и ***Klient_No***.

Модель 2.11

Условия задачи: Условия задачи дополнительно изменены по сравнению с предыдущей версией модели (Модель 2.10). Так же, как и в предыдущей модели, новый клиент оценивает длину очереди и если она больше 2 человек, уходит, а если меньше — встаёт в очередь. Некоторые клиенты имеют право на обслуживание вне очереди и длина общей очереди их не волнует. Но может образовываться льготная очередь и её длина важна для льготников. В остальном условия — те же, т. е. клиенты делятся на три типа — клиенты, желающие только постричься, желающие только побриться и желающие постричься и побриться. Клиенты появляются через каждые 20±5 минут. Желающих постричься 60% от общего числа клиентов и из них 20% желают

также побриться, желающих только побриться 40% от общего числа клиентов. Продолжительность стрижки 25±10 минут, продолжительность бритья 10±5 минуты. Необходима статистика очереди для всех клиентов. Моделируется рабочий день (480 минут).

```
*=============================================================================
* GPSS World - barber11.gps
*-----------------------------------------------------------------------------
* Парикмахерская с 1 креслом
*-----------------------------------------------------------------------------
* Часть клиентов только стрижётся, часть только бреется, часть стрижётся и бреется.
* Новый клиент оценивает длину очереди и уходит, если она велика
* Часть клиентов имеют льготы по обслуживанию вне очереди и они остаются, не
* зависимо от длины очереди и проходят, минуя её. При этом может образоваться
* очередь льготников ("Кто последний без очереди?"). Новый льготник также оценивает
* свою льготную очередь и уходит, если она велика.
*=============================================================================
; Приход клиента и вход его в очередь (общую или льготную)
;-----------------------------------------------------------------------------
       GENERATE     20,5                      ; Клиент пришёл
       TRANSFER     0.1,Met7,Met6             ; Льготник или нет?
; Льготная очередь
Met6   PRIORITY     10                        ; Право на льготное обслуживание
       TEST L       Q$Ochered_Lgot,2,Met4     ; Уход, если льготная очередь >2 чел.
       QUEUE        Ochered_Lgot              ; Встал в льготную очередь
       TRANSFER     ,Met8                     ; Льготник пошёл обслуживаться
; Общая очередь
Met7   TEST L       Q$Ochered,2,Met4          ; Уход, если общая очередь >2 чел.
       QUEUE        Ochered                   ; Встал в общую очередь
;-----------------------------------------------------------------------------
; Обслуживание клиента
;-----------------------------------------------------------------------------
Met8   SEIZE        Kreslo           ; Начал обслуживаться
       TEST L       PR,10,Met10      ; Из какой очереди прошёл клиент?
       DEPART       Ochered          ; Покинул общую очередь
       TRANSFER     ,Met11           ;
Met10  DEPART       Ochered_Lgot     ; Покинул льготную очередь
Met11  TRANSFER     0.4,Met1,Met2    ; 40% выбирают бритьё (Met2),60% стрижку (Met1)
Met1   ADVANCE      25,10            ; Стрижётся
       TRANSFER     0.2,Met3,Met2    ; 20% выбирают дополнительно бритьё (Met2)
Met2   ADVANCE      10,5             ; Бреется
Met3   RELEASE      Kreslo           ; Закончил обслуживание
       TRANSFER     ,Met5            ; Пошёл к выходу
Met4   SAVEVALUE    Klient_No+,1     ; Подсчёт клиентов, не ставших ждать
Met5   TERMINATE    0                ; Клиент ушёл
; Таймер
       GENERATE     480              ; Рабочий день окончился
       TERMINATE    1                ; Моделирование окончилось
       START        1                ; Число дней моделирования
```

Пояснения. Как и в предыдущей программе (Модель 2.10), в данной версии моделируется возможность обслуживания некоторых клиентов вне очереди. Для этого используется оператор задания приоритета транзакта **PRIORITY**, на который направляется 10% вошедших клиентов (такая задана пропорция льготников от общего числа клиентов) с помощью оператора **TRANSFER** ***0.1,Met6,Met7***. Сразу после оператора ***Met6*** **PRIORITY** ***10*** стоит оператор **TEST L** ***Q$Ochered_Lgot,2,Met4***, с помощью которого льготник оценивает длину льготной очереди (через СЧА ***Q$Ochered_Lgot***). При этом нет необходимости другими способами подсчитывать количество льготников, так как вся необходимая статистика будет

автоматически собираться оператором **QUEUE *Ochered_Lgot***. Состояние общей ***Ochered*** и льготной ***Ochered_Lgot*** очередей можно в динамике отслеживать с помощью *Окна очередей* ***Queues Window***.

Модель 2.12

Условия задачи: Условия задачи аналогичны предыдущей версии модели (Модель 2.11). Так же, как и в предыдущей модели, новый клиент оценивает длину очереди и если она больше 2 человек, уходит, а если меньше — встаёт в очередь. Некоторые клиенты имеют право на обслуживание вне очереди и длина общей очереди их не волнует. Но может образовываться льготная очередь и её длина важна для льготников. В остальном условия — те же, т. е. клиенты делятся на три типа — клиенты, желающие только постричься, желающие только побриться и желающие постричься и побриться. Клиенты появляются через каждые 20±5 минут. Желающих постричься 60% от общего числа клиентов и из них 20% желают также побриться, желающих только побриться 40% от общего числа клиентов. Продолжительность стрижки 25±10 минут, продолжительность бритья 10±5 минуты. Необходима статистика очереди для всех клиентов. Моделируется рабочий день (480 минут). Дополнительным требованием является необходимость сбора информации о клиентах разного типа — льготниках, не льготниках, обслуживаемых в данный момент, ушедших без обслуживания. Моделируется рабочий день (480 минут).

```
*=============================================================================
* GPSS World - barber12.gps
*-----------------------------------------------------------------------------
* Парикмахерская с 1 креслом
*-----------------------------------------------------------------------------
* Часть клиентов только стрижётся, часть только бреется, часть стрижётся и бреется.
* Новый клиент оценивает длину очереди и уходит, если она велика
* Часть клиентов имеют льготы по обслуживанию вне очереди и они остаются, не
* зависимо от длины очереди и проходят, минуя её. При этом может образоваться
* очередь льготников ("Кто последний без очереди?"). Новый льготник также оценивает
* свою льготную очередь и уходит, если она велика.
* Для получения текущей информации о состоянии модели используются сохраняемые
* величины (SAVEVALUE):
* Nomer_Nov - номер нового клиента
* Nomer_Lgot - номер клиента, вошедшего в льготную очередь
* Nomer_General - номер клиента, вошедшего в общую очередь
* Nomer_Kreslo - номер обслуживаемого клиента
* Nomer_For - номер ушедшего без обслуживания клиента
*=============================================================================
; Приход клиента и вход его в очередь (общую или льготную)
;-----------------------------------------------------------------------------
       GENERATE     20,5                        ; Клиент пришёл
       SAVEVALUE    Nomer_Nov,XN1               ; Номер нового клиента
       TRANSFER     0.1,Met7,Met6               ; Льготник или нет?
; Льготная очередь
Met6   PRIORITY     10                          ; Право на льготное обслуживание
       TEST L       Q$Ochered_Lgot,2,Met4       ; Уход, если льготная очередь >2 чел.
       SAVEVALUE    Nomer_Lgot,XN1              ; Клиент, вставший в льготную очередь
       QUEUE        Ochered_Lgot                ; Встал в льготную очередь
       TRANSFER     ,Met8                       ; Льготник пошёл обслуживаться
; Общая очередь
Met7   TEST L       Q$Ochered,2,Met4            ; Уход, если общая очередь >2 чел.
       SAVEVALUE    Nomer_General,XN1           ; Клиент, вставший в общую очередь
```

```
       QUEUE        Ochered                    ; Встал в общую очередь
;-------------------------------------------------------------------------------
; Обслуживание клиента
;-------------------------------------------------------------------------------
Met8   SEIZE  Kreslo ; Начал обслуживаться
       SAVEVALUE    Nomer_Kreslo,XN1  ; Номер обслуживаемого клиента
       TEST L       PR,10,Met10       ; Из какой очереди прошёл клиент?
       DEPART       Ochered           ; Покинул общую очередь
       TRANSFER     ,Met11            ;
Met10  DEPART       Ochered_Lgot      ; Покинул льготную очередь
Met11  TRANSFER     0.4,Met1,Met2     ; 40% выбирают бритьё (Met2),60% стрижку (Met1)
Met1   ADVANCE      25,10             ; Стрижётся
       TRANSFER     0.2,Met3,Met2     ; 20% выбирают дополнительно бритьё (Met2)
Met2   ADVANCE      10,5              ; Бреется
Met3   RELEASE      Kreslo            ; Закончил обслуживание
       TRANSFER     ,Met5             ; Пошёл к выходу
Met4   SAVEVALUE    Summa_For+,1      ; Подсчёт клиентов, не ставших ждать
       SAVEVALUE    Nomer_For,XN1     ; Номер клиента, не ставшего ждать
Met5   TERMINATE    0                 ; Клиент ушёл
; Таймер
       GENERATE     480    ; Рабочий день окончился
       TERMINATE    1      ; Моделирование окончилось
;      START        1      ; Число дней моделирования
```

Пояснения. По сравнению с предыдущей моделью (Модель 2.11) введены дополнительные *Сохраняемые величины* для просмотра текущей информации о том, где какой клиент находится в данный момент (например, при пошаговом моделировании). С помощью оператора **SAVEVALUE *Nomer_Nov,XN1*** запоминается номер транзакта, вошедшего в парикмахерскую. С помощью оператора **SAVEVALUE *Nomer_Lgot,XN1*** запоминается номер транзакта, вставшего в льготную очередь. С помощью оператора **SAVEVALUE *Nomer_General,XN1*** запоминается номер транзакта, вставшего в общую очередь. С помощью оператора **SAVEVALUE *Nomer_Kreslo,XN1*** запоминается номер транзакта, севшего в кресло. С помощью оператора **SAVEVALUE *Nomer_For,XN1*** запоминается номер транзакта, покинувшего парикмахерскую из-за длинных очередей. В каждом случае применяется СЧА ***XN1***, означающий "номер активного транзакта", значение которого и записывается в соответствующую *Сохраняемую величину*, когда этот активный транзакт в соответствующем операторе находится. С помощью оператора **SAVEVALUE *Summa_For,XN1*** подсчитывается число транзактов, покинувших парикмахерскую без обслуживания из-за длинных очередей (как общей, так и льготной). Состояние общей ***Ochered*** и льготной ***Ochered_Lgot*** очередей можно в динамике отслеживать в *Окне очередей* ***Queues Window***, а содержимое *Сохраняемых величин* ***Nomer_Nov***, ***Nomer_Lgot***, ***Nomer_General***, ***Nomer_Kreslo***, ***Nomer_For*** можно отслеживать в *Окне сохраняемых величин* ***Savevalues Window***.

Модель 2.13

Условия задачи: Клиент заходит в парикмахерскую через каждые 25±20 минут. Время обслуживания составляет 25±5 минут. В парикмахерской 2 кресла. Статистика очереди не ведётся.

```
*=========================================================================================
* GPSS World - barber02N.gps
*-----------------------------------------------------------------------------------------
* Парикмахерская с несколькими креслами
*-----------------------------------------------------------------------------------------
* Клиент входит, обслуживается и выходит
*=========================================================================================
Kreslo STORAGE       2          ; Число кресел
;---------------------------------
       GENERATE      25,20      ; Клиент пришёл
       ENTER         Kreslo,1   ; Пошёл обслуживаться
       ADVANCE       25,5       ; Обслуживается
       LEAVE         Kreslo,1   ; Закончил обслуживание
       TERMINATE     1          ; Клиент ушёл
```

Пояснения. Модель аналогична модели (Модель 2.2) с одним креслом, но отличается от неё наличием 2 кресел. Поэтому имеется описание *Многоканального устройства* с помощью оператора ***Kreslo*** **STORAGE** ***2*** . В нём слово **STORAGE** — оператор, слово ***Kreslo*** — название *Многоканального устройства*, число ***2*** определяет количество каналов (в данном случае — кресел). Оператор **ENTER** ***Kreslo,1*** даёт команду активному транзакту (клиенту) занять одно из кресел (стоит число ***1***) в *Многоканальном устройстве* ***Kreslo***. Если есть свободное кресло, то оно занимается. Оператор **ADVANCE** ***25,5*** задаёт продолжительность обслуживания клиента (например, стрижки) в течение 25±5 минут. После окончания обслуживания очередного клиента с помощью оператора **LEAVE** ***Kreslo,1*** он освобождает одно из кресел (стоит число ***1***) в *Многоканальном устройстве* ***Kreslo***. Контролировать в динамике (например, при пошаговом моделировании) состояние *Многоканального устройства* ***Kreslo*** можно в *Окне многоканальных устройств* ***Storages Window***.

Модель 2.14

Условия задачи: Клиент заходит в парикмахерскую через каждые 25±20 минут. Время обслуживания составляет 25±5 минут. В парикмахерской 2 кресла. Ведётся статистика очереди.

```
*=========================================================================================
* GPSS World - barber03N.gps
*-----------------------------------------------------------------------------------------
* Парикмахерская с несколькими креслами
*-----------------------------------------------------------------------------------------
* Клиент входит, обслуживается и выходит
*=========================================================================================
Kreslo STORAGE       2          ; Число кресел
;---------------------------------
       GENERATE      25,20      ; Клиент пришёл
       QUEUE         Ochered    ; Встал в очередь
       ENTER         Kreslo,1   ; Пошёл обслуживаться
       DEPART        Ochered    ; Покинул очередь
       ADVANCE       25,5       ; Обслуживается
       LEAVE         Kreslo,1   ; Закончил обслуживание
       TERMINATE     1          ; Клиент ушёл
```

Пояснения. Модель аналогична предыдущей модели (Модель 2.13), но отличается от неё наличием операторов, связанных с *Очередью*: **QUEUE** ***Ochered*** — включает клиента в очередь под названием ***Ochered***,

DEPART — исключает клиента из очереди по тем же названием ***Ochered***. Теперь можно наблюдать текущее состояние очереди в *Окне очередей* ***Queues Window***. Контролировать в динамике (например, при пошаговом моделировании) состояние *Многоканального устройства* ***Kreslo*** можно в *Окне многоканальных устройств* ***Storages Window***.

Модель 2.15

Условия задачи: Клиент заходит в парикмахерскую через каждые 10±2 минут. Время обслуживания составляет 25±5 минут. В парикмахерской 2 кресла. Ведётся статистика очереди. Моделируется рабочий день (480 минут).

```
*=============================================================================
* GPSS World - barber04N.gps
*-----------------------------------------------------------------------------
* Парикмахерская с несколькими креслами
*-----------------------------------------------------------------------------
* Клиент входит, обслуживается и выходит
*=============================================================================
Kreslo STORAGE      2         ; Число кресел
;--------------------------------
       GENERATE     10,2      ; Клиент пришёл
       QUEUE        Ochered   ; Встал в очередь
       ENTER        Kreslo,1  ; Пошёл обслуживаться
       DEPART       Ochered   ; Покинул очередь
       ADVANCE      25,5      ; Обслуживается
       LEAVE        Kreslo,1  ; Закончил обслуживание
       TERMINATE    0         ; Клиент ушёл
; Таймер
       GENERATE     480       ; Рабочий день окончился
       TERMINATE    1         ; Моделирование окончилось
       START        1         ; Число дней моделирования
```

Пояснения. Модель аналогична предыдущей модели (Модель 2.14), но отличается от неё наличием "таймера" в нижней части программы, как в модели 4. Контролировать в динамике (например, при пошаговом моделировании) состояние *Многоканального устройства* ***Kreslo*** можно в *Окне многоканальных устройств* ***Storages Window***.

Модель 2.16

Условия задачи: Клиент заходит в парикмахерскую через каждые 12±5 минут. Время обслуживания составляет 20±5 минут. Из числа вошедших 60% стригутся, а 40% ещё и бреются. В парикмахерской 2 кресла. Ведётся статистика очереди. Моделируется рабочий день (480 минут).

```
*=============================================================================
* GPSS World - barber06N.gps
*-----------------------------------------------------------------------------
* Парикмахерская с несколькими креслами
*-----------------------------------------------------------------------------
* Один поток клиентов, но некоторые из них только стригутся, а некоторые
* стригутся и бреются
*=============================================================================
Kreslo STORAGE      2             ; Число кресел
;-----------------------------------------------
       GENERATE     12,5          ; Клиент пришёл
       QUEUE        Ochered       ; Встал в очередь
       ENTER        Kreslo,1      ; Пошёл обслуживаться
```

```
       DEPART       Ochered       ; Покинул очередь
       ADVANCE      20,5          ; Стрижётся
       TRANSFER     0.4,Met1,Met2 ; 40% выбирают бритьё (Met2),60% стрижку (Met1)
Met2   ADVANCE      15,3          ; Бреется
Met1   LEAVE        Kreslo,1      ; Закончил обслуживание
       TERMINATE    0             ; Клиент ушёл
; Таймер
       GENERATE     480           ; Рабочий день окончился
       TERMINATE    1             ; Моделирование окончилось
       START        1             ; Число дней моделирования
```

Пояснения. Модель аналогична модели (Модель 2.6), но отличается от неё использованием *Многоканального устройства*. Контролировать в динамике (например, при пошаговом моделировании) состояние *Многоканального устройства* ***Kreslo*** можно в *Окне многоканальных устройств* ***Storages Window***.

Модель 2.17

Условия задачи: Условия задачи соответствуют условиям модели (Модель 2.12), но предполагается наличие 2 кресел и других времён обслуживания.

```
*=============================================================================
* GPSS World - barber12N.gps
*-----------------------------------------------------------------------------
* Парикмахерская с несколькими креслами
*-----------------------------------------------------------------------------
* Часть клиентов только стрижётся, часть только бреется, часть стрижётся и бреется.
* Новый клиент оценивает длину очереди и уходит, если она велика
* Часть клиентов имеют льготы по обслуживанию вне очереди и они остаются, не
* зависимо от длины очереди и проходят, минуя её. При этом может образоваться
* очередь льготников ("Кто последний без очереди?"). Новый льготник также оценивает
* свою льготную очередь и уходит, если она велика.
* Для получения текущей информации о состоянии модели используются сохраняемые
* величины (SAVEVALUE):
* Nomer_Nov - номер нового клиента
* Nomer_Lgot - номер клиента, вошедшего в льготную очередь
* Nomer_General - номер клиента, вошедшего в общую очередь
* Nomer_Kreslo - номер обслуживаемого клиента
* Nomer_For - номер ушедшего без обслуживания клиента
*=============================================================================
; Приход клиента и вход его в очередь (общую или льготную)
;-----------------------------------------------------------------------------
Kreslo STORAGE      2                     ; Число кресел
;---------------------------------
       GENERATE     20,5                  ; Клиент пришёл
       SAVEVALUE    Nomer_Nov,XN1         ; Номер нового клиента
       TRANSFER     0.1,Met7,Met6         ; Льготник или нет?
; Льготная очередь
Met6   PRIORITY     10                    ; Право на льготное обслуживание
       TEST L       Q$Ochered_Lgot,2,Met4 ; Уход, если льготная очередь >2 чел.
       SAVEVALUE    Nomer_Lgot,XN1        ; Клиент, вставший в льготную очередь
       QUEUE        Ochered_Lgot          ; Встал в льготную очередь
       TRANSFER     ,Met8                 ; Льготник пошёл обслуживаться
; Общая очередь
Met7   TEST L       Q$Ochered,2,Met4      ; Уход, если общая очередь >2 чел.
       SAVEVALUE    Nomer_General,XN1     ; Клиент, вставший в общую очередь
       QUEUE        Ochered               ; Встал в общую очередь
;-----------------------------------------------------------------------------
; Обслуживание клиента
;-----------------------------------------------------------------------------
Met8   ENTER        Kreslo            ; Начал обслуживаться
       SAVEVALUE    Nomer_Kreslo,XN1  ; Номер обслуживаемого клиента
       TEST L       PR,10,Met10       ; Из какой очереди прошёл клиент?
```

```
       DEPART       Ochered           ; Покинул общую очередь
       TRANSFER     ,Met11            ;
Met10  DEPART       Ochered_Lgot      ; Покинул льготную очередь
Met11  TRANSFER     0.4,Met1,Met2     ; 40% выбирают бритьё (Met2),60% стрижку (Met1)
Met1   ADVANCE      25,10             ; Стрижётся
       TRANSFER     0.2,Met3,Met2     ; 20% выбирают дополнительно бритьё (Met2)
Met2   ADVANCE      10,5              ; Бреется
Met3   LEAVE        Kreslo            ; Закончил обслуживание
       TRANSFER     ,Met5             ; Пошёл к выходу
Met4   SAVEVALUE    Summa_For+,1      ; Подсчёт клиентов, не ставших ждать
       SAVEVALUE    Nomer_For,XN1     ; Номер клиента, не ставшего ждать
Met5   TERMINATE    0                 ; Клиент ушёл
; Таймер
       GENERATE     480               ; Рабочий день окончился
       TERMINATE    1                 ; Моделирование окончилось
;      START        1                 ; Число дней моделирования
```

Пояснения. Модель аналогична модели (Модель 2.12), но отличается от неё использованием *Многоканального устройства*. Контролировать в динамике (например, при пошаговом моделировании) состояния *Многоканального устройства* **`Kreslo`** можно в *Окне многоканальных устройств* ***Storages Window***.

2.2 Модель учебного процесса в вузе

2.2.1 Общая характеристика рассматриваемого примера

Рассматривается сравнительно простой пример сдачи студентами устного экзамена.

Моделируемый процесс протекает следующим образом.

Студенты приходят на экзамен и по очереди входят в аудиторию. Количество билетов может быть меньше числа студентов в группе и поэтому может образоваться очередь за билетами. Имеется список студентов, которые входят в аудиторию случайным образом. Также случайным образом они берут билеты. Каждый билет включает два вопроса различной сложности. Затем студенты идут готовиться в помещение, достаточное для того, чтобы вместить всех, получивших билеты. После подготовки, зависящей от сложности билета и подготовленности студента, студенты выстраиваются в очередь к преподавателю. Они садятся поочерёдно к преподавателю и рассказывают свои вопросы. Продолжительность этого различна для различных студентов. Качество ответа оценивается оценкой, которая зависит от подготовленности студента (но с точки зрения наблюдения является величиной случайной). Когда студент сел к преподавателю его билет освобождается и может быть передан очередному студенту.

Интерес могут представлять различные характеристики процесса: время ожидания студентами билетов (если билетов меньше, чем студентов), общее время экзамена, число студентов, сдавших экзамен на 2, 3, 4 и 5 и др.

С точки зрения GPSS в этом примере присутствуют *Очереди*, *Обслуживающие устройства*, *Многоканальные устройства*. Студенты могут накап-

ливаться перед *Обслуживающими устройствами* и *Многоканальными устройствами*.

При написании программы использовались наиболее простые программистские решения. Для отладки применялось пошаговое моделирование с использованием нескольких окон GPSS.

2.2.2 Разработка программной модели на языке GPSS World

Программная модель на языке GPSS World представлена ниже.

```
*=============================================================================
* GPSS World - Examen10.gps
*-----------------------------------------------------------------------------
* Устный экзамен
*-----------------------------------------------------------------------------
* Условия.
* Имеется список группы, стопка билетов. В каждом билете 2 вопроса. Все вопросы
* различной сложности (сложность рапределяется случайным образом по билетам, под
* сложностью понимается среднее время подготовки вопроса). Количество билетов
* может быть меньше количества студентов.
* Описание процесса.
* Входит студент (имя - случайным образом). Если есть очередь за билетами, то ждёт.
* Когда подходит очередь брать билет, студент его берёт (случайный номер).
* Если все билеты уже были разобраны, то студент получает тот билет, который
* был сдан только что ответившим студентом.
* Студент идёт готовиться. Время подготовки зависит от сложности билета.
* Подготовленный студент ждёт очереди к преподавателю для ответа.
* Когда очередь к преподавателю подходит, студент идёт отвечать.
* По результатам ответа студент получает оценку (от 2 до 5) случайным образом.
* Результаты заносятся в ведомость: номер студента, фамилия студента, номер билета,
* оценка
*=============================================================================
KolBil   EQU         15                                ; Количество билетов
KolSt    EQU         20                                ; Количество студентов
BiletPak STORAGE     15                                ; Стопка билетов для выбора
Bilet    MATRIX      ,15,3                             ; Список билетов(по два вопроса)
Grup     MATRIX      ,20,2                             ; Список группы
Vedom    MATRIX      ,20,4                             ; Ведомость
         INCLUDE     "Gruppa.txt"                      ; Список группы исходный
         INITIAL     X$SvobBilet,0                     ; Номер освободившегося билета
         INITIAL     X$FixKolBil,0                     ; Количество выданных билетов
*-----------------------------------------------------------------------------
* Генерирование сложности вопросов в билетах
*-----------------------------------------------------------------------------
         GENERATE    ,,,1                              ; Транзакты для билетов
         ASSIGN      Ind,0                             ; Обнуление счётчика цикла
Met30    ASSIGN      Ind+,1                            ; Счётчик цикла билетов
         TEST LE     P$Ind,KolBil,Met20                ; Сравнение с максимумом цикла
         MSAVEVALUE  Bilet,P$Ind,1,(DUniform(1,5,15)) ; Сложность вопроса 1
         MSAVEVALUE  Bilet,P$Ind,2,(DUniform(2,5,15)) ; Сложность вопроса 2
         TRANSFER    ,Met30                            ; Назад в цикл задания сложности
Met20    TERMINATE   0                                 ; Выход из цикла сложности
*-----------------------------------------------------------------------------
* Появление студента и его идентификация по списку группы
*-----------------------------------------------------------------------------
         GENERATE    2,1,,KolSt                ; Студент вошёл
         SAVEVALUE   Stud+,1                   ; Номер студента по порядку входа
; Цикл случайной проверки, был ли уже студент с такой фамилией
Met10    ASSIGN      Num,(DUniform(3,1,KolSt))  ; Случайное число для студента
         TEST E      MX$Grup(P$Num,2),0,Met10   ; Пришёл ли уже такой студент?
; Окончание цикла проверки и определения фамилии студента
         ASSIGN      Name,MX$Grup(P$Num,1)      ; Новый студент: имя из списка
         MSAVEVALUE  Grup,P$Num,2,X$Stud        ; Вписать в матрицу, что пришёл
         MSAVEVALUE  Vedom,P$Num,1,P$Num        ; В ведомость номер студента
```

```
         MSAVEVALUE  Vedom,P$Num,2,P$Name       ; В ведомость имя студента
; Окончание выявления фвмилии студента
         SAVEVALUE   FixName,P$Name             ; Имя студента для визуализации
         SAVEVALUE   FixNum,P$Num               ; Номер студента для визуализации
*-------------------------------------------------------------------------------
* Получение билета и подготовка ответа
*-------------------------------------------------------------------------------
         QUEUE       BiletQ                     ; Занимает очередь за билетом
         ENTER       BiletPak                   ; Пошёл за билетом
         DEPART      BiletQ                     ; Покинул очередь за билетом
; Получение билета, освободившегося после очередного сдавшего экзамен студента
         TEST NE     X$SvobBilet,0,Met40
         ASSIGN      Bil,X$SvobBilet            ; Свободный билет
         TRANSFER    ,Met60
; Получение билета (определение номера билета случайным образом) в цикле
Met40    ASSIGN      Bil,(DUniform(4,1,KolBil)) ; Случайное число для билета
         TEST E      MX$Bilet(P$Bil,3),0,Met40  ; Взят ли уже такой билет?
; Окончание цикла определения номера выданного билета
Met60    SAVEVALUE   FixKolBil+,1               ; Количество выданных билетов
         MSAVEVALUE  Bilet,P$Bil,3,P$Num        ; Вписать студента с этим билетом
         MSAVEVALUE  Vedom,P$Num,3,P$Bil        ; В ведомость записан № билета
         SAVEVALUE   FixBil,P$Bil               ; Номер билета для визуализации
; Подготовка вопросов
         SAVEVALUE   Resp1,MX$Bilet(P$Bil,1)
         ADVANCE     Resp1,(0.1#Resp1)          ; Готовит вопрос 1
         SAVEVALUE   Resp2,MX$Bilet(P$Bil,2)
         ADVANCE     Resp2,(0.1#Resp2)          ; Готовит вопрос 2
         LEAVE       BiletPak                   ; Студент готов
         QUEUE       Otvet                      ; Студент ждёт очереди отвечать
*-------------------------------------------------------------------------------
* Изложение ответа преподавателю
*-------------------------------------------------------------------------------
         SEIZE       Prepod                     ; Студент пошёл отвечать
         DEPART      Otvet                      ; Студент покинул очередь отвечать
         SAVEVALUE   SvobBilet,P$Bil            ; Номер освободившегося билета
         MSAVEVALUE  Bilet,P$Bil,3,0            ; Студент сдал билет
         ADVANCE     5,2                        ; Отвечает на вопрос 1 билета
         ADVANCE     5,2                        ; Отвечает на вопрос 2 билета
         ASSIGN      Ocenka,(DUniform(5,2,5))   ; Оценка студенту в зачётку
         MSAVEVALUE  Vedom,P$Num,4,P$Ocenka     ; Оценка студенту в ведомость
         ADVANCE     1                          ; Время простановки оценок
         RELEASE     Prepod                     ; Студент закончил ответ
         TERMINATE   1                          ; Студент ушёл
*-------------------------------------------------------------------------------
```

Здесь реализованы некоторые операции, которые в реальном объекте (на экзамене) вынесены за пределы его функционирования.

Программа разделена на следующие фрагменты.

Первый фрагмент является описание ряда переменных и констант.

Второй фрагмент "**Генерирование сложности вопросов в билетах**" выполняет функцию задания сложности билетов случайным образом. В реальности для каждой учебной дисциплины билеты разрабатываются заранее и можно определить тем или иным способом сложность каждого из вопросов билета (например, с помощью тестирования на студентах или по результатам предыдущих экзаменов).

Третий фрагмент "**Появление студента и его идентификация по списку группы**" отражает процесс прихода студентов на экзамен случайным образом. Имеется список группы и каждый пришедший студент отмечается в нём с получением фамилии и других данных.

Четвёртый фрагмент "**Получение билета и подготовка ответа**" характеризует случайный характер получения билета и различия в требуемом времени подготовки. Здесь отражается также факт меньшего количества билетов, чем студентов, что может создавать очередь за билетами. В этом случае каждый ожидающий билета студент получит тот билет, который отдаст студент, ответивший на него преподавателю. Этот процесс отражён в модели.

Пятый фрагмент "**Изложение ответа преподавателю**" имитирует занятие преподавателя и получении оценки. В данном случае оценка генерируется случайным образом, но в будущем можно будет увязывать её с оценками по другим учебным дисциплинам.

2.3 Модель движения автобуса по маршруту

Имитационная модель сложного объекта должна отражать много деталей и поэтому часто не может быть представлена в простом виде: её необходимо структурировать и выделять обособленные фрагменты, внутри которых "развиваются" собственные события. Специфика имитационного моделирования заключается в том, что работа этих фрагментов должна происходить под действием "своих" транзактов, имеющих, возможно, различные физический смысл, и синхронизированных между собой в процессе перемещения. В настоящее время отсутствуют формальные правила создания таких моделей, которые можно было бы использовать без специального продумывания — как это, например, делается при создании автоматизированных информационных систем и в некоторых других случаях, когда возможно применение средств автоматизированного проектирования программ. В практике имитационного программирования приходится проявлять настоящую изобретательность, чтобы отразить ограниченными средствами многообразие жизни.

Ниже приводится текст с пояснениями программы, имитирующей движение автобуса по кольцевому маршруту. Целью такого исследования может быть, например, загрузка автобуса, возможная прибыль, оценка необходимости введения дополнительных автобусов.

Листинг программ содержит две большие части — фрагмент программы, связанный с состоянием автобуса (**Движение автобуса**), и фрагмент программы, связанный с состоянием пассажиров на остановках (**Потоки пассажиров**).

Для удобства перенастройки программы при изменении параметров большинство числовых значений задается с помощью переменных в начальном фрагменте программы (**Исходные данные**).

Суть моделируемого процесса описана в головке программы и заключается в том, что небольшой автобус движется по кольцевому маршруту, состоящему из 5 остановок. При этом различают начальную и конечную остановки (они располагаются, возможно, рядом) и промежуточные.

```
------------------------------------------------------------------------------
; GPSS World: buso-01.gps
;------------------------------------------------------------------------------
; Движение автобуса по кольцевому маршруту
;------------------------------------------------------------------------------
; Автобус движется по маршруту в одном направлении по кольцу. Имеется 5 остановок,
; из которых одна - начальная, одна - конечная и три - промежуточных.
; На каждую остановку, кроме конечной, подходят пассажиры. Когда автобус
; останавливается, то вначале все желающие выходят, а затем входит некоторое
; количество пассажиров, но не более, чем есть на остановке и не более, чем может
; вместить автобус. На первой остановке только входят, а на конечной - только
; выходят. Время входа и выхода каждого пассажира учитывается.
;------------------------------------------------------------------------------
; Особенности программы.
; Реализована последовательная работа, когда число остановок отражается
; непосредственно в программе.
;==============================================================================
; Исходные данные
;------------------------------------------------------------------------------
Bus     STORAGE     20           ; Автобус на 20 мест
lok     EQU         20           ; Вместимость автобуса
tmp1    EQU         0.1          ; Время выхода 1 пассажира
tmp2    EQU         0.15         ; Время входа 1 пассажира
tmp3    EQU         6            ; Время движения между остановками
tmp4    EQU         5            ; Время выезда на маршрут с базы
tmp5    EQU         1            ; Время появления пассажира
        INITIAL     X$HaltNum,0  ; Номер текущей остановки
        INITIAL     X$KvantEL,0  ; Число желающих выйти
        INITIAL     X$KvantEL2,0 ; Число желающих выйти
        INITIAL     X$ELBus,0    ; Разрешение на выход
        INITIAL     X$ELBusFin,0 ; Завершение выхода
        INITIAL     X$ENBus,0    ; Разрешение на вход
        INITIAL     X$KvantEN,0  ; Число вошедших
        INITIAL     X$PluVet,0   ; Разрешение на отъезд от ост.
        INITIAL     X$ENBus1,0   ; Число вошедших
        INITIAL     X$ENBus2,0   ; Число вошедших
        INITIAL     X$ENBus3,0   ; Число вошедших
        INITIAL     X$ENBus4,0   ; Число вошедших
;------------------------------------------------------------------------------
; Движение автобуса
;------------------------------------------------------------------------------
        GENERATE    ,,,1             ; Автобус вышел на маршрут
        ASSIGN      1,0              ; Номер остановки обнулён
        ADVANCE     tmp4,(0.2#tmp4)  ; Выезд из гаража на маршрут
        TRANSFER    ,t1100           ; Переход на маршрут на ост.1
t1000   ADVANCE     tmp3,(0.2#tmp3)  ; Поехал к остановке
t1100   ASSIGN      1+,1             ; Автобус прибыл на остановку
        SAVEVALUE   HaltNum,P1       ; Номер остановки зафиксирован
;----------Проверка типа остановки-----------------------------------------------
        TEST NE     P1,1,t1800       ; Начальная остановка?
        TEST NE     P1,5,t1900       ; Конечная остановка?
;----------Промежуточная остановка-----------------------------------------------
        SAVEVALUE   KvantEL,(DUniform(2,0,S$Bus))    ; Число желающих выйти
        SAVEVALUE   KvantEL2,X$KvantEL               ; Счётчик желающих выйти
        TEST NE     X$KvantEL,0,t1500                ; Есть ли на выход?
        SAVEVALUE   ELBus,1                          ; Разрешение на выход
;---------------------------------------Вход
        TEST E      X$ELBusFin,1              ; Задержка на время выхода
t1500   SAVEVALUE   ELBus,0                   ; Сброс разрешения на выход
        SAVEVALUE   KvantEL,0                 ; Сброс числа пассажиров на выход
        SAVEVALUE   KvantEL2,0                ; Сброс числа пассажиров на выход
        SAVEVALUE   ELBusFin,0                ; Сброс задержки на время выхода

        TEST NE     (Q*1#(lok-S$Bus)),0,t1700  ; Можно на вход?
        SAVEVALUE   ENBus,1                   ; Разрешение на вход
        TEST E      X$PluVet,1                ; Задержка до разрешения движения
;---------------------------------------Сброс параметров остановки
t1700   SAVEVALUE   ENBus,0                   ; Сброс разрешения на вход
```

```
        SAVEVALUE   PluVet,0                    ; Сброс разрешения на движение
        TRANSFER    ,t1000                      ; Движение к следующей остановке
;----------Начальная остановка 1-------------------------------------------------
t1800   SAVEVALUE   ENBus,1                     ; Разрешение на вход
        TEST E      X$PluVet,1                  ; Задержка движения
        SAVEVALUE   ENBus,0                     ; Сброс разрешения на вход
        SAVEVALUE   PluVet,0                    ; Сброс разрешения на вход
        TRANSFER    ,t1000                      ; Поехал к следующей остановке
;----------Конечная остановка 5--------------------------------------------------
t1900   SAVEVALUE   KvantEL,S$Bus               ; Все на выход на кон.
        SAVEVALUE   KvantEL2,X$KvantEL          ; Счётчик желающих выйти на кон.
        SAVEVALUE   ELBus,1                     ; Разрешение на выход на кон.
        TEST E      X$ELBusFin,1                ; Задержка движения на кон.
        ASSIGN      P1,0                        ; Сброс идикатора номера остановки
        SAVEVALUE   KvantEL,0                   ; Сброс числа пассажиров на выход
        SAVEVALUE   KvantEL2,0                  ; Сброс числа пассажиров на выход
        SAVEVALUE   ELBus,0                     ; Сброс разрешения на выход
        SAVEVALUE   ENBus,0                     ; Сброс разрешения на вход
        SAVEVALUE   PluVet,0                    ; Сброс разрешения на движение
        SAVEVALUE   ENBus1,0                    ; Сброс счётчика вошедших на ост.1
        SAVEVALUE   ENBus2,0                    ; Сброс счётчика вошедших на ост.2
        SAVEVALUE   ENBus3,0                    ; Сброс счётчика вошедших на ост.3
        SAVEVALUE   ENBus4,0                    ; Сброс счётчика вошедших на ост.4
        ASSIGN      1,0                         ; Сброс счётчика остановок
        TRANSFER    ,t1000                      ; Возврат на остановку 1 по кольцу
;================================================================================
; Потоки пассажиров
;================================================================================
; Остановка 1 ***********************************************************************
        GENERATE    tmp5,(0.5#tmp5)             ; Пассажир пришёл 1
        ASSIGN      2,1                         ; Отметка пассажира 1
        QUEUE       1                           ; Встал в очередь 1
        TEST E      (X$ENBus#X$HaltNum),1       ; Разрешение на вход 1-;---------------
--------------------------- Вход-начало-
        ENTER       Bus,1                       ; Вход в автобус 1
        DEPART      1                           ; Освобождение очереди 1
        SAVEVALUE   ENBus1+,1                   ; Подсчёт числа вошедших 1
        TEST E      (Q1#(lok-S$Bus)),0,t13      ; Контроль входящих 1
        SAVEVALUE   KvantEN,X$ENBus1            ; Последний вошедший 1
        SAVEVALUE   ENBus,0                     ; Запрет на вход 1
t13     ADVANCE     tmp2                        ; Время на вход 1
;---------------------------------------- Вход-окончание
        SAVEVALUE   KvantEN-,1                  ; Последний вошедший 1
        TEST E      X$KvantEN,0,t200            ; Последний вошёл и можно ехать 1
        SAVEVALUE   PluVet,1                    ; Разрешение на движение 1
t200    TRANSFER    ,t2000                      ; Накопление на выход 1
; Остановка 2 ***********************************************************************
        GENERATE    tmp5,(0.5#tmp5)             ; Пассажир пришёл 2
        ASSIGN      2,2                         ; Отметка пассажира 2
        QUEUE       2                           ; Встал в очередь 2
        TEST E      (X$ENBus#X$HaltNum),2       ; Разрешение на вход 2
;---------------------------------------- Вход-начало
        ENTER       Bus,1                       ; Вход в автобус 2
        DEPART      2                           ; Освобождение очереди 2
        SAVEVALUE   ENBus2+,1                   ; Подсчёт числа вошедших 2
        TEST E      (Q2#(lok-S$Bus)),0,t23      ; Контроль входящих 2
        SAVEVALUE   KvantEN,X$ENBus2            ; Последний вошедший 2
        SAVEVALUE   ENBus,0                     ; Запрет на вход 2
t23     ADVANCE     tmp2                        ; Время на вход 2
;---------------------------------------- Вход-окончание

        SAVEVALUE   KvantEN-,1                  ; Последний вошедший 2
        TEST E      X$KvantEN,0,t300            ; Последний вошёл и можно ехать 2
        SAVEVALUE   PluVet,1                    ; Разрешение на движение 2
t300    TRANSFER    ,t2000                      ; Накопление на выход 2
; Остановка 3 ***********************************************************************
        GENERATE    tmp5,(0.5#tmp5)             ; Пассажир пришёл 3
        ASSIGN      2,3                         ; Отметка пассажира 3
```

```
        QUEUE       3                           ; Встал в очередь 3
        TEST E      (X$ENBus#X$HaltNum),3       ; Разрешение на вход 3
;---------------------------------------- Вход-начало
        ENTER       Bus,1                       ; Вход в автобус 3
        DEPART      3                           ; Освобождение очереди 3
        SAVEVALUE   ENBus3+,1                   ; Подсчёт числа вошедших 3
        TEST E      (Q3#(lok-S$Bus)),0,t33      ; Контроль входящих 3
        SAVEVALUE   KvantEN,X$ENBus3            ; Последний вошедший 3
        SAVEVALUE   ENBus,0                     ; Запрет на вход 3
t33     ADVANCE     tmp2                        ; Время на вход 3
;---------------------------------------- Вход-окончание
        SAVEVALUE   KvantEN-,1                  ; Последний вошедший 3
        TEST E      X$KvantEN,0,t400            ; Последний вошёл и можно ехать 3
        SAVEVALUE   PluVet,1                    ; Разрешение на движение 3
t400    TRANSFER    ,t2000                      ; Накопление на выход 3
; Остановка 4 ***************************************************************************
        GENERATE    tmp5,(0.5#tmp5)             ; Пассажир пришёл 4
        ASSIGN      2,4                         ; Отметка пассажира 4
        QUEUE       4                           ; Встал в очередь 4
        TEST E      (X$ENBus#X$HaltNum),4       ; Разрешение на вход 4
;---------------------------------------- Вход-начало
        ENTER       Bus,1                       ; Вход в автобус 4
        DEPART      4                           ; Освобождение очереди 4
        SAVEVALUE   ENBus4+,1                   ; Подсчёт числа вошедших 4
        TEST E      (Q4#(lok-S$Bus)),0,t43      ; Контроль входящих 4
        SAVEVALUE   KvantEN,X$ENBus4            ; Последний вошедший 4
        SAVEVALUE   ENBus,0                     ; Запрет на вход 4
t43     ADVANCE     tmp2                        ; Время на вход 4
;---------------------------------------- Вход-окончание
        SAVEVALUE   KvantEN-,1                  ; Последний вошедший 4
        TEST E      X$PasEnter,0,t500           ; Последний вошёл и можно ехать 4
        SAVEVALUE   PluVet,1                    ; Разрешение на движение 4
t500    TRANSFER    ,t2000                      ; Накопление на выход 4
; Остановка 5 ***************************************************************************
; Выход *********************************************************************
t2000   TEST E      1,1
        TEST E      X$ELBus,1                   ; Разрешение на выход 5
        LEAVE       Bus,1                       ; Выход 5
        SAVEVALUE   KvantEL-,1                  ; Подсчёт выходящих 5
        TEST E      X$KvantEL,0,t2200           ; Кто последний на выход? 5
        SAVEVALUE   ELBus,0                     ; Запрет на выход 5
t2200   ADVANCE     tmp1                        ; Время на выход 5
        SAVEVALUE   KvantEL2-,1                 ; Подсчёт выходящих 5
        TEST E      X$KvantEL2,0,t2400          ; Вышел ли последний? 5
        SAVEVALUE   ELBusFin,1                  ; Окончание выхода 5
t2400   TERMINATE   0                           ; Уход пассажира 5
;-------------------------------------------------------------------------------------
; Таймер (в минутах)
;-------------------------------------------------------------------------------------
        GENERATE    60
        TERMINATE   1
```

Пассажиры приходят на свои остановки случайным образом, ожидают автобус и в случае его прихода пытаются сесть в него. Это удаётся, если есть пустые места (вместимость автобуса ограничена).

При движении автобуса на каждой остановке случайное число пассажиров решает выйти из автобуса, освобождая тем самым места для новых пассажиров. Учитываются процессы посадки и высадки пассажиров.

Как обычно, процесс описывается в основном временами задержек на соответствующие операции и происходящими событиями.

3 Элементы языка GPSS World

3.1 Блоки и команды GPSS World

В данном разделе приводится описание основных блоков языка GPSS World, взятое из [9] и несколько переработанное.

Изменения состояния в моделях GPSS World происходят в результате входа сообщений в блоки и выполнения подпрограмм, связанных с этими блоками. В данном разделе описываются следующие типы блоков: **GENERATE**, **TERMINATE**, **ADVANCE**, **ASSIGN**, **INDEX**, **MARK**, **COUNT**, **SELECT**, **PRIORITY** и **BUFFER**.

Блок GENERATE

Блок **GENERATE** является источником потока сообщений в модели. В данном блоке производится подготовка сообщений и запуск их в модель через интервалы времени, заданные пользователем. Кроме задания правильной временной последовательности, пользователь может в блоке **GENERATE** задать некоторую информацию об атрибутах сообщений.

Блок **GENERATE** имеет следующий формат записи:

```
GENERATE [<A>],[<B>],[<C>],[<D>],[<E>]
```

В поле *A* указывается время, которое определяет интервал между моментами генерации сообщений блоком **GENERATE**. Операнд *A* может быть именем, положительным целым числом или непосредственно СЧА. Нельзя использовать в качестве операнда параметры транзакта.

В поле *B* задается модификатор, который изменяет значения интервала генерации сообщений по сравнению с интервалом, указанным в поле *A*. Операнд *B* может быть именем, положительным целым числом или непосредственно СЧА. Нельзя использовать в качестве операнда параметры транзакта.

Может быть два типа модификаторов:

– модификатор — интервал;

– модификатор — функция.

С помощью модификатора-интервала задается равномерный закон распределения времени между генерацией сообщений. При вычислении разницы значений, заданных в полях *A* и *B*, получается нижняя граница интервала, а при вычислении суммы — верхняя граница. После генерации очередного транзакта выбирается число из полученного интервала, и это будет значение времени, через которое следующее транзакт выйдет из блока **GENERATE**.

Например:

```
GENERATE   25, 10
```

В этом случае генерация сообщений производится по равномерному закону из интервала — (15,35).

Более сложные распределения могут быть представлены при использовании модификатора-функции, под действием которого вычисленное значение аргумента поля *A* умножается на значение функции, заданной в поле *B*. От значения функции целая часть не берется; отбрасывание дробной части производится только после умножения его на среднее значение.

Следует обратить внимание на то, что транзакта генерируются с заданными интервалами только в том случае, если у блоков, следующих за блоком **GENERATE** (например, **GATE**, **TEST**, **SEIZE**, **PREEMPT** или **ENTER**), не выставлены блокирующие условия. Каждое последующее транзакт формируется только тогда, когда транзакт из блока **GENERATE** входит в следующий блок. Формирование последующих сообщений включает вычисление интервала времени, в течение которого подготовленное транзакт остается в блоке **GENERATE**. Вычисленное значение при сложении со значением абсолютного условного времени дает значение времени, при котором транзакт войдет в модель.

Из-за возможных воздействий на модель при изменении заданного интервала генерации сообщений нежелательно, чтобы после блока **GENERATE** следовал блок, создающий блокирующее условие.

Если первый из вычисленных интервалов между моментами генерации сообщений равен 0, то этот интервал принимается равным 1. Если поля *A* и *B* пустые, что указывает на нулевой интервал между моментами генерации сообщений, то блок **GENERATE** будет генерировать транзакта до тех пор, пока не использует все транзакта, которые могут быть активными в какой-то определённый момент времени. Чтобы предупредить это, следует либо задать предел генерации (поле *D*), либо за блоком **GENERATE** должен следовать блок, который вызывает блокирующее условие.

В поле *C* задается начальная задержка. Начальная задержка относится к моменту формирования первого транзакта в блоке **GENERATE** как при первом просчете модели, так и после выполнения операции **CLEAR**. Начальная задержка — это момент времени, в который первое сгенерированное транзакт должно выйти из блока **GENERATE**; поля *A* и B на задержку транзакта влияния не имеют. Начальная задержка может быть меньше, равна или больше среднего времени, заданного в поле *A*. Операнд *C* может быть именем, положительным целым числом или непосредственно СЧА. Нельзя использовать в качестве операнда параметры транзакта.

В поле *D* задается предел генерации. Эта величина представляет собой максимальное число сообщений, которое будет создано в блоке **GENERATE**. Операнд *D* может быть именем, положительным целым числом или непосредственно СЧА. Нельзя использовать в качестве операнда параметры транзакта. Если поле *D* пусто, блок генерирует неограниченное число сообщений. Предел генерации инициализируется повторно операцией **CLEAR**.

Поле *E* определяет приоритет сообщений. Операнд *E* может быть именем, положительным целым числом или непосредственно СЧА. Нельзя использовать в качестве операнда параметры транзакта. Если поле *E* не задано, приоритет по умолчанию равен 0.

В начальный момент времени в каждом блоке **GENERATE** производится подготовка к выходу одного транзакта. На этой стадии модель еще полностью не инициализирована для выполнения. По этой причине все СЧА, описанные в блоке **GENERATE**, должны быть уже определёны. В модели блоку **GENERATE** должны предшествовать операторы описания **INITIAL, FUNCTION** и **VARIABLE**. Это делается для того, чтобы СЧА в блоке **GENERATE**, который ссылается на них, давали желаемые результаты.

Когда транзакт покидает блок **GENERATE**, счётчик общего числа прошедших через блок сообщений (**N***j*) увеличивается на единицу. Так как последующее транзакт не генерируется до тех пор, пока предыдущее транзакт не покидает блок **GENERATE**, то содержимое счётчика текущего числа находящихся в блоке сообщений (**W***j*) никогда не превышает 1.

При повторном описании блока **GENERATE** при помощи нового оператора описания блока интерпретатор GPSS просматривает все находящиеся в данный момент модели транзакта и проверяет, есть ли среди них транзакта, связанные с повторно описываемым блоком **GENERATE** (таких сообщений может и не быть, если данный блок уже создал заданное число сообщений). Эти транзакта, если они есть, уничтожаются. Операнды нового блока **GENERATE** заменяют операнды предыдущего блока **GENERATE** и затем создается новое транзакт, используя спецификации нового блока.

При использовании блока **GENERATE** необходимо помнить, что транзакт не должно входить в блок **GENERATE**. Если транзакт пытается это сделать, возникает ошибка выполнения.

Оператор **CLEAR** удаляет из модели GPSS все транзакта, в том числе и те, которые связаны с блоками **GENERATE**. После завершения действия оператора **CLEAR**, транзакта генерируются в каждом блоке **GENERATE**, как если бы команда **GENERATE** встретилась в первый раз. Всем СЧА придаются новые значения, и начальная задержка снова воздействует на первое генерируемое транзакт. Предел генерации вновь устанавливается равным значению, полученному при повторном задании значений СЧА.

Если в операторе описания блока **GENERATE** в поле *D* было задано максимальное число генерируемых блоком сообщений, и заданное число сообщений уже вышло из этого блока, т. е. блок уже закончил работу, то в процессе моделирования этот блок может быть снова запущен только в одном из двух случаев:

1) выполнены операции, заданные оператором **CLEAR**, и производится повторный запуск всех блоков **GENERATE**;

2) блок **GENERATE** описан повторно.

Блок TERMINATE

Блок **TERMINATE** имеет следующий формат записи:

```
TERMINATE [<A>]
```

Блок **TERMINATE** удаляет из модели входящие транзакта.

В поле *A* задается число единиц, на которое этот блок изменяет содержимое счётчика завершений, определяющего момент окончания моделирования. Операнд *A* может быть именем, положительным целым числом, СЧА или СЧА*<параметр>. По умолчанию значение, определяемое полем *A*, равно 0. Если поле *A* пусто, то транзакт уничтожается, а содержимое счётчика не изменяется.

Когда пользователь подготавливает модель, он задает время счета, указывая в операторе **START** значение счётчика завершений. Поскольку различные пути сообщений в модели имеют различные смысловые значения, каждый блок **TERMINATE** может либо уменьшать, либо не уменьшать содержимое счётчика завершений. Если содержимое счётчика уменьшилось до 0, счет завершается.

Рассмотрим пример, в котором показывается возможность управления временем моделирования.

```
GENERATE    1000
TERMINATE   1
START       5
.............
START       20
```

В этом примере отсутствует сама модель, а приведен только сегмент управления временем моделирования.

Каждое транзакт, входящее в блок **TERMINATE**, будет уменьшать содержимое счётчика на единицу. Предположим, что все остальные блоки TERMINATE в модели имеют пустые поля *A*, что означает, что содержимое счётчика не уменьшается. Тогда программа будет считать до тех пор, пока условное время не станет равным **5000**, поскольку в первой карте **START** начальное содержимое счётчика задано равным 5. Поскольку блок **GENERATE** генерирует по одному транзакту через каждые **1000** единиц условного времени, пятое транзакт войдет в блок **TERMINATE** в момент, когда условное время будет равно **5000**. Во второй команде **START** начальное значение счётчика задано равным **20**, поэтому второй прогон будет продолжаться в течение 20000 единиц условного времени.

Каждый раз, когда транзакт входит в блок TERMINATE, общее число сообщений, вошедших в блок **TERMINATE** (**N***j*), увеличивается на единицу. Число сообщений, находящихся в данный момент времени в блоке TERMINATE, всегда равно нулю, т. е. **W***j* = 0.

Стандартным числовым атрибутом, связанным с описываемым оператором, является **TG1** — число, равное текущему значению счётчика завершений. TG1 возвращает содержимое счётчика завершений, которое уменьшается блоком TERMINATE при заданном операнде *A*. Эта величина первоначально задается оператором **START** и указывает на завершение моделирования, когда становится равной 0.

Блок ADVANCE

Блок **ADVANCE** имеет следующий формат записи:

```
ADVANCE <A>,[<B>]
```

Блок **ADVANCE** задерживает продвижение транзакта на заданный период времени. В поле *A* задается среднее время пребывания транзакта в блоке **ADVANCE**. Содержимое поля *A* может быть именем, любым целым числом, в том числе и 0, СЧА или СЧА*****<параметр>. Если время задержки равно 0, транзакт помещается в список текущих событий перед транзактами с таким же приоритетом. Транзакты с положительным временем задержки помещаются в список будущих событий.

В поле *B* указывается способ модификации среднего значения, заданного в поле *A*. Операнд *B* может быть именем, положительным целым числом, СЧА или СЧА*****<параметр>.

Может быть два типа модификаторов:

– модификатор-интервал;

– модификатор-функция.

Интервал изменения среднего времени задержки может быть задан константой, значение которой не должно превосходить среднего времени задержки, вычисленного для данного транзакта. Эта константа определяет интервал, в котором времена задержки распределены равномерно. Все времена задержки выражаются целыми числами. Любое из (2B+1) целых чисел, заключенных в интервале (A-B, A+B), будет выбираться с вероятностью 1/(2B+1).

Например, если среднее время задано равным 10, а константа, определяющая интервал изменения, равна 5, то время задержки транзакта в блоке изменяется от 5 до 15. Для каждого из входящих в блок **ADVANCE** сообщений выбирается только одно из возможных значений времени задержки. Любое целое число из интервала (5—15), включая границы 5 и 15, будет выбираться с вероятностью 1/11. Дробные значения времени задержки не допускаются, поскольку условное время интерпретатора принимает только целые значения. Константа, определяющая интервал времени задержки, не должна превосходить среднего времени задержки, в противном случае может быть получено отрицательное время задержки в блоке **ADVANCE**. Отрицательное значение задержки всегда считается ошибкой.

Если в поле *B* записан модификатор-функция, то вычисленное значение атрибута, заданного в поле *A*, умножается на значение функции, заданной в поле *B*. Результат округляется до целого значения и используется как время задержки.

Операции блока ADVANCE

После того, как транзакт вошло в блок **ADVANCE** и было включено в список будущих событий, другое транзакт может войти в блок **PREEMPT** и генерировать прерывание на устройстве, занятом транзактм, находящимся в блоке **ADVANCE**. Для транзакта, находящегося в блоке **ADVANCE**, возможны два варианта:

1) генерированное другим транзактм прерывание является для данного транзакта первым;

2) данное транзакт уже было прервано на каком-либо из занятых им устройств.

В первом случае происходит следующее:

1) остаток времени, в течение которого транзакт должно было находиться в блоке **ADVANCE**, равен разнице вычисленного времени выхода транзакта из блока **ADVANCE** и текущего значения абсолютного условного времени;

2) транзакт удаляется из списка будущих событий;

3) транзакт рассматривается, как находящееся в состоянии прерывания;

4) указывается, что транзакт находится в списке прерывания;

5) счётчик прерываний устанавливается в единицу.

Находящееся в блоке **ADVANCE** транзакт, которое уже было прервано на одном из устройств и переведено в состояние прерывания, может быть прервано еще на 255 занятых им устройствах. Счётчик прерывания увеличивается на единицу при каждом новом прерывании.

Транзакт, занимающее блок **ADVANCE** и некоторое устройство, может находиться в списке текущих событий (например, когда блок **ADVANCE** имеет нулевую задержку). Обслуживание этого транзакта также может быть прервано другим транзактм. В этом случае удаление занимающего устройство транзакта из списка текущих событий и перевод его в состояние прерывания производится не сразу. Сначала устанавливается индикатор состояния прерывания. Транзакт, занимающее устройство, будет обрабатываться интерпретатором как обычно и перейдет в состояние прерывания только тогда, когда оно войдет в блок **ADVANCE** с ненулевой задержкой.

Когда прерываемое транзакт входит в блок **ADVANCE** выполняются следующие операции:

1) записывается отличное от нуля время задержки в блоке **ADVANCE.** Время пребывания транзакта в блоке **ADVANCE** будет отсчитываться с момента, когда значение счётчика прерываний этого транзакта станет равным нулю. В этот момент транзакт будет включено в список будущих событий;

2) индикатор состояния прерывания устанавливается в единицу;

3) транзакт удаляется из списка текущих событий;

4) транзакт помещается в список прерываний.

Когда вычисленное значение времени задержки транзакта в блоке ADVANCE отлично от нуля, значения счётчика текущего числа сообщений в блоке (**W***j*) и счётчик общего числа входов (**N***j*) увеличиваются на единицу. Если вычисленное время задержки равно нулю, и транзакт может перейти к следующему блоку, то увеличивается только счётчик **N***j*. Если же транзакт не может войти в следующий блок, то увеличивается и счётчик **W***j*.

Блок ASSIGN

Блок **ASSIGN** имеет следующий формат записи:

```
ASSIGN   <A>,<B>,[<C>]
```

Блок **ASSIGN** заменяет, увеличивает или уменьшает текущее значение параметра транзакта на заданное значение.

В поле *A* задается номер параметра, которому присваивается значение. Операнд *A* может быть именем, положительным целым числом, СЧА, СЧА*<параметр> и следующими за ними знаками **+, –**. Если значение параметра нужно увеличить или уменьшить, то справа в поле *A* ставится знак сложения или вычитания. Например, аргумент поля *A* может быть закодирован следующим образом:

1) **2** — заменяется текущее значение параметра 2;

2) **2–** — вычитается заданное значение из текущего значения параметра 2;

3) **2+** — прибавляется заданное значение к текущему значению параметра 2;

Поле *B* определяет значение, которое следует добавить или вычесть из значения параметра, заданного аргументом поля *A*, или заменить его. Если такой параметр не существует, то он создается. Операнд *B* может быть именем, любым целым числом, СЧА, или СЧА*****<параметр>.

Пример:

```
ASSIGN   2000+,-3
```

В этом примере значение поля *B*, равное **–3**, добавляется к значению параметра с номером **2000**, который задан операндом *A*. Если такой параметр в транзакте отсутствует, то он создается со значением, равным 0, до того, как будет произведено добавление. Тогда значение параметра транзакта становится равным –3.

Поле *C* задает номер модификатора-функции. При использовании поля *C* значение аргумента поля *B* умножается на значение модификатора-функции (от функции берется целая часть). Полученное произведение, от которого выделяется целая часть, становится значением, которое изменяет значение параметра, заданного в поле *A*. Следует отметить: если в поле *C* запи-

сано **FN1**, это не означает, что номер модификатора-функции равен 1, он равен целому значению **FN1**.

Рассмотрим модель, показывающую, как можно использовать блок ASSIGN для ввода информации о моделируемой системе в модель.

```
10  FUNCTION   RN16 C3
0.075, 1/0.55, 2/.999, 3
1   FUNCTION   ...
.
.
.
2   FUNCTION   ...
.
.
.
3   FUNCTION   ...
.
.
.
    GENERATE   200,12
    ASSIGN     8,FN10
    ASSIGN     2,FN*P8
```

Предполагается, что система связи получает информационные транзакта трех типов, причем каждый из них характеризуется своим распределением числа символов в транзакте. В модели в каждом транзакте параметр **8** будет указывать тип транзакта, а параметр **2** число символов в транзакте.

Блок **ASSIGN** может использоваться для управления логикой модели. Предположим, что система последовательно обрабатывает по одному символу поступившего информационного транзакта. Рассмотрим часть модели, в которой блок **ASSIGN** используется для построения цикла обработки транзакта.

```
.
MET1   ASSIGN   1+,1
       ADVANCE  X1
       TEST E   P1,P2,MET1
```

Каждое транзакт представляет собой одно информационное транзакт и должно повторять блоки, изображающие процесс обработки символа, в данном случае блок **ADVANCE**, столько раз, сколько символов в соответствующем информационном транзакте. Длина информационного транзакта записана в параметр **2**.

Для организации цикла с соответствующим числом повторений для каждого транзакта, используются блоки **ASSIGN** и **TEST**.

Блок **ASSIGN** добавляет единицу к счётчику числа выполненных циклов обработки транзакта (счётчик организован в параметре 1). Предполагается, что параметр 1 устанавливается в "0" перед входом в цикл обработки. Пока не выполнится условие **P1 = P2**, блок **TEST** будет возвращать транзакт к началу цикла. После выполнения условия из блока **TEST** транзакт перейдет к следующему по номеру блоку.

СЧА, связанным с описанным оператором, является **P**<параметр> или ***<параметр> — значение параметра. **P**<параметр> возвращает значение параметра, заданного <параметром>.

Блок INDEX

Блок **INDEX** имеет следующий формат записи:

```
INDEX   <A>,<B>
```

Блок **INDEX** к определённому в поле *A* значению параметра транзакта прибавляет величину, определённую полем *B*. Результат записывается в параметр 1. Поле *A* задает номер параметра, значение которого увеличивается на значение аргумента поля *B*. Если параметра с таким номером не существует, то возникает ошибка выполнения. Однако, если для обрабатываемого транзакта не существует параметра с номером 1, то он создается.

Операнд *A* может быть именем, положительным целым числом, СЧА или СЧА*<параметр>.

Поле *B* задает числовую величину, которая прибавляется к содержимому параметра. Операнд *B* может быть именем, любым целым числом, СЧА или СЧА*<параметр>.

Пример:

```
INDEX   2,1
```

В этом примере, когда транзакт входит в блок **INDEX**, его параметру с номером **1** присваивается сумма **1** и значения параметра **2** данного транзакта.

СЧА, связанным с описываемым оператором, является **P**<номер параметра> или ***<номер параметра> — значение параметра. Возвращает значение параметра с указанным номером.

Блок MARK

Блок **MARK** имеет следующий формат записи:

```
MARK [<A>]
```

Блок **MARK** либо заменяет значение отметки времени транзакта на текущее значение абсолютного условного времени (операнд *A* не определён), либо записывает значение условного времени в заданный параметр транзакта (при использовании операнда *A*).

Поле *A* содержит номер параметра, в который записывается значение абсолютного условного времени. Если такого параметра не существует, то он создается. Операнд *A* может быть именем, положительным целым числом, СЧА или СЧА*<параметр>.

Примеры:

```
MARK BEGINNING
```

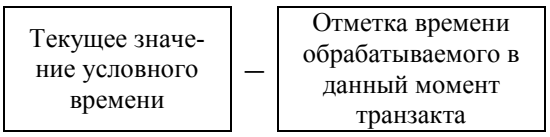
В этом примере, когда транзакт входит в блок **MARK**, его параметру с именем **BEGINNING** присваивается значение абсолютного условного времени **AC1**.

```
MARK
```

В этом примере значение отметки времени обрабатываемого в данный момент транзакта становится равным значению абсолютного условного времени.

Исходное значение времени создания транзакта может быть заменено на текущее значение абсолютного условного времени при прохождении транзакта через блок **MARK**. Каждое транзакт имеет следующие два стандартных числовых атрибута, связанных с временем прохождения участков модели данным транзактм:

1) **M1** — время прохождения транзактм модели. Вычисление значения этого СЧА производится следующим образом:



2) **MP***j* — промежуточное прохождение транзактм участка модели. Вычисление значения этого СЧА производится следующим образом:

Блок COUNT

Блок **COUNT** имеет следующий формат записи:

```
COUNT <X>  <A>,<B>,<C>,[<D>],[<E>]
```

<*X*> — здесь и далее обозначение используемого логического или условного операторов. Данный операнд может принимать следующие значения: **FNV**, **FV**, **I**, **LS**, **LR**, **NI**, **NU**, **SE**, **SF**, **CNE**, **SNF**, **SNV**, **SV**, **U**, **'E'**, **'G'**, **'GE'**, **'L'**, **'LE'**, **MIN**, **MAX** или **'NE'**.

Блок **COUNT** определяет число объектов, удовлетворяющих заданному условию. Например, пользователю может понадобиться узнать число свободных устройств в определённом диапазоне номеров устройств или число памятей с коэффициентом использования меньше 50 (в частях от 1000) и т. д.

Логический оператор задает логическое условие. Например, подсчет свободных устройств, подсчет выключенных логических ключей и т. д. Ниже приведен список логических операторов для различных типов объектов. Устройства имеют следующие логические условные операторы:

NU — устройство свободно (доступно);

U — устройство занято (в результате выполнения транзактм блока **SEIZE** или **PREEMPT**);

NI — устройство не прервано (т. е. либо оно свободно, либо занято транзактм, выполнившим блок **SEIZE**);

I — устройство прервано (устройство занято транзактм, выполнившим блок **PREEMPT**);

FV — устройство доступно;

FNV — устройство недоступно.

Многоканальные устройства имеют следующие логические условные операторы:

SE — многоканальное устройство пусто (нулевое содержимое);

SNE — многоканальное устройство не пусто (ненулевое содержимое);

SF — многоканальное устройство заполнено;

SNF — многоканальное устройство не заполнено;

SV — многоканальное устройство доступно;

SNV — многоканальное устройство недоступно.

Логические ключи имеют следующие логические операторы:

LR — проверка ключа на состояние "выключено";

LS — проверка ключа на состояние "включено";

При использовании логических операторов поля *D* и *E* могут быть пустыми.

В блоке **COUNT** можно использовать следующие условные операторы:

'L' — меньше. Условие выполняется, если значение стандартного числового атрибута, заданного в поле *E* , меньше значения стандартного числового атрибута, заданного в поле *D*;

'LE' — меньше или равно. Условие выполняется, если значение СЧА, заданного в поле *E*, меньше или равно значению СЧА, заданного в поле *D*;

'E' — равно. Условие выполняется, если значение СЧА, заданного в поле *E*, равно значению СЧА, заданного в поле *D*;

'NE' — не равно. Условие выполняется, если значение СЧА, заданного в поле *E*, не равно значению СЧА, заданного в поле *D*;

'G' — больше. Условие выполняется, если значение СЧА, заданного в поле *E*, больше значения СЧА, заданного в поле *D*;

'GE' — больше или равно. Условие выполняется, если значение СЧА, заданного в поле *E*, больше или равно значению СЧА, заданного в поле *D*;

MAX — наибольшее значение из всех значений СЧА объектов, удовлетворяющих заданному условию;

MIN — наименьшее значение из всех значений СЧА объектов, удовлетворяющих заданному условию.

Если используются условные операторы, то поля *D* и *E* блока COUNT должны быть заданы.

В поле *A* задается номер параметра вошедшего в блок транзакта, в котором будет организован счётчик числа объектов. Операнд *A* может быть именем, положительным целым числом, СЧА или СЧА*<параметр>.

В поле B определяется нижняя граница диапазона изменения номеров или имен объектов, для которых проверяется заданное условие. Операнд *B* может быть именем, положительным целым числом, СЧА или СЧА*<параметр>.

В поле *C* определяется верхняя граница диапазона изменения номеров или имен объектов, для которых проверяется заданное условие. Операнд *C* может быть именем, положительным целым числом, СЧА или СЧА**<параметр>. Значение поля *C* должно быть больше либо равно значению, заданному в поле *B*. Номер проверяемых объектов не должен превышать 32768.

Поле *D* задает сравниваемое значение (величина сравнения) для аргумента поля *E*. Операнд *D* может быть именем, любым целым числом, СЧА или СЧА**<параметр>. Это значение используется совместно с заданными условными операторами (**'E'**, **'NE'**, **'G'**, **'GE'**, **'L'**, **'LE'**). Значение СЧА сравнивается со значением объекта, заданного аргументом поля *E*. Если условный оператор не используется, поле *D* можно не задавать.

Поле *E* используется совместно с аргументом поля *D* и условным оператором. В поле *E* задается какой-либо из стандартных числовых атрибутов просматриваемых объектов. Необходимо только записывать мнемоническое обозначение атрибута, поскольку диапазон изменения номеров объектов задан полями *B* и *C*.

Рассмотрим несколько примеров использования блока **COUNT**.

```
COUNT 'LE'  1,1,5,X10,FC
```

В этом примере подсчитывается число устройств (из устройств **1-5** включительно), у которых счётчик числа входов (**FC**) меньше или равен текущему значению ячейки **10**. Результат подсчета будет записан в параметре 1 вошедшего в блок **COUNT** транзакта.

В нижеприведенном примере подсчитывается число заполненных многоканальных устройств (**SF**) при изменении номеров в интервале **10-20** (включительно). Результат подсчета записывается в параметре **5** вошедшего в блок транзакта.

```
COUNT SF  5,10,20
```

Блок SELECT

Блок **SELECT** имеет следующий формат записи:

```
SELECT <X>  <A>,<B>,<C>,[<D>],[<E>],[<F>]
```

Блок **SELECT** выбирает первый объект в заданном диапазоне, который удовлетворяет определённому условию. Номер этого объекта записывается в

заданный параметр вошедшего в блок транзакта. *B* основном, действия блока **SELECT** аналогичны действиям блока **COUNT**.

<*X*> — обозначение логического или условного операторов.

Логические и условные операторы имеют такой же смысл, как и в блоке COUNT.

MIN и **MAX** задают работу блока соответственно в минимальном и максимальном режимах.

В поле *A* задается номер параметра вошедшего в блок **COUNT** транзакта, куда записывается номер объекта, удовлетворяющего заданному условию.

Операнд *A* может быть именем, положительным целым числом, СЧА или СЧА*****<параметр>.

Поле *B* определяет нижнюю границу диапазона изменения номеров объектов, для которых проверяется заданное условие. Операнд *B* может быть именем, положительным целым числом, СЧА или СЧА*****<параметр>.

Поле *C* определяет верхнюю границу диапазона изменения номеров объектов, для которых проверяется заданное условие. Операнд *C* может быть именем, положительным целым числом, СЧА или СЧА*****<параметр>.

В поле *D* задается величина сравнения. Это значение используется совместно с условными операторами (**'E'**, **'NE'**, **'GE'**, **'L'**, **'LE'**). Операнд *D* может быть именем, любым целым числом, СЧА или СЧА*****<параметр>.

Значение стандартного числового атрибута сравнивается со значением атрибута объекта, заданного в поле *E*. Если не используется условный оператор, поле *D* можно не задавать.

Поле *E* используется совместно с полем *D* и условным оператором. В поле *E* задается только мнемоническое обозначение СЧА объекта без указания конкретного номера, поскольку диапазон изменения номеров объектов задан полями *B* и *C*. В поле *E* может быть задан любой из стандартных числовых атрибутов объекта.

Поле *F* задает номер альтернативного блока для входящего транзакта, если заданным условиям не удовлетворяет ни один объект в диапазоне. Операнд *F* может быть именем, положительным целым числом, СЧА или СЧА*****<параметр>. Если операнд *F* не используется, то транзакт будет продвинуто к следующему по номеру блоку. Рассмотрим примеры использования блока **SELECT**.

```
SELECT MAX  1,5,10,,FR
```

В данном примере из устройств с номерами **5-10** (включительно) выбирается устройство с максимальным коэффициентом использования (**FR**). Номер выбранного устройства записывается в параметре 1 вошедшего транзакта.

В следующем примере из очередей с номерами **10-20** (включительно) выбирается первая очередь, длина которой (**Q**) больше или равна значению

ячейки 5 (**X5**). Номер очереди, удовлетворяющей этому условию, записывается в параметре **10** вошедшего в блок транзакта.

```
SELECT 'GE'  10,10,20,X5,Q
```

Блок PRIORITY

Блок **PRIORITY** имеет следующий формат записи:

```
PRIORITY <A>,[<B>]
```

Блок **PRIORITY** присваивает транзакту нужный приоритет. Приоритет сообщений влияет на порядок обработки сообщений процедурой просмотра и порядок занятия транзактами объектов оборудования.

Поле *A* задает новое значение приоритета. Операнд *A* может быть именем, любым целым числом, СЧА или СЧА*****<параметр>.

Новое значение приоритета может быть меньше, больше или равно текущему значению приоритета.

Поле *B* определяет режим **BUFFER**, в нем указывается значение "**BU**".

В общем случае процедура просмотра списка текущих сообщений пытается продвинуть обрабатываемое транзакт в следующий блок. Однако, если в поле *B* блока **PRIORUTY** записано слово **BU**, то после присвоения транзакту нового значения приоритета блок **PRIORITY** становится эквивалентным блоку **BUFFER.** Транзакт помещается в список текущих событий в конец своего нового приоритетного класса, процедура просмотра возвращается к началу списка текущих событий и начинает просмотр снова. Поскольку блок **BUFFER** не задерживает транзакта, вошедшие в этот блок, транзакт будет обработано процедурой просмотра снова при том же значении условного времени. Режим **BUFFER** подробно рассматривается при описании блока **BUFFER.**

Например:

```
PRIORITY  10
```

В этом примере вошедшему в блок транзакту присваивается приоритет, равный **10.**

Внутренние операции блока PRIORITY

Блок **PRIORITY** назначает новый уровень приоритета и затем перемещает транзакт в списке текущих событий так, чтобы оно стало последним среди сообщений с соответствующим значением приоритета. Эта операция выполняется и в том случае, когда новое значение приоритета равно старому. Бывают случаи, когда нужно изменить положение транзакта в том же классе приоритетов, в котором оно находится. Например, это может быть сделано перед входом транзакта в блок **SPLIT**. Поскольку транзакт окажется последним в своем классе приоритетов, копии (вновь созданные транзакта) сообщений, созданные блоком **SPLIT**, будут расположены в списке текущих собы-

тий непосредственно за породившим их транзактм, поскольку каждая копия помещается в список текущих событий в конец соответствующего класса.

При входе транзакта в блок **PRIORITY** автоматически устанавливается флаг изменения состояния. Следовательно, после того, как интерпретатор закончит продвижение транзакта, вошедшего в блок **PRIORITY**, он автоматически вернется к началу просмотра списка текущих событий. Возврат к началу списка необходим, поскольку в случае, если блок **PRIORITY** уменьшил приоритет вошедшего в него транзакта, интерпретатор пропустит все транзакта, находящиеся в списке текущих событий, начиная с того места, где рассматриваемое транзакт находилось до входа в блок **PRIORITY**, кончая его новым местом в списке. Установка флага изменения состояния и возврат к началу списка обеспечивает просмотр всех пропущенных сообщений в тот же момент условного времени.

В режиме **BUFFER** интерпретатор немедленно возвращается к началу списка текущих событий. Флаг изменения состояния также сразу же устанавливается в "0".

СЧА, связанным с описываемым оператором, является **PR** — значение приоритета обрабатываемого транзакта.

Блок BUFFER

Блок **BUFFER** имеет следующий формат записи:

```
BUFFER
```

Блок **BUFFER** используется для немедленного прекращения обработки транзакта и возврата интерпретатора к началу списка текущих событий. Обрабатываемое транзакт помещается в список текущих событий после сообщений с тем же классом приоритетов.

Интерпретатор пытается продвигать активное транзакт так далеко, на сколько это возможно в текущей модели.

В обычном режиме интерпретатор удаляет активное транзакт из списка текущих событий, вызывает процедуру обработки следующего блока и, если не произойдет ни одно из событий, прекращающих продвижение транзакта, помещает обрабатываемое транзакт в список текущих событий перед транзактами с тем же классом приоритетов. Если на пути продвижения транзакта произойдет одно из следующих событий:

1) транзакт входит в блок **ADVANCE** с положительной задержкой и переводится в список будущих событий;

2) транзакт не может войти ни в один из указанных последующих блоков;

3) транзакт входит в блоки **MATCH**, **ASSEMBLE** или **GATHER**, которые производят следующие действия:

– либо уничтожают транзакта (последние n-1 транзакта в блоке `ASSEMBLE`);

– либо переводят транзакт в состояние синхронизации (блоки **MATCH** и **GATHER** и первое транзакт в блоке **ASSEMBLE**);

4) транзакт уничтожается блоком **TERMINATE**;

5) транзакт переводится в список пользователя в блоке **LINK**; обычно интерпретатор переходит к обработке другого транзакта.

Обработка транзакта, вошедшего в блок **BUFFER**, является исключением из этого правила. После входа транзакта в блок **BUFFER** интерпретатор помещает транзакт в конец соответствующего класса приоритетов и возвращается к первому транзакту списка текущих событий (т. е. транзакту, имеющему наивысший приоритет). Поскольку блок **BUFFER** не задерживает входящие в него транзакта, эти транзакта обрабатываются при том же значении условного времени.

Если входящее в блок **BUFFER** транзакт стоит первым в списке текущих событий, то блок **BUFFER** не выполняет никаких полезных операций и является лишним. Отметим также, что ни одно из сообщений, стоящих в списке текущих событий за транзактм, вошедшим в блок **BUFFER**, не будет обработано до тех пор, пока интерпретатор не вернется к обработке транзакта, вошедшего в блок **BUFFER** и не закончит его обработку.

Блок **PRIORITY**, в поле *B* которого записано слово **BUFFER**, эквивалентен блоку **BUFFER** после того, как транзакт помещено в конец соответствующего класса приоритетов, заданного аргументом поля *A*. В результате интерпретатор возвращается к началу списка текущих событий. Поскольку блок **PRIORITY**, так же как и блок **BUFFER**, не задерживает входящие в него транзакта, транзакт с измененным значением приоритета будет обработано в тот же момент условного времени.

В результате действия приведенных ниже блоков **PRIORITY**, входящее в них транзакт будет обработано последним, т. е. после всех остальных сообщений, находящихся в списке текущих событий.

```
     ASSIGN     1,PR,PH
K    PRIORITY   0,BUFFER
K+1  PRIORITY   PH1
```

Блок **PRIORITY** с номером **K** помещает транзакт в конец списка текущих событий. Текущее значение приоритета транзакта записано в параметре **1** формата "полуслово" с тем, чтобы его можно было восстановить, когда транзакт войдет в блок **K+1**. В режиме **BUFFER** интерпретатор возвращается к началу списка текущих событий. Он просматривает все транзакта в списке текущих событий и опять встречает транзакт, вошедшее в блок **PRIORITY** с номером к. Это транзакт входит в блок **PRIORITY** с номером **K+1**, который восстанавливает первоначальный уровень приоритета транзакта, записанный в параметре **1** формата "полуслово".

Блоки, изменяющие порядок прохождения блоков транзактами

Обычно интерпретатор пытается продвинуть транзакт к следующему по номеру блоку. Однако существуют четыре типа блоков, которые позволяют изменять номер следующего блока. Эти блоки **TRANSFER**, **LOOP**, **TEST** и **GATE**.

Блок TRANSFER

Блок **TRANSFER** имеет следующий формат:

```
TRANSFER  [<A>],[<B>],[<C>],[<D>]
```

Блок **TRANSFER** является основным средством, позволяющим направить транзакт к любому блоку модели.

Поле *A* задает режим выбора следующего блока, к которому должно перейти транзакт.

Существуют следующие режимы работы блока **TRANSFER**:

безусловный (пробел);

статистический (**.**);

BOTH;

ALL;

PICK;

функция (**FN**);

параметр (**P**);

подпрограмма (**SBR**);

SIM.

Кроме того, операнд *A* может быть дробным числом, именем, положительным целым числом, СЧА или СЧА*****<параметр>.

Поля *B* и *C* задают возможные значения номеров следующих блоков или их положение. Использование значений описано при рассмотрении определённых режимов выбора. Операнды могут быть именем, положительным целым числом, СЧА или СЧА*****<параметр>. Если поле *B* пусто, ассемблер записывает в нем номер блока, следующего за блоком **TRANSFER**.

Безусловный режим выбора

Если операнд *A* пропущен, то блок **TRANSFER** работает в безусловном режиме. Входящее в блок **TRANSFER** транзакт переходит к блоку, указанному в поле *B*. Если транзакт в этот блок войти не может, попытка направить транзакт к какому-либо другому блоку не производится.

Например:

```
XFER  TRANSFER   ,NEXT
NEXT  SEIZE      1
```

Транзакта, входящие в блок **TRANSFER XFER**, переходят к блоку **NEXT**.

```
TRANSFER      ,V$TER
```

Транзакта, которые входят в вышеприведенный блок **TRANSFER**, сразу переходят в блок, номер которого определяется переменной **FER**.

Статистический режим выбора

Когда операнд *A* не является зарезервированным словом, блок **TRANSFER** работает в статистическом режиме выбора.

Значение аргумента, записанного после точки (**.**) в поле *A*, рассматривается как трехзначное число, показывающее (в частях от тысячи), какой процент входящих в блок сообщений следует направить к блоку, указанному в поле *C*. Остальные транзакта направляются к блоку, указанному в поле *B*, или к следующему по номеру блоку, если операнд *B* пропущен. Для каждого транзакта выбирается один из двух возможных вариантов; после того как выбор сделан, второй вариант для этого транзакта не рассматривается.

Числовое значение может быть задано при помощи любого стандартного числового атрибута. Если вычисленное значение аргумента меньше или равно нулю, будет происходить безусловная передача сообщений к блоку, указанному в поле *B*. Если же значение аргумента больше или равно 1000, то будет происходить безусловная передача сообщений к блоку, указанному в поле *C*.

Например,

```
BCD  TRANSFER  .709,BLK1,BLK2
```

Из общего числа сообщений, входящих в блок **BCD**, в среднем **.709** будут пытаться войти в блок **BLK2**. Остальные .209 будут пытаться войти в блок **BLK1**.

```
BCD  TRANSFER  .P1,BLK3,BLK4
```

Трехзначное число, записанное в параметре **1** сообщений, входящих в блок **BCD**, интерпретируется как вероятность (в частях от тысячи) того, что транзакт попытается войти в блок **BLK4**. В остальных случаях транзакт попытается войти в блок **BLK3**.

```
CDE  TRANSFER  .X1,BLK5,BLK6
```

Если в момент входа сообщений в блок **CDE** в ячейке **SAVEVALUE 1** записано число 30, то в среднем 3 % от общего числа сообщений будет направлено к блоку **BLK6**, а остальные 97 % попытаются войти в блок **BLK5**.

Режим BOTH

Если в поле *A* стоит зарезервированное слово **BOTH**, блок **TRANSFER** работает в режиме **BOTH**.

В этом режиме каждое входящее транзакт сначала пытается перейти к блоку, указанному в поле *B*. Если это сделать не удается, транзакт пытается перейти к блоку, указанному в поле *C*. Если транзакт не сможет перейти ни к тому, ни к другому блоку, оно остается в блоке **TRANSFER** и будет повторять в том порядке попытки перехода при каждом просмотре списка текущих со-

бытий, до тех пор, пока не сможет выйти из блока **TRANSFER**. Ниже приведен фрагмент программы, в котором транзакт сначала пытается перейти к блоку **TRY1**. Если оно не может войти в этот блок, оно пытается войти в блок **TRY2**. Если транзакт не может войти и в этот блок, оно остается в списке текущих событий и повторяет эти попытки при каждом просмотре списка до тех пор, пока не выйдет из блока **TRANSFER**.

```
      TRANSFER BOTH,TRY1,TRY2
TRY1  SEIZE    1
TRY2  SEIZE    2
```

Режим ALL

Если в поле *A* стоит зарезервированное слово **ALL**, блок **TRANSFER** работает в режиме **ALL**.

В этом режиме каждое входящее транзакт прежде всего пытается перейти к блоку, указанному в поле *B*. Если транзакт в этот блок войти не может, то последовательно проверяются все блоки в определённом ряду в поисках первого, способного принять это транзакт, включая блок, указанный операндом *C*. Номер каждого проверяемого блока вычисляется как сумма номера предыдущего блока и шага, заданного операндом *D*:

N + M, N + 2M, N + 3M, ... L,

где:

N — номер блока, указанного в поле *B*;

M — значение шага, заданного в поле *D*;

L — номер блока, указанного в поле *C*.

Этот номер должен быть больше номера блока, указанного в поле *B*, на величину, кратную шагу **M**. Если операнд *D* не задан, то проверяется каждый блок, номер которого принадлежит этому ряду, включая блок, определённый операндом *C*. Блоки, номера которых выше номера блока, указанного в поле *C*, не проверяются. Как только первый блок, способный принять транзакт, будет найден, транзакт входит в этот блок и оттуда продолжает свое дальнейшее движение. Если транзакт не может перейти ни к одному из указанных блоков, оно остается в блоке **TRANSFER** и повторяет описанную выше процедуру при каждом просмотре списка текущих событий до тех пор, пока не выйдет из блока.

Поскольку обычно в полях *B* и *C* записываются символические метки блоков, блоки следует располагать таким образом, чтобы при присвоении номеров разность между номерами блоков, указанных в полях *B* и *C*, была кратна шагу, указанному в поле *D*. Например,

```
TRANSFER  ALL,60,120,10
```

В этом примере транзакт будет последовательно пытаться перейти к блокам **60, 70, 80, ... 120.**

```
TRANSFER  ALL,NEXT1,NEXT2,5
```

Здесь режим **ALL** допустим только в том случае, если разность между номерами, присвоенными блокам **NEXT1** и **NEXT2**, кратна **5**.

```
TRANSFER  ALL, 60, 120, 25
```

В данном примере режим **ALL** недопустим, потому что разность между номерами блоков, записанных в полях *B* и *C*, не является кратной шагу, указанному в поле *D*.

Условными являются только режимы **BOTH** и **ALL**. Во всех остальных режимах выбор следующего блока производится в момент входа транзакта в блок. В режимах **BOTH** и **ALL** выбор следующего блока производится в момент снятия блокирующего условия. Следует отметить, что каждый раз, когда интерпретатор при просмотре списка текущих событий обнаруживает транзакт, задержанное в блоках **TRANSFER BOTH** или **TRANSFER ALL**, он пытается продвинуть транзакт, начиная с блока, указанного в поле *B*. Следовательно, в режиме **BOTH** в тех случаях, когда возможен переход к обоим блокам (*B* и *C*), блок *B* имеет некоторое преимущество. Аналогично, в режиме **ALL** в случае, когда возможен переход к нескольким блокам, блоки с меньшими номерами имеют некоторое преимущество перед блоками с большими номерами.

Режим PICK

Если в поле *A* стоит зарезервированное слово **PICK**, блок **TRANSFER** работает в режиме **PICK**. В этом режиме из последовательности блоков с номерами N, N+1, N+2,...M (N — номер блока, указанного в поле *B*, а M — номер блока, указанного в поле *C*) случайным образом выбирается один блок, к которому должно быть направлено транзакт. Все блоки, включая указанные в полях *B* и *C*, выбираются с одинаковой вероятностью, равной 1/(M-N)+1. Транзакт пытается перейти только к выбранному для него блоку. Если транзакт не может сразу перейти к следующему блоку, то оно будет ждать в блоке **TRANSFER** до тех пор, пока не будет снято блокирующее условие. Номер блока в поле *C* должен быть больше или равен N+1. Например:

```
TRANSFER  PICK,30,39
```

Транзакт, вошедшее в блок `TRANSFER`, пытается войти в один из 10 блоков (`30,31,...39`) с равной вероятностью : 1/10.

Режим "функция" (FN)

Если в поле *A* стоит зарезервированное слово **FN**, блок **TRANSFER** работает в режиме "функция".

Вычисляется значение функции, номер которой задан в поле *B* блока **TRANSFER**; если результат нецелый, от него берется целая часть. Для определёния номера следующего блока полученное целое число складывается с аргументом поля *C* (в поле *C* может быть записан ноль). Транзакт пытается перейти только к блоку с вычисленным номером. Транзакт остается в блоке

TRANSFER до тех пор, пока не сможет перейти именно к этому блоку. Например:

```
TRANSFER  FN,3,PH3
```

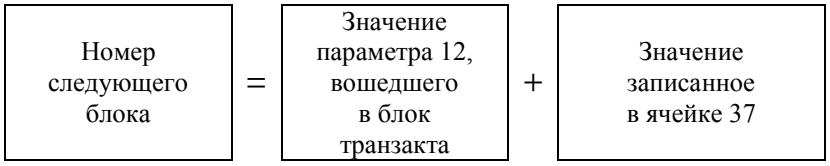
Режим "параметр"

Если в поле *A* стоит зарезервированное слово **P**, блок **TRANSFER** работает в режиме "параметр".

Значение аргумента поля *B* интерпретируется как номер *j* параметра входящего транзакта. Для определёния следующего номера блока для данного транзакта , значение этого параметра складывается со значением аргумента поля *C*. Если операнд *C* не задан , номер следующего блока будет равен значению параметра.

Например:

```
TRANSFER  P,12,37
```



Режим "подпрограмма" (SBR)

Если в поле *A* стоит зарезервированное слово **SBR**, блок **TRANSFER** работает в режиме "подпрограмма".

Вошедшее в блок **TRANSFER** транзакт будет пытаться перейти к блоку, указанному в поле *B*. Значение аргумента поля *C* интерпретируется как номер параметра; в этом параметре записывается номер *j* данного блока **TRANSFER**. Если такого параметра нет, то он создается. Этот режим блока **TRANSFER** обычно используется для перехода к подпрограмме, началом которой является блок, указанный в поле *B*. Например,

```
TRANSFER  SBR,NEXT,10
```

Если в конце подпрограммы записать блок

```
TRANSFER  P,10,1   ,
```

то транзакт сможет вернуться к блоку, следующему за блоком **TRANSFER SBR**, где следующий блок равен текущему значению, записанному в параметре под номером **10** (в данном случае это номер блока **TRANSFER SBR**) плюс **1**.

Режим SIM

Если в поле *A* стоит зарезервированное слово **SIM**, то блок **TRANSFER** работает в режиме **SIM.**

Режим введен для случая, когда требуется одновременное выполнение нескольких условий. Каждое транзакт имеет свой индикатор задержки (назовем его индикатором **SIM**). В этом индикаторе записывается результат любой попытки транзакта войти в следующий блок. Если интерпретатор обнаруживает условия, препятствующие входу транзакта в блок, то индикатор **SIM** этого транзакта устанавливается в единицу.

Если все условия перехода к следующему блоку удовлетворяются, то индикатор **SIM** остается равным нулю. Если не выполняется хотя бы одно из условий, то индикатор **SIM** данного транзакта устанавливается в единицу (в режимах **BOTH** и **ALL** индикатор **SIM** устанавливается в единицу только в том случае, когда переход невозможен ни к одному из указанных блоков). При входе транзакта в блок **TRANSFER** проверяется значение индикатора SIM. Если он равен нулю, транзакт направляется к следующему блоку, указанному в поле *B*. Если индикатор **SIM** равен единице, транзакт направляется к блоку, указанному в поле *C*, а индикатор **SIM** устанавливается в "0". В любом случае транзакт будет пытаться перейти только к выбранному для него блоку и будет находиться в блоке **TRANSFER** до выполнения соответствующих условий. В момент, когда создаются условия для выхода транзакта из блока **TRANSFER**, значение индикатора **SIM** не проверяется. Состояние индикатора **SIM** отмечается символом X в колонке DELAY распечатки информации о транзактах.

При задержке сообщений в блоках **ASSEMBLE, GATHER** или **MATCH** индикатор **SIM** в единицу не устанавливается.

Изменение индикатора SIM в блоке ADVANCE

Каждый раз, когда транзакт выходит из блока **ADVANCE** с нулевым временем задержки, индикатор **SIM** становится равным нулю. После того, как транзакт покинуло блок **ADVANCE**, оно может быть снова задержано по каким-либо причинам прежде, чем дойдет до тех блоков, в которых проверяется одновременность выполнения ряда условий. Следовательно, индикатор SIM может быть установлен в единицу до того, как начнется проверка условий. В таком случае перед блоками, в которых проверяется одновременное выполнение условий, следует поместить блок **TRANSFER SIM**, в котором в полях *B* и *C* указан один и тот же блок — первый из блоков, проверяющих условия. После прохождения через этот блок, индикатор **SIM** транзакта снова станет равным нулю. Например,

```
TRANSFER  SIM,10,10
```

Этот блок позволяет установить в "0" индикаторы всех сообщений, входящих в блок **10**.

Внутренние операции блока TRANSFER

При входе транзакта в блок **TRANSFER** (за исключением блоков, работающих в режимах **BOTH** и **ALL**) вычисляется номер следующего блока, к которому транзакт должно перейти, и транзакт пытается перейти к этому блоку. Вычисленный номер блока должен быть допустимым номером блока в текущей модели. Если транзакт не может перейти в этот блок, то номер блока запоминается. Номер вычисляется только один раз. Интерпретатор все время пытается продвинуть транзакт только к этому блоку.

Если блок **TRANSFER** работает в режиме **BOTH** или **ALL**, запоминается номер последнего блока, указанного в поле *C*. Этот номер также вычисляется только один раз. Если транзакт не сможет перейти ни к одному из указанных в режимах **BOTH** или **ALL** блоков, интерпретатор будет повторять попытки для всех указанных блоков при каждом просмотре списка текущих событий. Эти попытки будут повторяться даже в том случае, если блокирующие условия не будут сняты. По этой причине, использование блока **TRANSFER** в режимах **BOTH** или **ALL** может увеличить время обработки. В этом случае транзакт можно поместить в список пользователя на время, пока не будет найден блок, способный принять транзакт. Это может быть сделано с помощью блоков **LINK** и **UNLINK**.

Блок LOOP

Блок **LOOP** имеет следующий формат:

```
LOOP  <A>,[<B>]
```

Блок **LOOP** используется для организации циклов, т. е. для управления числом повторений определённой последовательности блоков в модели. Транзакт никогда не задерживается на входе блока **LOOP**.

В поле *A* задается параметр, который используется в качестве счётчика цикла. Операнд А может быть именем, положительным целым числом, СЧА или СЧА*****<параметр>.

Интерпретатор определяет значение параметра, заданного полем *A*. Если оказывается, что такой параметр не существует, то возникает ошибка выполнения и моделирование прекращается, иначе значение параметра уменьшается на единицу и записывается в том же параметре. Далее производится выбор следующего блока, к которому должно перейти транзакт из блока **LOOP**. Если новое значение параметра не равно нулю, то транзакт перейдет в блок, номер которого указан в поле *B*. Если же значение параметра — "0", то транзакт переходит к следующему по номеру блоку.

В поле *B* обычно указывается номер блока, являющегося началом цикла. Операнд *B* может быть именем, положительным целым, СЧА или

СЧА*****<параметр>. Один и тот же цикл может одновременно выполняться произвольным числом сообщений. Если транзакт входит в блок **LOOP** со значением параметра, равным n, то это транзакт войдет в блок **LOOP** n раз и вернется к началу этого цикла (n-1) раз. Если в момент первого входа транзакта в блок **LOOP** значение параметра, номер которого указан в поле *A*, нулевое или отрицательное, происходит ошибка выполнения.

Рассмотрим пример использования блока **LOOP**:

```
       ASSIGN  4,10
RPT    LOGIC S P4
       LOOP    4,RPT
```

Здесь с помощью блока **LOOP** организован цикл прохождения транзактов через блок с именем **RPT.**

СЧА, связанным с описываемым оператором, является **P**<параметр> — значение параметра. Возвращает значение параметра, номер которого задан <параметром>.

Блок TEST

Блок TEST имеет следующий формат:

```
TEST <X>  <A>,<B>,[<C>]
```

Блок **TEST** в противоположность блоку **LOOP**, не изменяет никаких атрибутов транзакта; он определяет номер следующего блока для вошедшего в него транзакта в зависимости от того, выполняется требуемое условие или нет. Блок управляет потоком сообщений, проверяя выполнение алгебраических отношений между значениями СЧА, заданных в полях *A* и *B*.

Операнды *A* и *B* — сравниваемые величины, которые могут быть именем, любым целым числом, СЧА или СЧА*****<параметр>.

Во вспомогательном поле операции оператора описания блока **TEST** — <*X*> — записывается один из шести условных операторов:

'L' — меньше. Отношение истинное, если значение аргумента поля *A* меньше значения аргумента поля *B*;

'LE' — меньше или равно. Отношение истинное, если значение аргумента поля *A* меньше или равно значению аргумента поля *B*;

'E' — равно. Отношение истинное, если значения обоих аргументов равны;

'NE' — не равно. Отношение истинное, если значения аргументов полей *A* и *B* не равны;

'G' — больше. Отношение истинное, если значение аргумента поля *A* больше значения аргумента поля *B*;

'GE' — больше или равно. Отношение истинное, если значение аргумента поля *A* больше или равно значению аргумента поля *B*.

Если отношение СЧА, заданных в полях *A* и *B*, истинно, транзакт переходит к следующему блоку. Если отношение ложно, транзакт переходит к блоку, номер которого задан полем *C*.

C — номер блока для входящего транзакта, если отношение величин, заданных в полях *A* и *B*, ложно. Операнд *C* может быть именем, положительным целым числом, СЧА или СЧА*****<параметр>.

Блок **TEST** может работать в двух режимах:

1) в режиме безусловного входа. Если в поле *C* задан номер следующего блока, транзакта никогда не задерживаются на входе блока **TEST**. Если заданное в блоке **TEST** отношение истинно, то транзакт пытается перейти к следующему по номеру блоку. Если отношение ложно, транзакт пытается перейти к блоку заданному полем *C*. Выбор следующего блока производится только один раз (в момент входа транзакта в блок **TEST**);

2) в режиме условного входа. Если поле *C* блока TEST пусто (т. е. не указан альтернативный выход), транзакта не могут войти в блок **TEST** до тех пор, пока условия не изменятся таким образом, что отношение будет истинно. Если отношение истинно, транзакт входит в блок **TEST** и пытается перейти к следующему по номеру блоку.

Отношения в блоках **TEST** проверяются интерпретатором при каждом просмотре транзакта, задержанного на входе блока **TEST**, работающего в режиме условного входа. Задержанные транзакта обычно помещаются в списки задержки и не обрабатываются интерпретатором до тех пор, пока не изменится блокирующее условие. Транзакта, задержанные блоками **TEST** в режиме условного входа в списки задержки не помещаются, следовательно использование таких блоков **TEST** может значительно увеличить время счета модели. Рассмотрим несколько примеров блоков **TEST**.

```
TEST 'L' C1,500,SNA
```

Пока значение относительного условного времени не достигнет **500**, транзакта от блока **TEST** будут переходить к следующему по номеру блоку. Как только значение условного времени станет равным **500** (и более), транзакта будут переходить к блоку, номер которого определяется полем *C*.

```
TEST 'GE'  N$PATH1,N$PATH2
```

Когда счётчик числа входов в блок **PATH1** (**N$PATH1**) больше или равен счётчику числа входов в блок **PATH2** (**N$PATH2**), транзакта входят в блок **TEST** и переходят к следующему по номеру блоку. Когда счётчик блока **PATH1** меньше счётчика блока **PATH2**, транзакта не могут войти в блок **TEST**.

```
TEST 'E'  V6,O,SNA
```

Когда вычисленное значение арифметической переменной **6** равно нулю, транзакта будут переходить к следующему по номеру блоку. Когда это значение не равно нулю, транзакта переходят к блоку, номер которого задан в поле *C*.

```
TEST 'G'  P7,R20
```

Транзакт может войти в блок **TEST** только в том случае, если значение параметра **7** больше свободного объема памяти 20 (**R20**). В противном случае транзакт не может войти в блок **TEST**.

Блок GATE

Блок **GATE** имеет следующий формат:

```
GATE  <X>  <A>,[<B>]
```

Блок GATE управляет потоком сообщений на основе значений логических операторов. Блок **GATE**, как и блок **TEST**, не изменяет никаких атрибутов сообщений, он определяет номер следующего блока, к которому должно перейти транзакт из блока **GATE**. Блок **GATE** может задержать транзакт на входе, если не задан альтернативный выход. Во вспомогательном поле операции <*X*> задается один из следующих логических операторов.

1) логические операторы, связанные с устройством:

NU — устройство *j*, заданное в поле *A*, свободно;

U — устройство *j*, заданное в поле *A*, занято (в результате выполнения транзактм блока **SEIZE** или **PREEMPT**);

NI — устройство *j*, заданное в поле *A*, не прервано;

I — устройство *j*, заданное в поле *A*, обслуживает прерывание;

FV — устройство *j*, заданное в поле *A*, доступно;

FNV — устройство *j*, заданное в поле *A*, не доступно;

2) логические операторы, связанные с многоканальными устройствами:

SE — многоканальное устройство *j*, заданное в поле *A*, пусто (**S**[*j*]=0);

SNE — многоканальное устройство *j*, заданное в поле *A*, не пусто (**S**[*j*]<>0);

SF — многоканальное устройство *j*, заданное в поле *A*, заполнено (**R**[*j*]=0);

SNF — многоканальное устройство *j*, заданное в поле *A*, не заполнено (**R**[*j*]<>0);

SV — многоканальное устройство *j*, заданное в поле *A*, доступно;

SNV — многоканальное устройство *j*, заданное в поле *A*, не доступно;

3) логические операторы, связанные с логическими ключами:

LS — логический ключ *j*, заданный в поле *A*, включен;

LR — логический ключ *j*, заданный в поле *A*, выключен;

4) логические операторы, связанные с транзактами:

M — в блоке *j*, заданном в поле *A* блока **GATE**, находится в состоянии синхронизации транзакт, принадлежащее к тому же семейству, что и транзакт, находящееся в блоке GATE или пытающееся войти в этот блок;

NM — в блоке *j*, заданном в поле *A* блока GATE, в состоянии синхронизации нет ни одного транзакта, принадлежащего к тому же семейству, что и транзакт, пытающееся войти в блок **GATE.**

Поле *A* содержит имя или номер объекта, для которого проводится проверка. Операнд *A* может быть именем, положительным целым числом, СЧА или СЧА*****<параметр>.

Поле *B* содержит номер следующего блока для входящего транзакта, когда логический оператор имеет значение "ложь". Операнд *B* может быть именем, положительным целым числом, СЧА или СЧА*****<параметр>. Если поле *B* определёно, то оно должно содержать номер блока, допустимый для текущей модели.

Режимы условного и безусловного входа блока GATE

Блоки **GATE**, как и блоки **TEST**, могут работать в двух режимах: в режиме безусловного входа и в режиме условного входа. В режиме безусловного входа, если в поле *B* блока **GATE** задан номер следующего блока, транзакта никогда не задерживаются на входе блока **GATE**. Если заданный логический оператор имеет значение "истина", транзакт пытается перейти к следующему по номеру блоку. Если логический оператор имеет значение "ложь", то транзакта будут пытаться перейти к блоку, номер которого задан в поле *B* блока **GATE**. Выбор следующего блока производится один раз в момент входа транзакта в блок **GATE.**

В режиме условного входа, если поле *B* блока **GATE** пусто (альтернативный выход не задан), то транзакта не смогут войти в блок **GATE** до тех пор, пока указанный в этом блоке логический оператор не будет иметь значение "истина". Интерпретатор не проверяет значение логических операторов (исключениями являются операторы **M** и **NM**). В режиме условного входа задержанные транзакта помещаются в списки задержки и таким образом исключаются из числа сообщений, обрабатываемых интерпретатором, до тех пор, пока соответствующий логический оператор не примет значение "истина". Рассмотрим пример:

```
QUEUE   LINE1
GATE SV LINE1
DEPART  LINE1
```

В данном случае транзакта помещаются в список задержки, если память **LINE1** не доступна в момент, когда они пытаются войти в блок **GATE**. Когда память становится доступной, транзакта удаляются из списка и делают попытку войти в память.

Блоки GATE очень мощные, но они могут повлечь значительные расходы машинного времени на безуспешные попытки транзакта войти в блок. Чтобы уменьшить частоту безуспешных попыток вхождения в блок, можно поместить транзакта в список пользователя, используя блоки **LINK** и **UNLINK.**

СЧА класса **MB** не могут быть использованы для спецификации условий блокировки в блоке **GATE**. Для этого необходимо использовать блоки **MATCH**.

Когда транзакт не может войти в блок **GATE**, его индикатор задержки становится равным 1 и остается таким до тех пор, пока транзакт не войдет в блок **TRANSFER** в режиме **SIM**.

Блоки для обработки сообщений, принадлежащих одному семейству

Блоки **GENERATE** являются основным средством создания сообщений и ввода их в модель. Вход сообщений в блок **GENERATE** не допускается. Помимо блока **GENERATE**, для создания сообщений используется также блок **SPLIT**, который создает заданное число копий вошедшего в блок транзакта. Эти копии принадлежат к тому же семейству, что и породившее их транзакт.

Блок **TERMINATE** является основным средством уничтожения сообщений и удаления их из модели. Для удаления сообщений, принадлежащих к одному семейству, может быть также использован блок **ASSEMBLE**.

Блоки **MATCH** и **GATHER** предназначены для управления движением сообщений, принадлежащих к одному семейству. Для управления такими транзактами используются блоки **GATE** и **GATE NM**.

Блок SPLIT

Блок **SPLIT** имеет следующий формат:

```
SPLIT  <A>,[<B>],[<C>]
```

Блок **SPLIT** выполняет функцию копирования входящего в него транзакта, которое называется исходным или порождающим.

В поле *A* задается число создаваемых копий. Операнд *A* может быть именем, положительным целым, СЧА, СЧА*****<параметр>. Если вычисленное значение аргумента поля *A* равно нулю, то блок **SPLIT** не выполняет никаких операций. После создания копий транзакт пытается перейти к следующему по номеру блоку. Все копии формируются в момент входа порождающего транзакта в блок **SPLIT**.

Поле *B* задает номер следующего блока, к которому переходят копии исходного транзакта, причем значение вычисляется для каждой копии отдельно. Операнд *B* может быть именем, положительным целым, СЧА, СЧА*****<параметр>.

В поле *C* может быть задан номер параметра, используемого для присвоения копиям последовательных номеров. Операнд *C* может быть именем, положительным целым, СЧА, СЧА*****<параметр>.

Если, например, задан параметр *j*, то *j*-му параметру исходного транзакта и n — копиям этого транзакта будут присвоены значения, как показано ниже.

Пусть X — входное значение параметра *j*, тогда значение параметра *j* первой копии равно X+1, второй копии — X+2, третьей копии — X+3 и т. д.

Помимо значений параметров в каждую копию записывается значение приоритета и отметка времени исходного транзакта. Копии поочередно поступают в список текущих событий, причем каждая копия помещается в конец соответствующего приоритетного класса.

Счётчик общего числа входов (**N***j*) и счётчик текущего числа сообщений (**W***j*) блока **SPLIT** увеличиваются на единицу каждым исходным транзактм и каждой копией. Счётчик числа сообщений уменьшается на единицу при каждом выходе исходного транзакта или копии из блока **SPLIT.**

Каждая новая копия становится членом семейства сообщений, порожденного одним исходным транзактм, которое было создано блоком GENERATE**.**

Транзакта, принадлежащие к одному семейству, объединяются интерпретатором в список.

По связям внутри семейства сообщений нельзя установить, какое из сообщений семейства является порождающим. Если копия транзакта входит в блок SPLIT, то вторичная копия становится членом того же семейства, что и первичная копия. Таким образом, каждое транзакт является членом одного и только одного семейства. Семейство может состоять из произвольного числа сообщений. Когда транзакт уничтожается, интерпретатор автоматически исключает его из членов соответствующего семейства. Таким образом, семейство существует до тех пор, пока из модели не удаляется последний из его членов. При каждом удалении транзакта связи между ними корректируются так, чтобы транзакта данного семейства по-прежнему образовывали замкнутый список**.**

В модели одновременно может существовать произвольное число семейств, оно все время меняется, поскольку каждое генерируемое блоком GENERATE транзакт создает новое семейство.

Блок ASSEMBLE

Блок **ASSEMBLE** имеет следующий формат:

```
ASSEMBLE  <A>
```

Блок **ASSEMBLE** объединяет заданное число сообщений, принадлежащих к одному семейству, в одно транзакт (т. е. осуществляет сборку заданного числа сообщений). После сборки из блока **ASSEMBLE** выходит только одно транзакт, которое переходит в следующий по номеру блок. В одном и том же блоке **ASSEMBLE** возможна одновременная сборка сообщений нескольких семейств. Когда транзакт входит в блок **ASSEMBLE**, интерпретатор просматривает семейство, к которому принадлежит это транзакт, и проверяет, есть ли другое транзакт из того же семейства в данном блоке **ASSEMBLE.**

Поле *A* задает число сообщений, участвующих в сборке. Операнд *A* может быть именем, положительным целым, СЧА, СЧА*****<параметр>. Первона-

чальное значение аргумента поля *A* не должно быть больше или равно единице. Если при входе исходного транзакта в блок **ASSEMBLE**, значение счётчика стало равным нулю (т. е. нужно было "объединить" только одно транзакт), транзакт немедленно покидает блок **ASSEMBLE** и переходит в следующий по номеру блок. Если результат отрицательный (вычисленное значение аргумента поля *A* нулевое или отрицательное), происходит ошибка выполнения. Обычно значение счётчика сборки больше единицы, поэтому при входе в блок **ASSEMBLE** исходного транзакта результат вычитания единицы из счётчика положительный. Этот результат (новое значение счётчика сборки) сохраняется, а исходное транзакт исключается из списка текущих событий и переходит в состояние синхронизации. Это транзакт не будет возвращено в список текущих событий до тех пор, пока в блок **ASSEMBLE** не войдет заданное число сообщений, и счётчик сборки не станет равным нулю.

Счётчики **N***j* и **W***j* блока **ASSEMBLE** увеличиваются на единицу. Затем интерпретатор переходит к обработке следующего транзакта списка текущих событий.

При входе транзакта того же семейства в блок **ASSEMBLE** счётчик сборки уменьшается на единицу. Вновь прибывшее транзакт уничтожается. Если значение счётчика сборки все еще больше нуля, интерпретатор переходит к обработке следующего транзакта из списка текущих событий.

Если значение счётчика после вычитания единицы стало равным нулю, то исходное транзакт возвращается в список текущих событий и становится последним транзактм в своем приоритетном классе. Следовательно, может получиться так, что интерпретатор обработает ряд сообщений до того, как приступит к обработке транзакта, вышедшего из блока **ASSEMBLE**. Индикатор синхронизации устанавливается в "0", завершение сборки отмечается установкой флага изменения состояния.

После возвращения исходного транзакта в список текущих событий, интерпретатор начинает просмотр с начала списка. Это обеспечит обработку исходного транзакта в тот же момент условного времени, когда заканчивается сборка.

Даже если исходное транзакт не может войти в следующий по номеру блок, оно больше не считается участвующим в процессе сборки. Это связано с тем, что индикатор синхронизации транзакта устанавливается в "0". Следовательно, если другое транзакт из того же семейства поступит в блок **ASSEMBLE**, оно будет рассматриваться как исходное транзакт, и начнется новый процесс сборки. Счётчик **W***j* уменьшается на единицу при каждом выходе транзакта из блока **ASSEMBLE.**

Транзакта, прерванные во время пребывания в блоке ASSEMBLE

После того, как исходное транзакт было переведено в состояние синхронизации, может служиться так, что другое транзакт вошло в блок PREEMPT и генерирует прерывание устройства, занятого исходным транзактм, которое

участвует в процессе сборки в блоке **ASSEMBLE.** Транзакт, находящееся в блоке **ASSEMBLE** либо могло быть прервано ранее на каком-либо из занимаемых им устройств, либо это прерывание является первым.

Если транзакт, находящееся в блоке **ASSEMBLE**, ранее прервано не было, то выполняются следующие операции:

– индикатор прерывания устанавливается в единицу, указывая на то, что транзакт прервано;

– счётчик прерываний увеличивается на единицу;

– индикатор списка устанавливается в единицу, указывая на то, что транзакт находится в списке прерывания для соответствующего устройства.

Транзакта в блоке **ASSEMBLE**, которые одновременно находятся в состоянии прерывания и в состоянии синхронизации, возвращаются в список текущих событий только после завершения сборки и снятия прерывания.

Вход прерванных сообщений в блок ASSEMBLE

Пусть транзакт, занимающее устройство, находится в списке текущих событий. Может случиться так, что другое транзакт войдет в блок **PREEMPT** и вызовет прерывание на этом устройстве.

Транзакт, занимавшее устройство, не будет сразу же удалено из списка текущих событий и переведено в состояние прерывания.

Вместо этого установится в единицу флаг прерываний. Транзакт будет переведено в состояние прерывания только в тот момент, когда оно войдет в блоки **MATCH, ASSEMBLE** или **GATHER**, где будет переведено в состояние синхронизации.

Когда такое прерванное транзакт входит в блок **ASSEMBLE**, выполняются следующие операции:

– как всегда при входе исходного транзакта, индикатор синхронизации устанавливается в единицу;

– флаг прерывания устанавливается в "0";

– индикатор прерывания на устройстве устанавливается в единицу;

– индикатор списка прерываний устанавливается в единицу;

– транзакт удаляется из списка текущих событий.

Следует отметить, что прерванное на каком-либо устройстве транзакт могло занять это устройство также путем прерывания, выполнив блок **PREEMPT**. В этом случае выполняются те же операции, как и в случае, когда прерываемое транзакт заняло устройство, выполнив блок **SEIZE.**

Примеры использования блоков SPLIT и ASSEMBLE

Рассмотрим фрагмент программы моделирования параллельных операций ввода-вывода.

```
.
      SEIZE     CPU
      SPLIT     1,PCA2
      ADVANCE   FN$TIM1
      RELEASE   CPU
```

```
PAE1  ASSEMBLE  2
.
.
.
PCA2  SEIZE     ARM
      ADVANCE   FN$TIM2
      SEIZE     CHAN
      ADVANCE   FN$TIM3
      RELEASE   CHAN
      RELEASE   ARM
      TRANSFER   ,PAE1
```

Транзакт занимает устройство, которое представляет собой центральный процессор (**CPU**) и порождает одну копию, при помощи которой моделируется операция ввода-вывода. Затем исходное транзакт моделирует выполнение программы процессором, задерживаясь на соответствующее время в блоке **ADVANCE**. Затем это транзакт освобождает процессор и входит в блок **ASSEMBLE**, в котором ожидает окончания операции ввода-вывода. Тем временем копия моделирует выполнение операции ввода-вывода. Копия занимает устройство доступа (устройство **ARM**), занимает канал, задерживается на время моделирования операций считывания или записи в блоке **ADVANCE,** затем освобождает канал, освобождает устройство доступа и поступает в блок **ASSEMBLE**. После этого исходное транзакт может продолжать движение.

Рассмотрим фрагмент программы передачи информационных сообщений.

```
.
       SEIZE     LINE1
       ASSIGN    1,FN21
       ADVANCE   FN$SEG
       TRANSFER  ,PBE2
PBC2   SPLIT     1,PBG2
       ADVANCE   FN$SEG
PBE2   LOOP      1,PBC2
       RELEASE   LINE1 PBG2 .
.
```

Транзакт занимает устройство **LINE1**, изображающее линию связи. Число сегментов в каждом информационном транзакте задано функцией **21** в параметре **1**. Прием одного сегмента информационного транзакта изображается в модели при помощи задержки соответствующего транзакта в блоке **ADVANCE.** Транзакт затем переходит к блоку **LOOP**. Если транзакт состоит из двух или более сегментов, оно передается в блок **SPLIT**, где порождает копию. Копия используется для моделирования обработки переданного сегмента. Исходное транзакт входит в блок **ADVANCE** для моделирования передачи следующего сегмента транзакта. После того, как будет закончена передача n-го сегмента, значение параметра **1** станет равным нулю в блоке **LOOP**, и транзакт из блока **LOOP** перейдет к следующему по номеру блоку. Затем транзакт освобождает линию связи (устройство **LINE1**) и переходит к моделированию обработки последнего (n-го) сегмента.

Блок GATHER

Блок **GATHER** имеет следующий формат записи:

```
GATHER  <A>
```

Блок **GATE** накапливает заданное число сообщений, принадлежащих к одному семейству. Ни одно из накапливаемых сообщений не уничтожается. Когда в блоке **GATHER** накопится заданное число сообщений, все эти транзакта одновременно попытаются войти в следующий по номеру блок. Транзакта никогда не задерживаются на входе блока **GATHER**. Одновременно в одном блоке **GATHER** может происходить накопление сообщений нескольких семейств.

При входе транзакта в блок **GATHER**, интерпретатор просматривает семейство, к которому оно принадлежит, и проверяет, находится ли в данном блоке **GATHER** другое транзакт из того же семейства.

Поле *A* задает число сообщений, принадлежащих к одному семейству, которое нужно накопить. Операнд *A* может быть именем, положительным целым, СЧА, СЧА*****<параметр>. Если вычисленное значение счётчика накапливания нулевое или отрицательное, происходит ошибка выполнения. Начальное значение счётчика должно быть больше или равно единице. Если в блок GATHER не вошло заданное число сообщений, транзакта, находящиеся в блоке, остаются в состоянии синхронизации.

Если первое транзакт является единственным членом семейства, происходит ошибка выполнения.

При входе первого из накапливаемых сообщений в блок **GATHER** выполняются операции, аналогичные операциям, выполняемым при входе исходного транзакта в блок **ASSEMBLE**. Счётчик числа накапливаемых сообщений, начальное значение которого задается полем *A* блока GATHER, уменьшается на единицу. Транзакт удаляется из списка текущих событий и переходит в состояние синхронизации.

Счётчики **N***j* и **W***j* блока **GATHER** увеличиваются на единицу. Индикаторы списков устанавливаются в "0", а индикатор синхронизации устанавливается в единицу. Затем интерпретатор переходит к обработке следующего транзакта в списке текущих событий.

При входе последующих сообщений в блок **GATHER** обнаруживается, что в блоке уже есть одно или несколько сообщений из того же семейства, участвующих в накоплении. Значение счётчика числа накапливаемых сообщений уменьшается на единицу. Если значение счётчика положительное, входящее транзакт удаляется из списка текущих событий. Транзакт включается в список накапливаемых сообщений, где транзакта располагаются в порядке поступления (первым входит, первым выходит). Счётчики **N***j* и **W***j* увеличиваются на единицу. Если значение числа накапливаемых сообщений после вычитания единицы стало равным нулю, все накапливаемые транзакта

возвращаются в список текущих событий, и каждое из них становится последним среди сообщений с таким же значением приоритета. Транзакта возвращаются в список текущих событий в следующем порядке:

– первым возвращается в список текущих событий первое из накапливаемых сообщений;

– остальные накопленные транзакта возвращаются в список текущих событий в том порядке, в каком они поступили в блок **GATHER**. Последним возвращается транзакт, при входе которого в блок **GATHER** значение счётчика стало равным нулю.

Завершение процесса накопления сообщений приводит к изменению состояния процедуры просмотра, т. е. устанавливается флаг изменения состояния. После того, как все накопленные транзакта (за исключением сообщений, прерванных на одном или нескольких устройствах) возвращаются в список текущих событий, интерпретатор возвращается к началу списка. Таким образом, обеспечивается обработка всех возвращенных в список сообщений в тот же момент условного времени, когда завершается процесс накопления.

Транзакта, прерванные в блоке GATHER

Допустим, что одно из сообщений, участвующих в процессе накопления, занимает некоторое устройство. Может случиться так, что во время прерывания этого транзакта в блоке **GATHER**, другое транзакт вызовет прерывание на устройстве, занятом первым транзактм. В этом случае для прерванного транзакта выполняются точно такие же операции, как для транзакта, прерванного во время пребывания в блоке **ASSEMBLE.**

Вход прерванных сообщений в блок GATHER

В момент, когда некоторое транзакт генерирует прерывание для какого-либо устройства, транзакт, занимающее это устройство, может находиться в списке текущих событий. В этом случае для прерванных сообщений при входе в блок **GATHER** выполняются точно такие же операции, как и для прерванных сообщений, входящих в блок **ASSEMBLE.**

Блок MATCH

Блок **MATCH** имеет следующий формат:

```
MATCH  <A>
```

Блок **MATCH** используется для синхронизации движения двух сообщений, принадлежащих к одному семейству, без удаления этих сообщений из модели.

Блоки **MATCH** не объединяют синхронизируемые транзакта. Синхронизация осуществляется путем подбора пар сообщений из одного семейства и задержки этих сообщений до тех пор, пока оба транзакта из одной пары не поступят в заданные точки модели. Транзакта никогда не задерживаются в блоке **MATCH.** Транзакта, для которых выполнилось условие синхронизации,

переходят к следующему по номеру блоку. В одной паре блоков MATCH могут одновременно находиться в состоянии синхронизации пары сообщений из различных семейств. Возможна также одновременная синхронизации пар сообщений из одного семейства в нескольких блоках **MATCH.**

Поле *A* задает имя или номер другого блока **MATCH**, называемого "сопряженным блоком **MATCH**". Если такого блока нет, происходит останов по ошибке. Операнд *A* может быть именем, положительным целым, СЧА, СЧА*****<параметр>.

Допускается использование блока **MATCH** в качестве сопряженного самому себе. В этом случае блок MATCH действует как блок **GATHER** с начальным значением счётчика, равным **2**. При входе транзакта в блок **MATCH** интерпретатор определяет номер блока, заданного в поле *A*, и затем просматривает семейство, к которому принадлежит вошедшее в блок **MATCH** транзакт.

Если для транзакта, вошедшего в блок **MATCH**, нет транзакта из того же семейства, находящегося в состоянии синхронизации в сопряженном блоке **MATCH**, то это транзакт удаляется из списка текущих событий и помещается в список синхронизации. Это транзакт не возвращается в список текущих событий до тех пор, пока другое транзакт из этого же семейства не войдет в сопряженный блок **MATCH**. Счётчики **N***j* и **W***j* блока увеличиваются на единицу. Индикатор синхронизации устанавливается в единицу, как в блоках **ASSEMBLE** и **GATHER**. Интерпретатор переходит затем к следующему транзакту из списка текущих событий.

Если в сопряженном блоке **MATCH** есть транзакт из того же семейства и оно находится в состоянии синхронизации, то индикатор синхронизации устанавливается в "0". Затем транзакт, которое находилось в сопряженном блоке **MATCH**, возвращается в список текущих событий, где оно становится последним среди сообщений с таким же значением приоритета. Тем временем интерпретатор продолжает обработку второго транзакта данной пары, вошедшего в рассматриваемый блок **MATCH**. Это транзакт обрабатывается так, как если бы оно проходило через блок **ADVANCE** с нулевой задержкой.

Выполнение условия синхронизации рассматривается интерпретатором как изменение состояния процедуры просмотра, т. е. устанавливается флаг изменения состояния. После того, как первое транзакт из пары возвращено в список текущих событий, второе транзакт, находящееся в рассматриваемом блоке MATCH, пытается пройти столько блоков с нулевой задержкой, сколько сможет. Как только движение этого транзакта будет заблокировано или войдет в блок **ADVANCE** с положительной задержкой, интерпретатор обнаружит, что флаг изменения состояния установлен, и вернется к началу списка текущих событий. Таким образом обеспечивается обработка первого из пары сообщений в момент выполнения условия синхронизации.

Может случиться так, что транзакт, для которого выполнилось условие синхронизации, не может сразу выйти из блока **MATCH** и войти в следующий

по номеру блок. Несмотря на то, что транзакт остается в блоке **MATCH**, оно не может быть синхронизировано с другим транзактм, поскольку его индикатор синхронизации установлен в "0". Счётчик числа сообщений в блоке **MATCH** (**W***j*) уменьшается на единицу при выходе из блока каждого транзакта, для которого выполнилось условие синхронизации.

Транзакта, прерванные во время пребывания в блоке MATCH

Допустим, что транзакт, переведенное в состояние синхронизации в блоке **MATCH**, ранее заняло некоторое устройство. Может случиться так, что во время пребывания этого транзакта в блоке **MATCH**, другое транзакт войдет в блок **PREEMPT** и вызовет прерывание первого транзакта на занимаемом им устройстве. Для прерванных в блоке **MATCH** сообщений выполняются точно такие же операции, как для сообщений, прерванных в блоке **ASSEMBLE.**

Вход прерванных сообщений в блок MATCH

В момент, когда транзакт входит в блок **PREEMPT** и генерирует прерывание на каком-либо устройстве, транзакт, которое занимало это устройство, может находиться в списке текущих событий. В этом случае выполняются операции, как и при входе прерванных сообщений в блок **ASSEMBLE.**

Стандартным числовым атрибутом, связанным с описываемым оператором является:

MB<номер блока> — флаг синхронизации. Возвращает 1, если транзакт, находящееся в блоке <номер блока> принадлежит к тому же семейству, что и текущее.

Блоки GATE M и GATE NM

Блоки **GATE M** и **GATE NM** проверяют значения логических условных операторов **M** и **NM** и, в зависимости от результатов проверки, управляют движением сообщений одного семейства. Блок **GATE** может задержать транзакт на входе, если не задан альтернативный выход. Логические условные операторы, записываемые во вспомогательном поле операции имеют следующие значения:

M — имеет значение "истина", если другое транзакт, принадлежащее к тому же семейству, что и транзакт, вошедшее в блок **GATE** или находящееся в блоке **GATE**, находится в состоянии синхронизации в блоке, номер которого определяется полем *A* блока **GATE;**

NM — имеет значение "истина", если в блоке с номером, определяемым значением поля *A* блока **GATE**, нет сообщений, принадлежащих к тому же семейству, что и транзакт, вошедшее в блок **GATE** или находящееся в блоке **GATE.**

Поле *A* задает номер блока, в котором проверяется выполнение условия синхронизации. Считается, что транзакт находится в состоянии синхрониза-

ции, если индикатор синхронизации установлен в единицу. Условие синхронизации выполняется в следующих трех случаях:

1) другое транзакт, принадлежащее к тому же семейству, что и транзакт в блоке **GATE**, является исходным транзактм в блоке **ASSEMBLE**, номер *j* которого является аргументом поля *A* блока **GATE**;

2) одно или более сообщений, принадлежащих к тому же семейству, что и транзакт в блоке **GATE**, участвует в процессе накопления в блоке **GATHER**, номер *j* которого является значением аргумента поля *A* блока **GATE**;

3) другое транзакт из того же семейства, что и транзакт в блоке **GATE**, находится в блоке MATCH, номер *j* которого является значением аргумента поля *A* блока GATE и ожидает поступления транзакта в сопряженный блок MATCH.

Для выполнения условия синхронизации необходимо, чтобы индикаторы синхронизации сообщений, участвующих в проверке условия, были установлены в 1. Поэтому для сообщений, которые закончили сборку, накопление или дождались поступления транзакта в сопряженный блок **MATCH**, но не могут войти в следующий по номеру блок, условия синхронизации не выполняются, т.к. индикаторы синхронизации этих сообщений установлены в "0".

Наличие в поле *B* блока **GATE** аргумента, определяющего номер следующего блока, указывает на то, что транзакта не задерживаются на входе блока **GATE**. Если логический условный оператор (**M** или **NM**) имеет значение "истина", то транзакта переходят к следующему по номеру блоку; если значение атрибута — "ложь", то транзакт переходит к блоку, номер которого определяется значением аргумента поля *B*. Если номер следующего блока не задан, блок **GATE** будет задерживать транзакт на входе до тех пор, пока указанный логический условный оператор не примет значение "истина". Тогда транзакт может перейти к блоку, следующему по номеру за блоком **GATE**.

Транзакта, которые не могут войти в блоки **GATE M** и **GATE NM**, работающие в режиме условного входа, не удаляются из списка текущих событий, и интерпретатор пытается продвинуть эти транзакта при каждом просмотре. Таким образом, применение блоков **GATE M** и **GATE NM** в режиме условного входа может привести к увеличению времени счета. Следует отметить, что состояние других членов семейства, к которому принадлежит транзакт, не меняется, когда это транзакт входит в блоки **GATE M** и **GATE NM**, проверяющие одно из условий синхронизации.

3.2 Системные числовые атрибуты (System Numerical Attributes)

В процессе работы программы транзакты перемещаются по модели от блока к блоку, в некоторых блоках они задерживаются, в некоторых изменяются их параметры. Одновременно с этим сама программа собирает инфор-

мацию о процессах в блоках. Например, для всех *Очередей* собирается следующая информация: число вошедших и вышедших транзактов, а также время нахождения транзакта в очереди. Одновременно производится анализ этой информации и выполняются некоторые простейшие статистические вычисления, определяются: максимальное, минимальное и среднее число транзактов в очереди; минимальное, максимальное и среднее время нахождения транзакта в очереди; всё это — с учётом и без учёта транзактов, которые вошли в очередь, но прошли её без задержки по любым причинам (например, в очереди никого не было или они имели приоритет).

Вся эта информация может быть использована в процессе моделирования для выбора действий в качестве условий формирования тех или иных событий. Для получения доступа к этим статистическим параметрам используются специальные средства — *Системные числовые атрибуты* (СЧА) (System Numerical Attributes). Они представляют собой особую форму записи подпрограмм-функций, извлекающих необходимую информацию из внутренних массивов данных.

Каждый тип устройств (*Очереди*, *Обслуживающие устройства*, *Многоканальные устройства*, *Матрицы* и др.) имеют свой набор СЧА. Общий вид записи СЧА: *X***$***Y*, где знак доллара является условным обозначением СЧА, слева от которого записывается условное обозначение типа СЧА (в виде одной буквы), а справа — имя конкретного блока, информацию о котором требуется получить. Например, выражение **Q$Ochered** означает, что требуется узнать текущий размер очереди (**Q**) с именем **Ochered**. В этом смысле СЧА можно считать также упорядоченным видом переменных, так как результатом их работы является извлечение и предоставление значения соответствующего параметра. С ними можно работать, как с обычными переменными, т. е. умножать, использовать для определения номера блока (блоки можно задавать именами или номерами или и тем и другим способом одновременно) и т. д. (В современных языках программирования СЧА могли бы записываться, например, в виде функции, тогда это могло бы выглядеть так: **Q(Ochered)**). При вызове они возвращают числовые или строковые значения указанных переменных и могут быть использованы в GPSS в качестве *операндов* (в операторах и командах) или в *выражениях* (например, в *Окнах* для просмотра результатов).

СЧА делятся на различные классы: СЧА *Очередей*, СЧА *Обслуживающих устройств*, СЧА общесистемные и др. Вид СЧА и порядок их использования показан ниже.

Системные Числовые Атрибуты — спецификаторы устройств (SNA Entity Specifiers)

Большинство СЧА могут быть представлены в одной из нескольких форм, начинающихся с наименования СЧА-класса. Например, выражение **W22** при вызове возвращает номер транзакта, находящегося в блоке № 22

(все блоки имеют наименование, например, очереди с различными именами, или номера). Этот номер присваивается транслятором при трансляции программы по порядку сверху вниз и виден в *Стандартном отчёте*. СЧА-класс задаётся первой буквой — в данном случае **W**.

СЧА могут быть записаны в следующих формах (для примера используется класс **W**):

W*j* — где j (положительное целое число) — номер блока в моделировании.

W$*Name* — где *Name* — название (обозначение) запрошенного блока, символ доллара **$** — знак присоединения к имени блока.

W*** *j* — где j (положительное целое число) — номер параметра активного транзакта, содержащего номер запрашиваемого блока (косвенная адресация); символ ***** — знак присоединения номера блока при косвенной адресации.

W*** *Name* — где *Name* — название параметра активного транзакта, содержащего номер запрашиваемого блока (косвенная адресация); символ ***** — знак присоединения номера блока при косвенной адресации.

W*$*Name* — где *Name* — название параметра активного транзакта, содержащего номер запрашиваемого блока. Знак **$** не является необходимым, но используется как разделитель. По сути, это выражение эквивалентно выражению **W***** *Name* (косвенная адресация).

W*** *Parameter* — означает, что тот или иной вид записи **W***** *j*, **W***** *Name* или **W*$***Name* может быть использован.

СЧА-класс **MX** для *Матриц* является особым случаем. Он содержит 3 косвенных адреса. Например, выражение

```
MX*Sales(*Partnumber,*January)
```

определяет *Матрицу*, чей номер находится в параметре с названием **Sales** (активного транзакта) и обращается к элементу *Матрицы*, имеющему номер строки **Partnumber** и номер столбца **January**. Обычно для инициализации параметров (в данном случае **Sales**, ***Partnumber** и ***January**) у активного в данный момент транзакта используется блок **ASSIGN**.

СЧА-классы **A1**, **AC1**, **C1**, **M1**, **MP**, **PR** и **TG1** отражают общесистемные переменные, не имеют параметров и используются сами по себе.

При программировании в GPSSW можно использовать подсказку о возможных формах использования СЧА в качестве операндов.

Доступные системные числовые атрибуты

Следующие СЧА могут быть использованы в качестве операндов в операторах (командах) и в выражениях. Во всех случаях *Entnum* заменяется спецификатором устройства, который может быть именем (располагается справа вплотную от разделителя **$**) или номером, или (для косвенной адресации) он может быть заменён на символ *****, за которым следует имя или номер.

Ниже приведён полный список СЧА в GPSS World:

A1 — номер ансамбля активного транзакта (целое число). Ансамбль появляется, когда в системе находятся родственные транзакты, полученные, например, после создания копий одного из транзактов (родителя) оператором **SPLIT**.

AC1 — значение абсолютного времени моделирования — начиная с использования последней команды **CLEAR** (действительное число), а при её отсутствии — с момента начала моделирования. Моделирование может прерываться (командами **HALT**, в связи с окончанием заданного времени и по другим причинам), а затем продолжаться. Абсолютное время показывает всё время моделирования.

BV*Entnum* — значение булевой (логической) переменной с именем или номером *Entnum* (действительное число).

C1 — значение относительного системного времени моделирования с момента последнего использования команды **RESET** (действительное число).

CA*Entnu* — среднее число транзактов в *Цепи пользователя* с названием *Entnum* (действительное число).

CC*Entnum* — общее число входов в *Цепь пользователя* с именем *Entnum* (целое число).

CH*Entnum* — текущее значение числа транзактов в *Цепи пользователя* с именем *Entnum* (целое число).

CM*Entnum* — максимальное число транзактов в *Цепи пользователя* с именем *Entnum* (целое число).

CT*Entnum* — среднее время нахождения транзактов в *Цепи пользователя* с именем *Entnum* (действительное число).

F*Entnum* — флаг занятости *Обслуживающего устройства*. Если *Обслуживающее устройство* с именем *Entnum* в настоящее время занято, **F***Entnum* возвращает 1, в противном случае возвращается 0 (целое число).

FC*Entnum* — число захватов *Обслуживающего устройства* с именем *Entnum* с помощью команд **SEIZE** или **PREEMPT** (целое число).

FI*Entnum* — флаг прерывания *Обслуживающего устройства*. Если *Обслуживающее устройство* с именем *Entnum* в настоящее время перехвачено (транзактом с более высоким приоритетом), **FI***Entnum* возвращает 1, иначе возвращается 0 (целое число).

FN*Entnum* — функция. Возвращает результат выполнения функции с именем *Entnum* (действительное число).

FR*Entnum* — коэффициент использования *Обслуживающего устройства*, равный отношению времени занятости транзактами *Обслуживающего устройства* с именем *Entnum*. к общему времени моделирования (действительное число).

FT*Entnum* — среднее время удержания транзактом *Обслуживающего устройства* с именем *Entnum* (действительное число).

FV*Entnum* — флаг готовности *Обслуживающего устройства*. Если *Обслуживающее устройство* с именем *Entnum* готово принять очередной транзакт, то **FV***Entnum* возвращает 1, если не готово — возвращается 0 (целое число).

GN*Entnum* — счётчик *Цифровой группы*. **GN***Entnum* возвращает количество членов *Цифровой группы* с именем *Entnum* (целое число).

GT*Entnum* — счётчик *Группы транзактов*. **GT***Entnum* возвращает количество членов *Группы транзактов* с именем *Entnum* (целое число).

LS*Entnum* — значение *Логического ключа* с именем *Entnum*. **LS***Entnum* возвращает 1 или 0 (целое число). *Логические ключи* — это переменные, которые используются в программе для запоминания какого-то события и выстраивания логики работы программы. Например, можно с помощью *Логического ключа* фиксировать включенность или выключенность какого-то устройства. *Логических ключей* может быть много и поэтому они имеют свои уникальные имена.

MB*Entnum* — проверка принадлежности транзакта в блоке с именем *Entnum* и активного транзакта одному *Ансамблю*. **MB***Entnum* возвращает 1, если оба транзакта принадлежат одному ансамблю (целое число). *Ансамбли* нужны, например, при моделировании сборки сложного устройства из совокупности деталей. Может моделироваться сборка нескольких устройств и тогда будут существовать несколько *Ансамблей*. Чтобы не перепутать детали, их проверяют на предназначенность соответствующему устройству.

MP*Parameter* — время перехода: текущее абсолютное системное время минус значение параметра с именем *Parameter* (действительное число).

MX*Entnum*(m,n) — сохраняемое значение *Матрицы*. Возвращает значение элемента матрицы в строке *m* и столбце *n* матрицы *Entnum*.

M1 — время перехода. **M1** возвращает абсолютное системное время минус "Временная Метка" транзакта (действительное число).

N*Entnum* — счётчик входов в блок. Сообщает общее число входов транзактов в блок с именем *Entnum* (целое число).

P*Parameter* или ***Parameter* — значение параметра с именем *Parameter* активного транзакта (целое или действительное числа, строка). При косвенной адресации используют форму SNA***Parameter*.

PR — приоритет активного транзакта (целое число в диапазоне 0…127).

Q*Entnum* — текущее число транзактов в *Очереди* с именем *Entnum* (целое число).

QA*Entnum* — среднее число транзактов в *Очереди* с именем *Entnum* за время моделирования (действительное число).

QC*Entnum* — общее число входов транзактов в *Очередь* с именем *Entnum* (целое число).

QM*Entnum* — максимальное число транзактов в *Очереди* с именем *Entnum* за время моделирования (целое число).

QT*Entnum* — среднее время нахождения транзактов в *Очереди* с именем *Entnum* (действительное число).

QX*Entnum* — среднее время нахождения транзактов в *Очереди* с именем *Entnum*, исключая "нулевые входы", когда транзакт входит в *Очередь* и тут же её покидает, не теряя на это время (действительное число).

QZ*Entnum* — число "нулевых входов" транзактов в *Очередь* с именем *Entnum*, когда транзакты проходят *Очередь* не задерживаясь (целое число).

R*Entnum* — размер незанятой ёмкости *Многоканального Устройства* с именем *Entnum* (целое число).

RN*Entnum* — случайное число. **RN***Entnum* возвращает числа в диапазоне 0…999 генератора случайных чисел с именем (номером) *Entnum* (целое число).

S*Entnum* — количество транзактов, находящихся в данный момент в *Многоканальном устройстве* с именем *Entnum*. **S***Entnum* возвращает число транзактов, вошедших в *Многоканальное устройство* с именем *Entnum* (целое число).

SA*Entnum* — среднее значение использованной ёмкости *Многоканального устройства* с именем *Entnum* (действительное число).

SC*Entnum* — число использований транзактами *Многоканального устройства* с именем *Entnum* (целое число).

SE*Entnum* — незанятость *Многоканального устройства*. **SE***Entnum* возвращает 1, если *Многоканальное устройство* с именем *Entnum* полностью доступен, возвращает 0 в ином случае (целое число).

SF*Entnum* — заполненность *Многоканального устройства*. **SF***Entnum* возвращает 1, если *Многоканальное устройство* с именем *Entnum* полностью заполнено, возвращает 0 в ином случае (целое число).

SR*Entnum* — коэффициент использования *Многоканального устройства*. Отношение среднего числа использований *Многоканального устройства Entnum* к общей его ёмкости (действительное число в диапазоне 0…1000).

SM*Entnum* — максимальное значение использования *Многоканального устройства* с именем *Entnum* (целое число).

ST*Entnum* — среднее время удержания на единицу *Многоканального устройства* с именем *Entnum* (действительное число).

SV*Entnum* — готовность *Многоканального устройства*. **SV***Entnum* возвращает 1, если *Многоканальное устройство* с именем *Entnum* в состоянии готовности, возвращает 0 в ином случае (целое число).

TB*Entnum* — среднее значение невзвешенных аргументов *Таблицы Entnum*. (действительное число).

TC*Entnum* — число включений в *Таблицу Entnum*. (целое число).

TD*Entnum* — среднеквадратическое отклонение для *Таблицы Entnum*. (действительное число).

TG1 — число, записанное в *Счётчике завершеня*. **TG1** возвращает число, которое остаётся после работы блока **TERMINATE** с положительным операндом *A*. Это значение первоначально задаётся командой **START** и для продолжения моделирования должно быть больше 0 (целое число).

V*Entnum* — результат вычисления арифметической или с плавающей точкой переменной с именем *Entnum* (действительное число).

W*Entnum* — текущее (на данный момент) число транзактов в блоке с именем *Entnum* (целое число).

X*Entnum* — значение *Сохраняемой величины* с именем *Entnum* (целое или действительное числа, строка).

XN1 — номер активного транзакта (целое число).

Z1 — размер свободной оперативной памяти компьютера (целое число).

3.3 Математические операции в GPSS World

Операторы математических операций используются для того, чтобы комбинировать элементы в математических выражениях. В GPSS при выполнении математических операций все значения переменных автоматически приводятся к численным, даже если они были заданы как литеральные.

Математические операции в GPSS World:

^*Выражение* — возведение в степень. Пример: **A^B** возвращает **A** в степени **B**.

(или *****) — умножение. Пример: **A#B** (или **A*B**) возвращает произведение **A** на **B**. По умолчанию символ **#** является знаком умножения, а символ ***** (в первой колонке программы) — индикатором строки примечаний. Но можно поменять значения этих символов на обратные через меню **Edit** → **Settings…** → **Switch # and ***. Для перехода в режим, когда умножение задаётся звездочкой *****, а комментарии решёточкой **#**, необходимо поставить галочку в окошке рядом с указателем **Switch # and ***.

/ — деление. Пример: **A/B** возвращает частное деления **A** на **B**.

**** — целочисленное деление. Пример: **A\B** возвращает целую часть результата деления **A** на **B**.

@ — целый остаток. **A@B** возвращает целвый остаток от деления **A** на **B**.

- — вычитание. Пример: **A-B** возвращает результат вычитания **B** из **A**.

+ — сложение. Пример: **A+B** возвращает сумму **A** и **B**.

>= 'GE' — "больше или равно". Пример: **A>=B** возвращает 1, если условие выполнено, и 0 — если не выполнено. Используется, в частности, с оператором **TEST**.

<= 'LE' — "меньше или равно". Пример: **A<=B** возвращает 1, если условие выполнено, и 0 — если не выполнено. Используется, в частности, с оператором **TEST**.

> 'G' — "больше". Пример: **A>B** возвращает 1, если условие выполнено, и 0 — если не выполнено. Используется, в частности, с оператором **TEST**.

< 'L' — "меньше". Пример: **A<B** возвращает 1, если условие выполнено, и 0 — если не выполнено. Используется, в частности, с оператором **TEST**.

= 'E' — "равно". Пример: **A=B** возвращает 1, если условие выполнено, и 0 — если не выполнено. Используется, в частности, с оператором **TEST**.

!= 'NE' — "не равно". Пример: **A!=B** возвращает 1, если условие выполнено, и 0 — если не выполнено.

& 'AND' — логическое "И". Пример: **A&B** возвращает 1, если оба операнда **A** и **B** не 0, и 0 — если хотя бы один из них 0.

| 'OR' — логическое "ИЛИ". Пример: **A'OR'B** возвращает 1, если хотя бы один из операндов **A** или **B** не 0, 0 — если оба они 0.

Приоритет математических операций в выражении:

^ возведение в степень;

(or *****) **/ ** умножение, деление, целочисленное деление;

@ целый остаток

- + вычитание и сложение;

>= <= >< операции сравнения;

= != равенство и неравенство;

& логическое "И";

| логическое "ИЛИ".

4 Визуализация результатов имитационного моделирования

4.1 Общие принципы визуализации результатов имитационного моделирования

Для вывода результатов имитационного моделирования в различных формах и представлениях используются следующие инструменты:

- *Журнал* ***Journal***;
- *Стандартный отчёт* ***Standard Report***;
- *Окна моделирования* ***Simulation Windows***;
- *Снимки моделирования* ***Simulation Snapshot***.

Журнал отражает все операции, совершаемые пользователем, и операции, выполняемые по его указаниям программой. *Журнал* можно сохранить в файле и затем проанализировать работу программы. При непрерывном моделировании в *Журнале* сохраняется информация о начале и окончании каждого процесса, а также возникшие ошибки. Наиболее эффективно использование *Журнала* в пошаговом режиме моделирования. На каждом шаге указывается, какой именно транзакт в данный момент является активным, в каком блоке он находится и в какой блок будет переходить, а также печатается комментарий к блоку. Если делать тщательно продуманные комментарии к блокам, то можно получить связный рассказ о работе программы, что помогает как при отладке, так и при исследовании сложных ветвящихся процессов.

Стандартный отчёт содержит основную информацию о программе (её текст, наличие блоков различных типов и др.), а также результаты моделирования в виде наиболее часто встречающихся статистических показателей. Содержимое *Стандартного отчёта* можно настраивать.

Окна моделирования предназначены для вывода информации в числовой или графической формах о переменных, имеющихся в программе. Существуют универсальные окна, например, для вывода графиков, а также специализированные, например, для построения гистограммы или для анимации движения транзактов через блоки в модели.

Снимки моделирования предназначены для фиксации в заданные моменты времени состояния программы с целью последующего анализа или её отладки. Фиксируются состояния *Цепи текущих событий*, *Цепи будущих событий*, *Цепи пользователя*, групп однородных транзактов, положение конкретного транзакта.

Все окна можно располагать рядом и одновременно наблюдать проходящие процессы с разных точек зрения и в различной форме. Но открытие окна влечёт за собой начало специальных вычислительных процессов, свя-

занных с этим окном, поэтому время счёта с каждым открытым окном увеличивается.

В целом эти инструменты визуализации результатов гармонично сочетаются между собой и создают основу *технологии имитационного моделирования* с помощью программы GPSS World.

4.2 Журнал Journal

Журнал формируется автоматически (если его формирование не отменено) и всегда появляется в отдельном окне. К нему можно перейти с помощью команды меню **Window → имя_файла — JOURNAL.** После трансляции программы в журнале появляется первая запись, например (Рисунок 4.1):

Рисунок 4.1 — Фрагмент *Журнала* сразу после трансляции при отсутствии в тексте программы команды **START**

Эти записи означают:

10 апреля 2005 года в 13 часов 44 минуты 55 секунд Началась трансляция модели.

10 апреля 2005 года в 13 часов 44 минуты 55 секунд Готово.

Если задать команду **START** и ввести цифру 100, то в *Журнале* будет получена следующая запись (Рисунок 4.2).

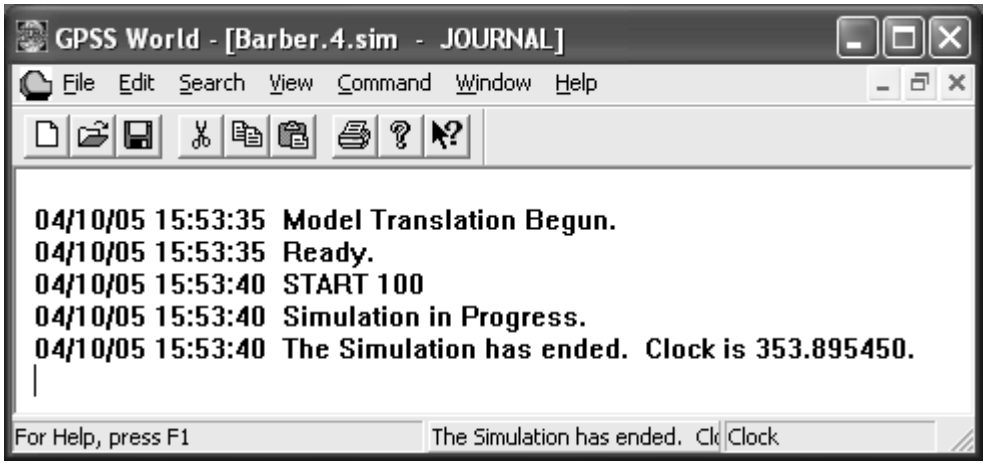


Рисунок 4.2 — Фрагмент *Журнала* после команды **START**

Она содержит следующую информацию (после данных о дате и времени):

• В *Счётчик завершения* записано 100 циклов до окончания работы программы

• Моделирование идёт

• Моделирование окончилось. Время 353.895450 ед. модельного времени.

Если вместо задания команды **START** нажать клавишу пошаговой работы **F5**, то в *Журнале* будет отражаться каждый шаг программы (Рисунок 4.3):

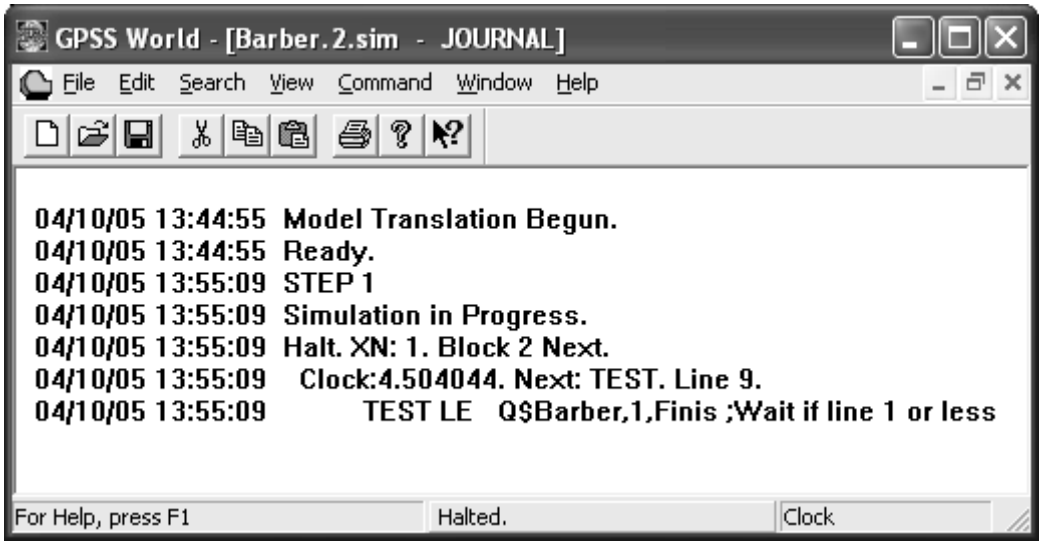


Рисунок 4.3 — Фрагмент *Журнала* после первого шага при пошаговой работе с помощью клавиши **F5**

После первого шага (начиная с третьей строки) информация имеет следующий смысл:

• Шаг 1

• Моделирование идёт.

• Остановка. Активный транзакт (**XN**) имеет № 1. Следующий блок № 2.

• Модельное время 4,504044 ед. времени. Следующий блок: **TEST**. Строка 9 (в ней располагается оператор **TEST**).

• Приводится вид следующей строки № 9 из текста программы и комментарий (та его часть, которая расположена в той же строке, что и оператор).

При следующем нажатии клавиши пошагового моделирования **F5** *Журнал* записывает следующую информацию (Рисунок 4.4):

• Шаг 1

• Моделирование идёт.

• Остановка. Активный транзакт (XN) имеет № 1. Следующий блок № 3.

• Модельное время 4,504044 ед. времени. Следующий блок: SAVEVALUE. Строка 11.

• Приводится вид следующей строки № 11 из текста программы и комментарий.

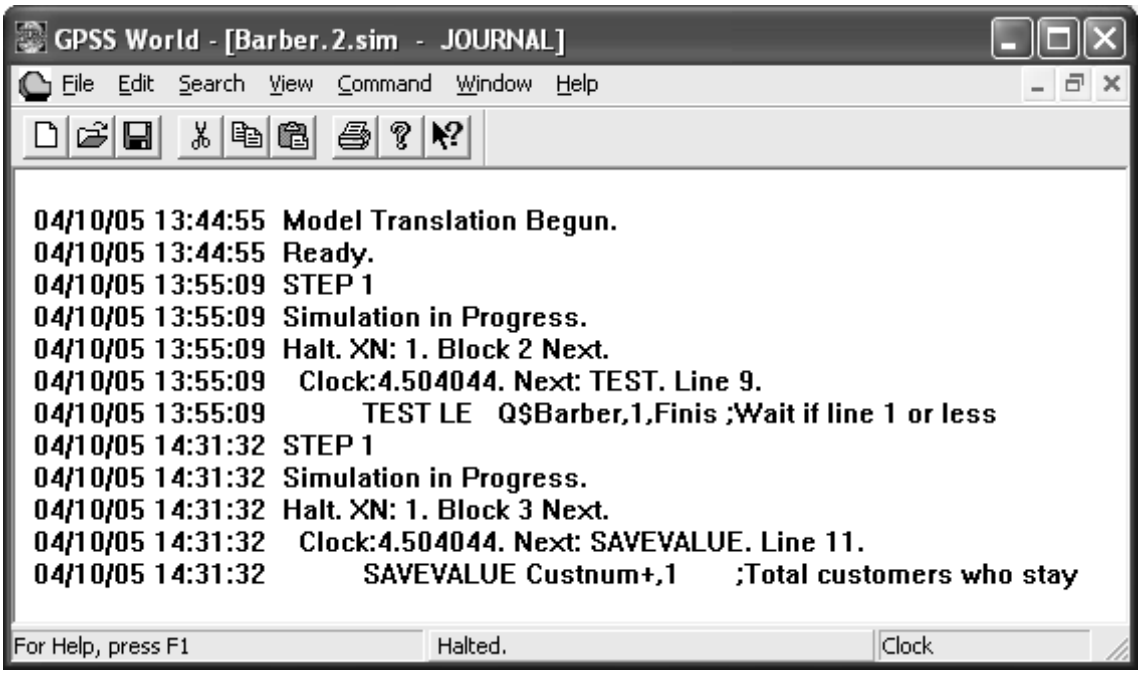


Рисунок 4.4 — Фрагмент *Журнала* после второго шага при пошаговой работе с помощью клавиши **F5**

Характерно то, что в самой последней строке приводится блок, который будет выполнен на следующем шаге, чтобы исследователь знал ход работы программы (последний выполненный оператор располагается в предпоследней строке).

При следующем нажатии клавиши пошагового моделирования **F5** журнал записывает аналогичную информацию.

Очевидно, что при содержательном оформлении комментариев можно получить связный рассказ о процессе в модели.

4.3 Стандартный отчёт Standard Report

Standard Report — *Стандартный отчёт* является тем документом, в котором накапливаются основные результаты моделирования. Он позволяет получить полное представление о том, чем завершилось моделирование, содержит максимальные и средние значения различных переменных и др. Рассмотренные ниже *Окна моделирования* позволяют детализировать информацию, имеющуюся в *Стандартном отчёте* и визуализировать её, но информации в *Стандартном отчёте* обычно самой по себе достаточно, чтобы делать выводы об основных результатах моделирования.

Вид *Стандартного отчёта* можно настраивать с помощью специального окна настройки *Стандартного отчёта*: **Edit → Settings → Report**. Тогда открывается окно настройки *Стандартного отчёта* (Рисунок 4.5).

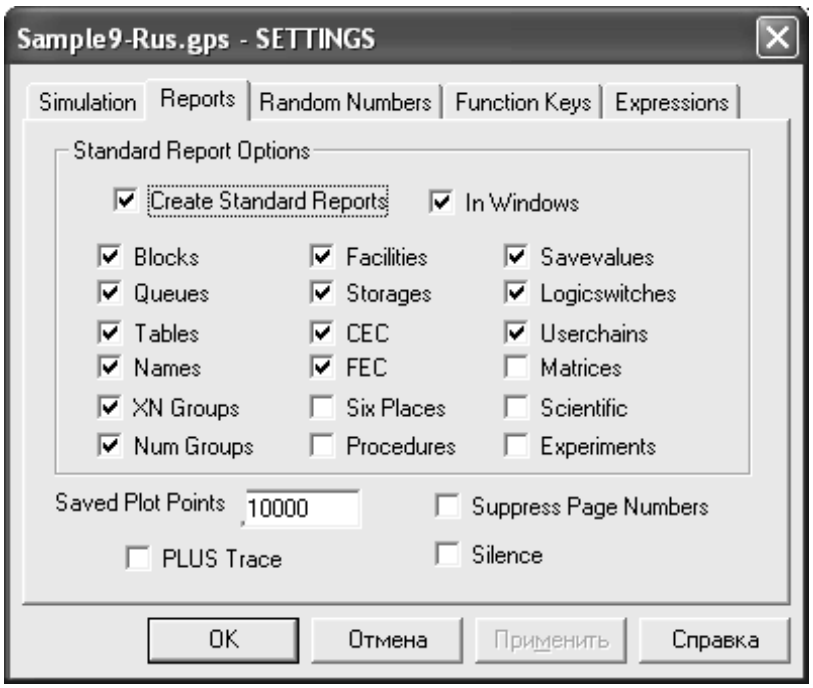


Рисунок 4.5 — Диалоговое окно настройки
Стандартного отчёта ***Standard Report***

Настраиваемые параметры на вкладке (Рисунок 4.5) имеют следующий смысл:

Blocks — блоки (вывод имён блоков и соответствующей им статистики);

Queues — очереди (вывод имён *Очередей* и соответствующей им статистики — очереди есть не во всякой программе и выводить их не всегда надо);

Tables — таблицы (вывод имён *Таблиц* и соответствующей им статистики — *Таблицы* используются для накапливания данных в процессе моделирования и построения в последующем гистограмм, они не всегда присутствуют в программе и поэтому загромождать ими *Стандартный отчёт* при их отсутствии не нужно);

Names — имена (вывод имён всех переменных программы и соответствующих им значений);

XN Groups — **XN**-группы (при наличии *Групп транзактов* можно вывести значения отдельных транзактов из этих групп);

Num Groups — числовые группы (при наличии *Числовых групп* можно вывести имена этих групп и соответствующую им статистику);

Facilities — обслуживающие устройства (*Обслуживающие устройства* есть в любой реальной программе);

Storages — многоканальные устройства (*Многоканальные устройства* — есть не во всех программах);

CEC (Current Events Chain) — цепь текущих событий (содержит информацию о транзактах, помещённых в *Цепь текущих событий*);

FEC (Future Events Chain) — цепь будущих событий (содержит информацию о транзактах, помещённых в *Цепь будущих событий*);

Six Places — шесть знакомест (точность представления чисел). При отсутствии флажка точность — три знака.

Procedures — процедуры (информация об использованных процедурах, вводимых программистом);

Savevalues — сохраняемые величины (*Сохраняемые величины* — обычные переменные, которые записываются в память и могут использоваться затем в программе; они отличаются от других объектов GPSS — блоков, *Обслуживающих устройств*, *Многоканальных устройств*, *Очередей*, которые не являются переменными, а некими объектами, содержащими много параметров; *Сохраняемые величины* имеются не во всех программах, в случае их наличия выводится таблица с перечнем *Сохраняемых величин* и их значениями на момент конца моделирования);

Logicswitches — логические ключи (приводятся данные о *Логических ключах*, которые есть не во всех программах; при их наличии выводится список *Логических ключей* и их значения (0 – выключен или 1 – включен) на момент окончания моделирования);

Userchain — цепь пользователя (приводится информация о наличии транзактов в *Цепи пользователя*, в которую транзакты помещаются принудительно при выполнении определённых условий, заданных программистом);

Matrices — матрицы (приводится информация о *Матрицах*, которые есть не во всех программах);

Scientific — научный (представление чисел с мантиссой в виде десятичной экспоненты);

Experiments — эксперименты (выводится информация об экспериментах).

Дополнительные параметры вверху:

Create Standard Window — создавать или нет *Стандартный отчёт*;

In Window — выводить или нет *Стандартный отчёт* на экран. Создание *Стандартного отчёта* предусматривает его обязательную запись в файл, но вывод на экран при этом не обязателен. Если создание *Стандартного отчёта* отменено, то вывод на экран также невозможен.

Дополнительные параметры внизу:

Saved Plot Points — число сохраняемых точек для построения графика;

Suppress Page Numbers — простановка номеров страниц в *Стандартном отчёте* вверху;

PLUS Trace — осуществлять или нет трассировку программ на языке PLUS;

Silence — выключение звукового сигнала, сообщающего об ошибке при работе программы.

Внешний вид *Стандартного отчёта* приведён на серии рисунков (Рисунок 4.6 — Рисунок 4.9). *Стандартный отчёт* обычно имеет сравнительно большие размеры и не всегда вмещается в окно экрана.

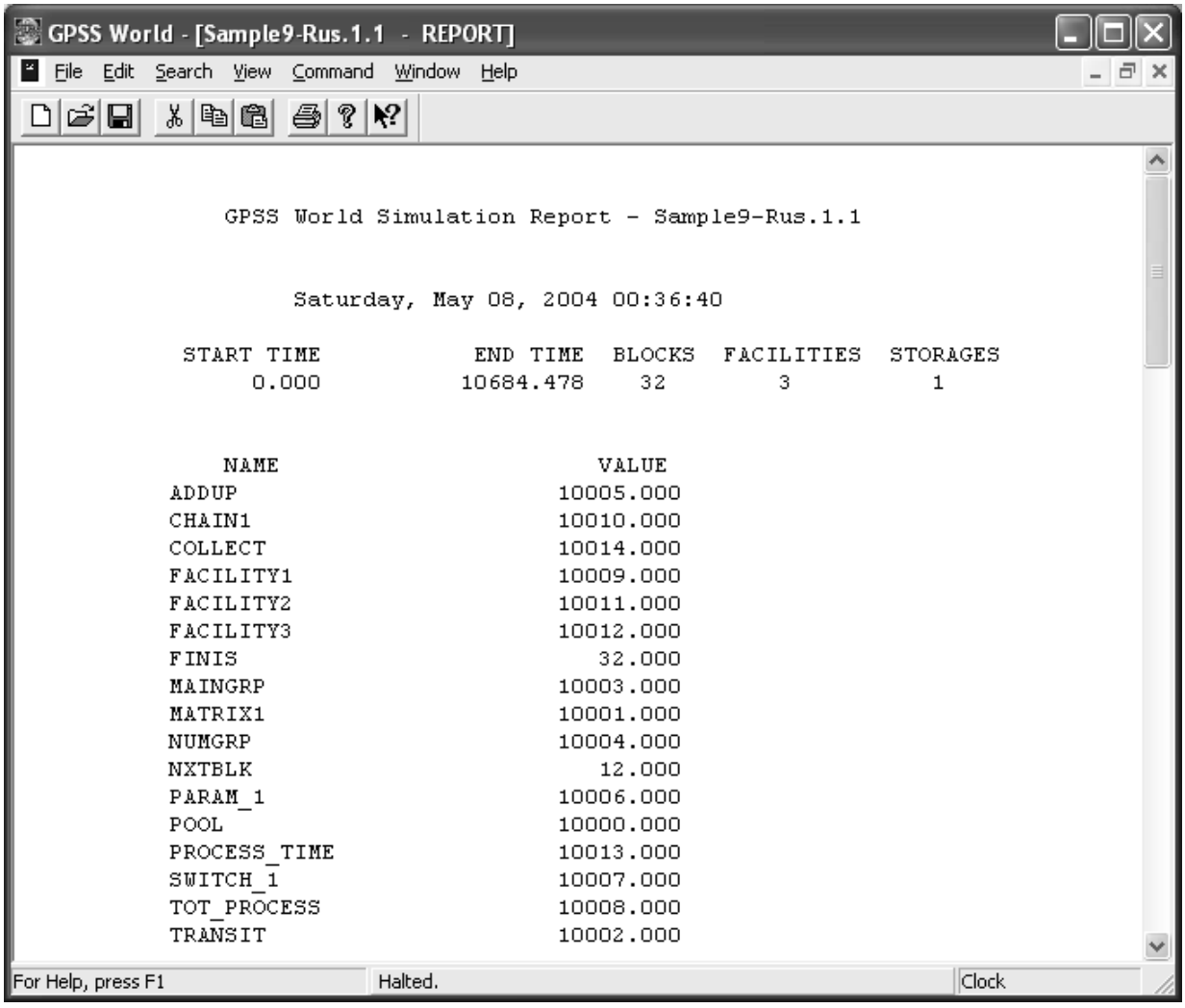


Рисунок 4.6 — *Стандартный отчёт*. Начало. Часть 1

В первой строке начальной части *Стандартного отчёта* (Рисунок 4.6) приводится название использованной программы (**GPSS World**), окна (**Simulation Report**) и программной модели (**Sample9-Rus.1.1**). В последнем случае имеется ввиду программа **Sample9-Rus.gps**.

Во второй строке — дата и время моделирования (**Saturday, May 08, 2004 00:36:40**).

Ниже приводятся таблицы с различными данными.

Первой идёт таблица с наиболее общими данными о программе и закончившемся процессе моделирования:

START TIME —время начала сеанса моделирования (в данном случае, как это часто и бывает, оно равно 0, но в случае нескольких последовательных сеансов моделирования может быть и другим);

END TIME — время окончания сеанса моделирования (в данном случае 10684,478 единиц внутреннего модельного времени — физический смысл его для каждой программы — свой);

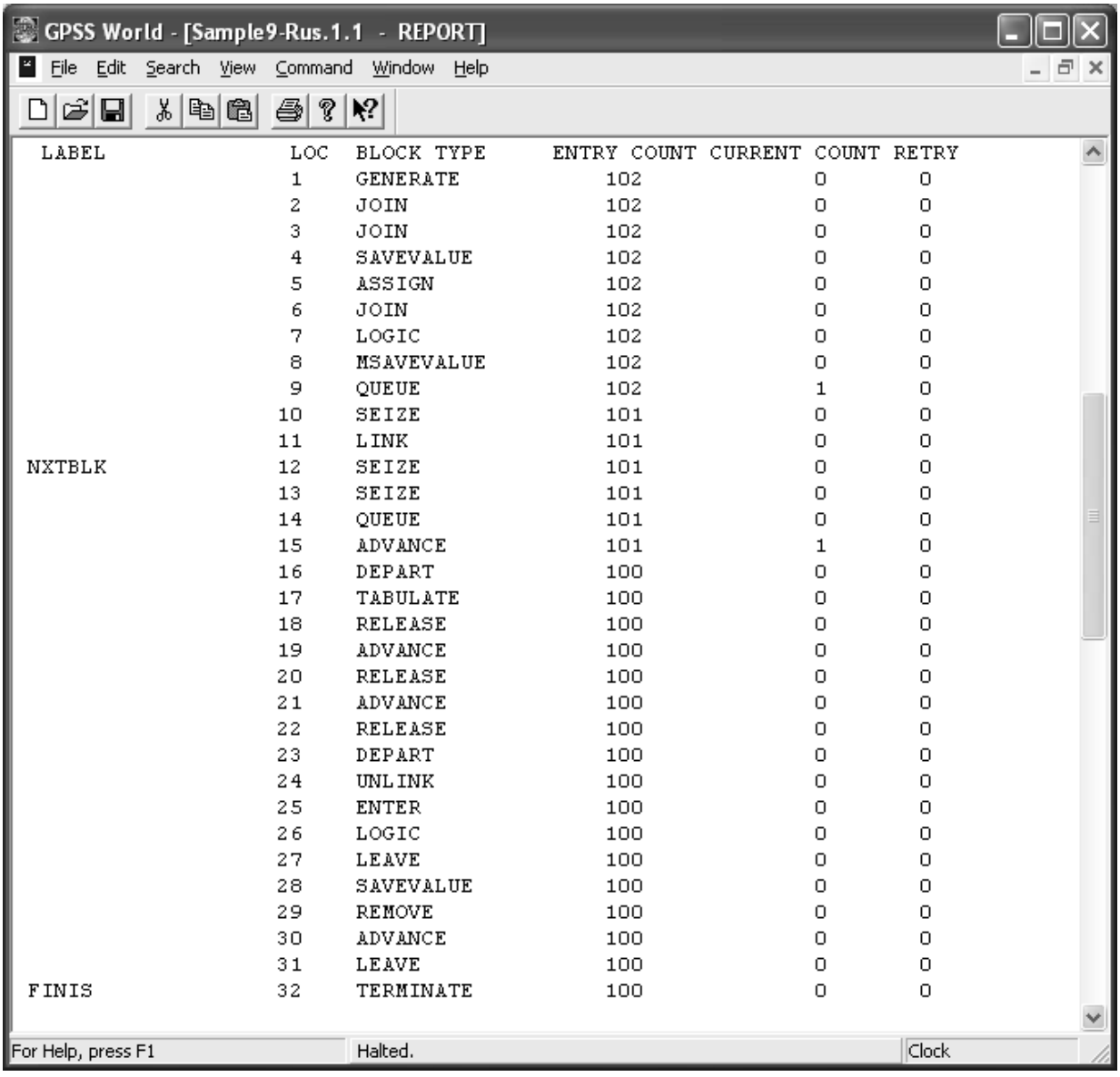
BLOCKS — количество блоков в программе (в данном случае их 32);

FACILITIES — количество *Обслуживающих устройств* в программе (в данном случае их 3);

STORAGES — количество *Многоканальных устройств* (в данном случае их 1).

Второй расположена таблица имён (блоков, *Обслуживающих устройств*, *Многоканальных устройств*, *Очередей*) с их значениями на момент окончания моделирования.

Во второй части окна *Стандартного отчёта* (Рисунок 4.7) приведена таблица, отражающая структуру программы.



LABEL	LOC	BLOCK TYPE	ENTRY COUNT	CURRENT COUNT	RETRY
	1	GENERATE	102	0	0
	2	JOIN	102	0	0
	3	JOIN	102	0	0
	4	SAVEVALUE	102	0	0
	5	ASSIGN	102	0	0
	6	JOIN	102	0	0
	7	LOGIC	102	0	0
	8	MSAVEVALUE	102	0	0
	9	QUEUE	102	1	0
	10	SEIZE	101	0	0
	11	LINK	101	0	0
NXTBLK	12	SEIZE	101	0	0
	13	SEIZE	101	0	0
	14	QUEUE	101	0	0
	15	ADVANCE	101	1	0
	16	DEPART	100	0	0
	17	TABULATE	100	0	0
	18	RELEASE	100	0	0
	19	ADVANCE	100	0	0
	20	RELEASE	100	0	0
	21	ADVANCE	100	0	0
	22	RELEASE	100	0	0
	23	DEPART	100	0	0
	24	UNLINK	100	0	0
	25	ENTER	100	0	0
	26	LOGIC	100	0	0
	27	LEAVE	100	0	0
	28	SAVEVALUE	100	0	0
	29	REMOVE	100	0	0
	30	ADVANCE	100	0	0
	31	LEAVE	100	0	0
FINIS	32	TERMINATE	100	0	0

Рисунок 4.7 — *Стандартный отчёт*. Продолжение. Часть 2

При этом указываются метки (**LABEL**; в данном случае их две — **NXTBLK** и **FINIS**), номера блоков (**LOC**; нумерация их выполняется автоматически транслятором GPSS — на эти номера можно ссылаться в программе: в данном случае их 32), имена всех блоков (**BLOCK TYPE**), количество вошедших в эти блоки транзактов (**ENTRY COUNT**), количество имеющихся в данный момент в блоках транзактов (**CURRENT COUNT**), повторные входы транзактов в каждый блок (**RETRY**).

В третьей части окна *Стандартного отчёта* (Рисунок 4.8) имеется 4 таблицы:

- для *Обслуживающих устройств* (**FACILITY**, в программе было 3 *Обслуживающих устройства* с именами **FACILITY1**, **FACILITY2** и **FACILITY3**),
- для *Очередей* (**QUEUE,** в программе было 2 очереди с именами **TOT_PROCESS** и **PROCESS_TIME**),
- для *Многоканальных устройств* (**STORAGE**, в программе было одно *Многоканальное устройство* с именем **POOL**),
- для *Таблиц* (**TABLE**, в программе была одна *Таблица* с именем **TRANSIT**).

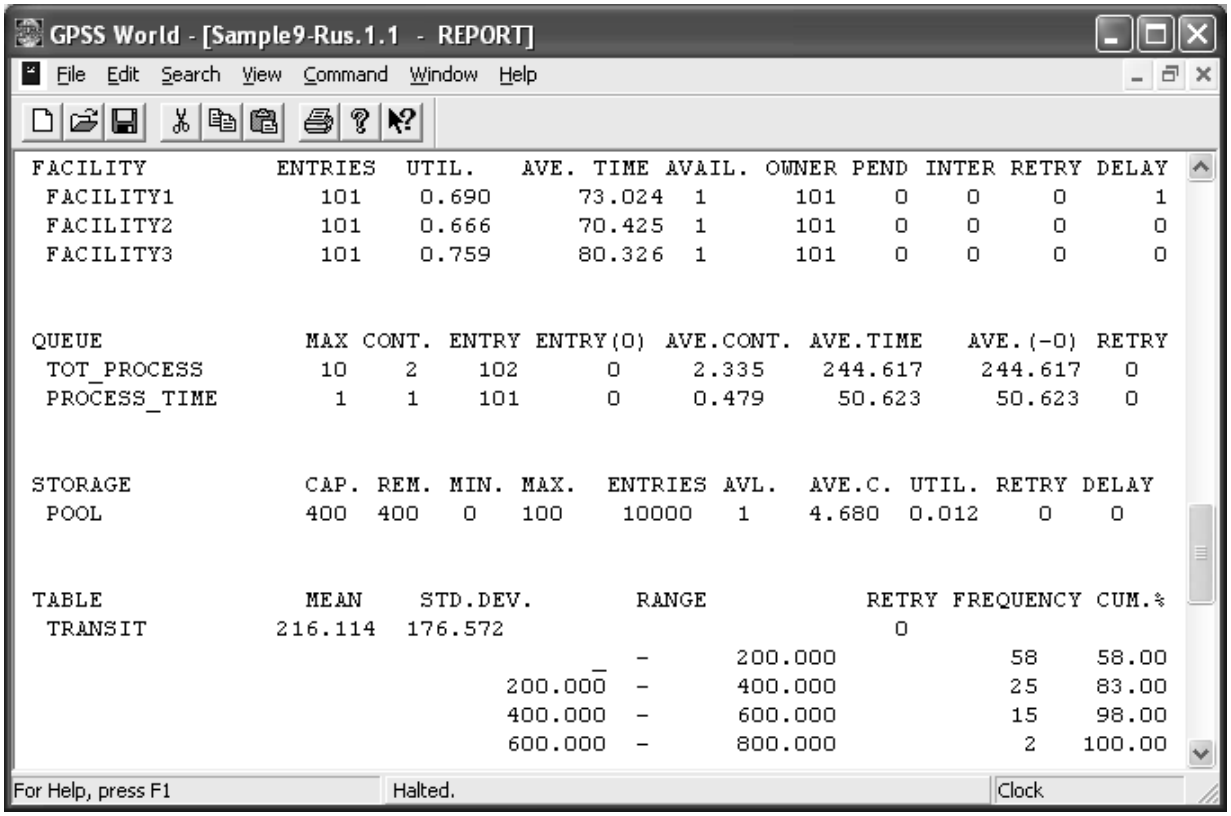


Рисунок 4.8 — *Стандартный отчёт*. Продолжение. Часть 3

Для каждого типа устройств имеется свой набор выводимых параметров по столбцам.

Для *Обслуживающих устройств* ***Facility*** это:

ENTRIES — число входов транзактов в *Обслуживающее устройство*;

UTIL. — коэффициент использования, равный отношению времени работы *Обслуживающего устройства* (при обслуживании транзактов) к общему времени моделирования (что-то вроде коэффициента полезного действия);

AVE. TIME [average time] — среднее время обслуживания транзактов;

AVAIL. [available] — доступность устройства для транзактов на момент вывода *Стандартного отчёта*;

OWNER — номер транзакта, находящегося в *Обслуживающем устройстве*;

PEND — состояние Цепи

INTER — состояние Цепи

RETRY — состояние Цепи

DELAY — состояние Цепи

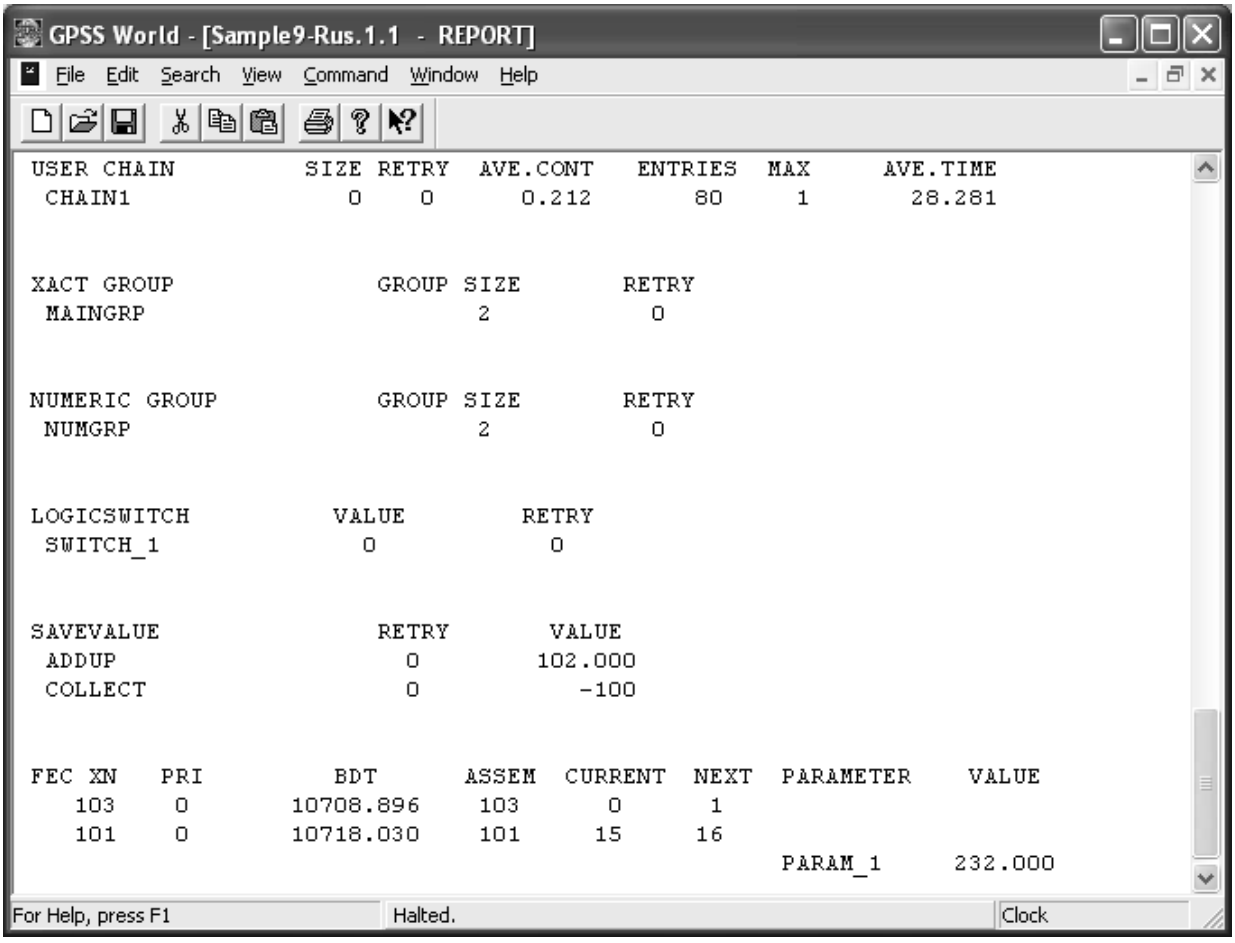


Рисунок 4.9 — *Стандартный отчёт*. Окончание. Часть 4

Стандартный отчёт можно не формировать или формировать и писать в файл или выводить на экран. Он содержит много полезной информации, часто достаточной для оценки результатов моделирования. Если этой информации недостаточно, то используют *Окна* GPSS World.

4.4 Окна GPSS World

4.4.1 Обзор Окон GPSS World

В GPSS World имеются встроенные средства визуализации значений переменных и процессов, которые называются *Окна*. Каждое окно представляет собой окно в смысле Windows со специальным набором отображаемой информации. *Окна* можно вывести на экран перед началом сеанса моделирования и отслеживать в них все изменения переменных внутри модели.

В GPSS имеются следующие окна:

Blocks Window *— Окно блоков*.
Expression Window *— Окно выражений*.
Facilities Window *— Окно обслуживающих устройств*.
Logicswitches Window *— Окно логических ключей*.
Matrix Window *— Окно матриц*.
Plot Window *— Окно графиков*.
Queues Window *— Окно очередей*.
Savevalues Window *— Окно сохраняемых величин*.
Storages Window *— Окно многоканальных устройств*.
Table Window *— Окно гистограмм*.

Для работы с этими окнами необходимо после трансляции программы (**Command → Create Simulation**) использовать команды меню **Window → Simulation Window**, чтобы попасть на вкладку с выше перечисленными названиями окон и вызвать необходимое окно. Без оттранслированной программы соответствующие команды меню неактивны.

Некоторые из окон вызываются не всегда, а только если соответствующая им информация имеется в программе (*Окно матриц* ***Matrix Window*** — если есть матрицы, *Окно гистограмм* ***Table Window*** — если для гистограмм собирается соответствующая информация с помощью команды **`TABULATE`**).

Окна позволяют наблюдать изменения значений различных переменных в цифровой, табличной, графической форме в процессе работы программы. Можно задать непрерывную работу (командой **`START`**), а можно — работу по шагам (функциональной клавишей **F5**). При пошаговой работе использование окон наиболее эффективно.

Окна можно расположить различным образом — одно окно на весь экран или рядом. В последнем случае удобно отслеживать состояния различных блоков и работу программы.

4.4.2 Окно блоков Blocks Window

Blocks Window (*Окно блоков*) визуально показывает перемещение транзактов через блоки с отражением мгновенной информации.

Возможны два представления *Окна блоков* — в развёрнутом виде (Рисунок 4.10) и в компактном виде (Рисунок 4.11). Они отличаются наличием (в развёрнутом виде) или отсутствием (в компактном виде) статистических данных по отдельным блокам. В первом случае при моделировании сразу видны количественные параметры блоков (число транзактов в *Очередях* и др.), но при этом размер окна не всегда вмещает все блоки большой программы. В компактном виде появляется возможность более наглядно представить даже сравнительно большие программы. Переключение между развёрнутым и компактным видами осуществляется с помощью команд меню ***View → Entities Details***. Если пометить ***Entities Details*** галочкой, то *Окно блоков* будет представлено в развёрнутом виде, при отсутствии галочки — в компактном.

Окно блоков в компактном виде представляет собой список блоков программы в той последовательности, в какой они присутствуют в программе. Блоки помечены условными графическими обозначениями (принятыми в GPSS World), номерами (по порядку размещения в программе) и сокращёнными наименованиями. При запуске программы слева от блоков появляются условные значки в виде прямоугольников, означающие появление транзактов, их перемещение, появление очередей, переполнение очередей и т. п.

Окно блоков в развёрнутом виде представляет собой таблицу с числом строк, равным числу блоков в программе и числом столбцов 7.

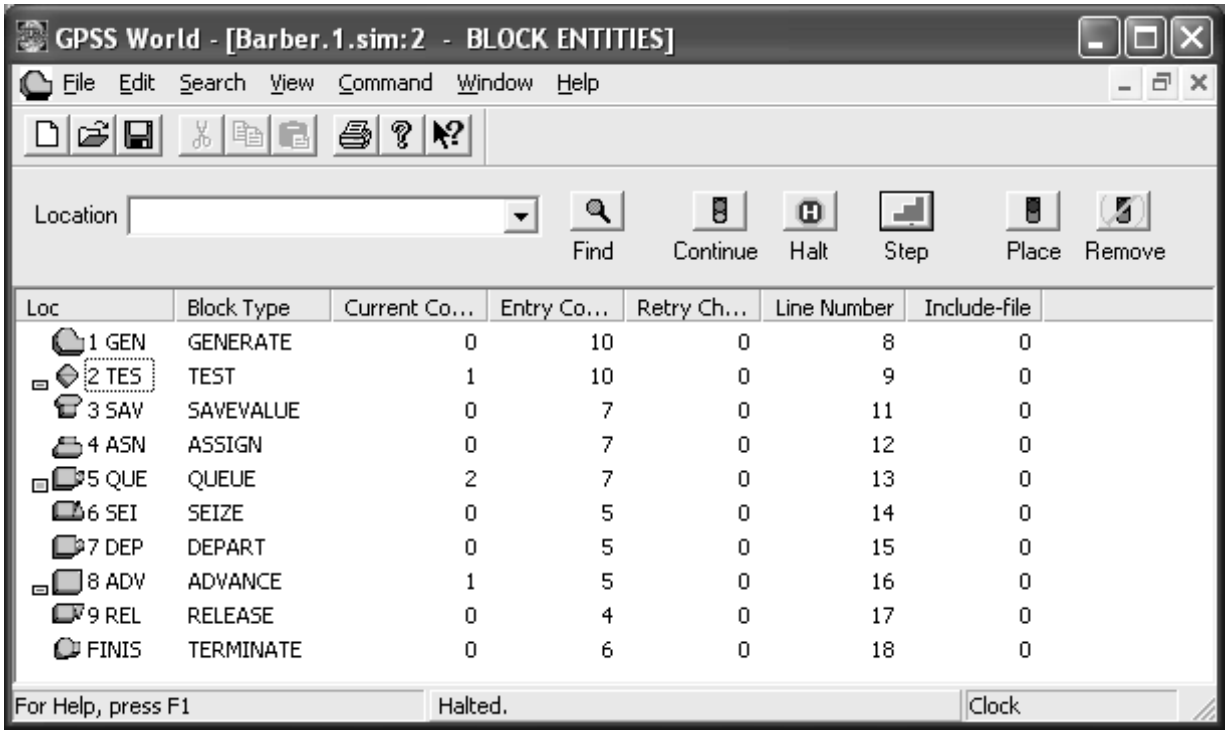


Рисунок 4.10 — *Окно блоков* ***Blocks Window*** в развёрнутом виде

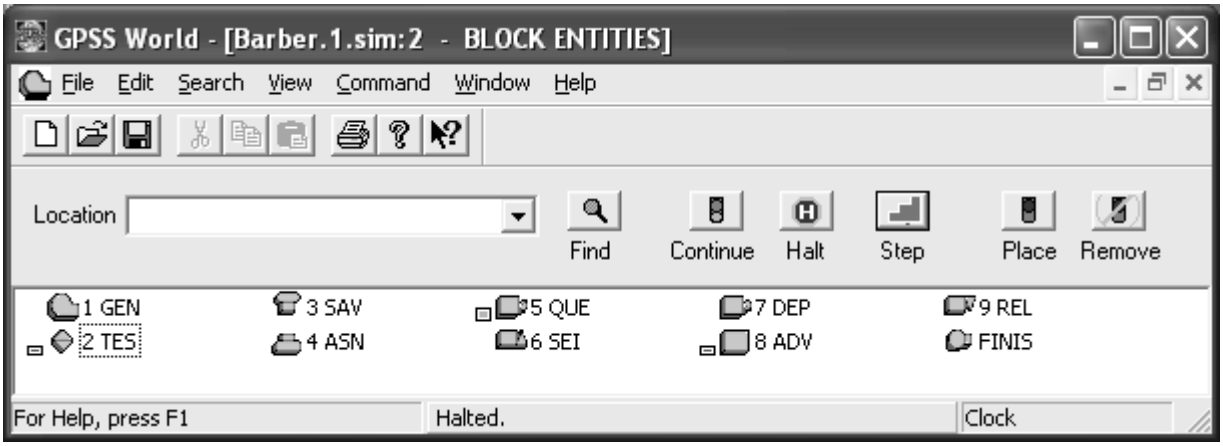


Рисунок 4.11 — *Окно блоков* ***Blocks Window*** в компактном виде

Информация в колонках *Окна блоков* ***Blocks Window*** имеет следующий смысл:

`Loc` — условное графическое обозначение, порядковый номер и краткое наименование блока в программе;

`Block Type` — полное наименование блока;

`Current Count` — текущее число транзактов в данном блоке;

`Entry Count` — число состоявшихся входов транзактов в данный блок;

`Retry Chain` — число повторных попыток входа транзактов в данный блок (когда с первой попытки вход был невозможен);

`Line Number` — номер строки в листинге (тексте) программы;

`Include File` — *Включённый файл* (если он есть).

Работа с *Окном блоков* ***Blocks Window*** может происходить как в непрерывном (автоматическом) режиме моделирования, так и в шаговом режиме. Выбор режима осуществляется с помощью кнопок на верхней панели окна, имеющих следующий смысл:

кнопка 

— вызывает продолжение автоматического моделирования, если оно было ранее прервано кнопкой **Halt**;

кнопка 

— останавливает автоматическое моделирование;

кнопка 

— делает 1 шаг в шаговом режиме с переходом к ближайшему следующему событию.

Для работы в автоматическом режиме (с возможностью останова кнопкой **Halt** и продолжения работы кнопкой **Continue**) необходимо в конце программы задать команду **`START`** или сделать старт после трансляции программы через меню **Command** → **Start** с указанием в появившемся окне **Start Command** числа циклов *Счётчика завершения*, например (Рисунок 4.12):

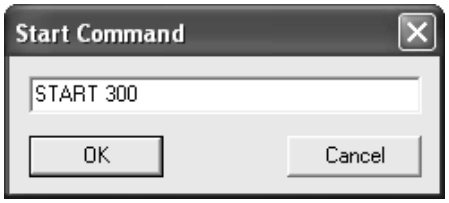


Рисунок 4.12 — Ввод числа циклов в *Счётчик завершения* для автоматического режима моделирования

Затем в *Окне блоков* следует манипулировать кнопками **Halt**, **Continue** и **Step**. При нажатии кнопки **Halt** моделирование прервётся и программа перейдёт в режим ожидания. При нажатии кнопки **Continue** моделирование продолжится. При нажатии кнопки **Step** произойдёт продвижение программы на 1 шаг — к очередному событию.

Если программа в автоматическом режиме закончила работу, то последовательным нажатием кнопки **Step** можно продолжить её работу каждый раз на 1 шаг.

В непрерывном режиме видно, как в динамике генерируются транзакты, разбегаются по блокам, исчезают из программы. В шаговом режиме можно отслеживать отдельные транзакты в отельные моменты времени в отдельных блоках.

4.4.3 Окно выражений Expression Window

Expression Window — *Окно выражений* (Рисунок 4.13). Показывает значения параметров модели в процессе моделирования, которые можно задать с помощью *Выражений*. В качестве *Выражений* используются *Системные числовые атрибуты* (СЧА) совместно с переменными, имеющими имя: например, **Q$***Имя* — означает длину очереди с именем этой очереди, определённым в программе. Например, **Q$Barber** означает, что в программе имеется очередь, названная программистом **Barber**. Часть выражения — **Q$** (с долларом справа) является *Системным числовым атрибутом* с определённой функцией (раздел 3.2), в данном случае — с функцией определения размера очереди с именем **Barber**.

Для шести конкретных типов блоков (*Очередей, Обслуживающих устройств, Логических ключей, Матриц, Многоканальных устройств, Сохраняемых величин*) существуют специальные окна (соответственно, ***Queue Window***, ***Facilities Window***, ***Logicswitches Window***, ***Matrix Window***, ***Storages Window***, ***Savevalues Window***), в которых по умолчанию выводится большое число параметров, характерных для этих блоков. Но может потребоваться вывод и других параметров, тогда следует воспользоваться окном ***Expression Window***. Для других блоков специальных окон нет и тогда окно ***Expression Window*** является единственным, где можно посмотреть их состояние. Кроме простейших СЧА, можно использовать арифметические выражения с ними,

как с обычными переменными, и получать таким образом более гибкую систему контроля параметров протекающих процессов.

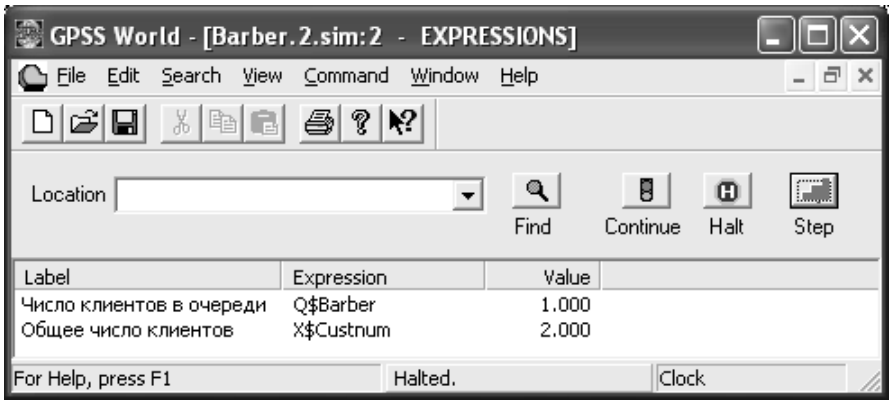


Рисунок 4.13 — *Окно выражений* ***Expression Window***

Первоначальная настройка *Окна выражений* ***Expression Window*** осуществляется с помощью специального окна редактирования (Рисунок 4.14), вызываемого через меню: **Window → Simulation Window → Expression Window** после транслирования программы.

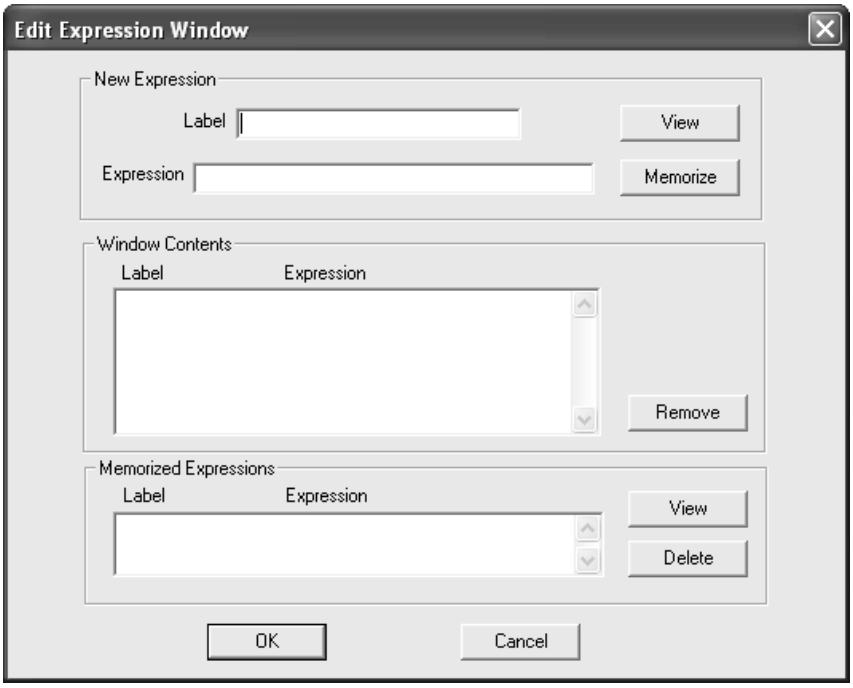


Рисунок 4.14 — Диалоговое окно *Окна выражений* ***Expression Window***

Чтобы настроить *Окно выражений* ***Expression Window***, используются следующие параметры.

В поле Label помещается поясняющее название выводимой переменной (произвольное, на любом языке), которое затем будет выводиться в *Окне выражений*: в данном случае — "Число клиентов в очереди" (Рисунок 4.15).

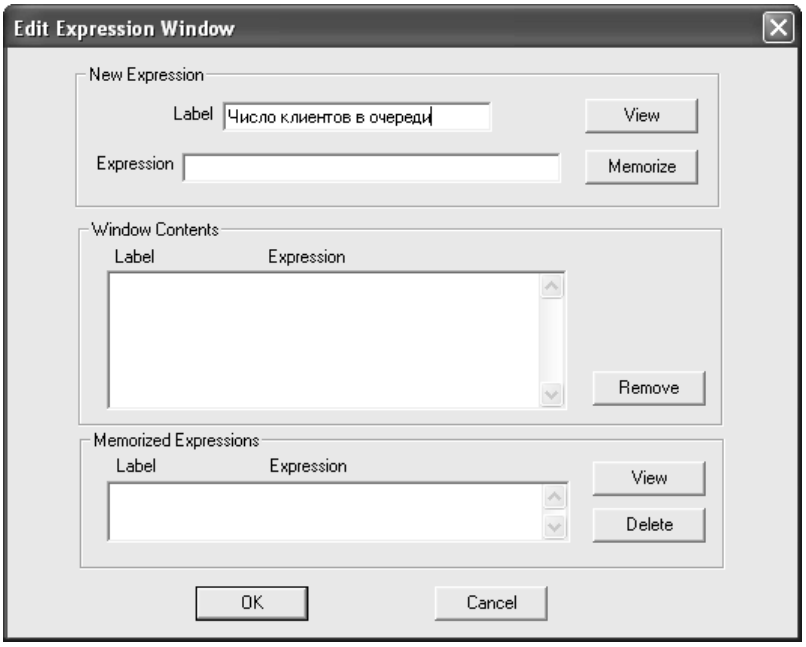


Рисунок 4.15 — Диалоговое окно *Окна выражений* ***Expression Window.*** Задание поясняющей надписи в поле ***Label***

В поле ***Expression*** записывается само выражение с использованием СЧА: в данном случае — **Q$Barber** (Рисунок 4.16).

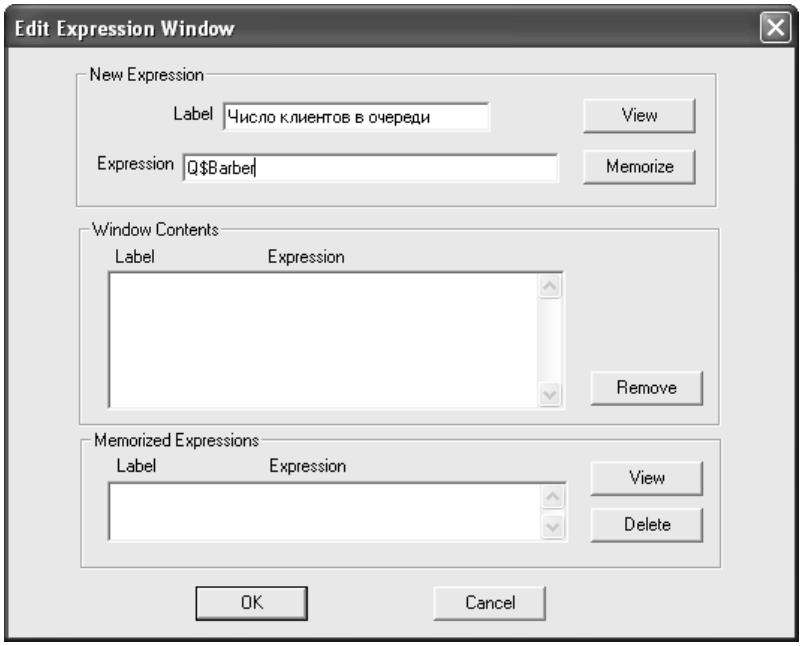


Рисунок 4.16 — Диалоговое окно *Окна выражений* ***Expression Window***. Задание выражения в поле Expression

Затем нажимается кнопка **View** и обе введённые записи переписываются вниз в поле Window Contents (Рисунок 4.17). Команда **View** означает, что записанные выражения (точнее, получаемые через них значения переменных) будут затем выводиться в *Окне выражений*.

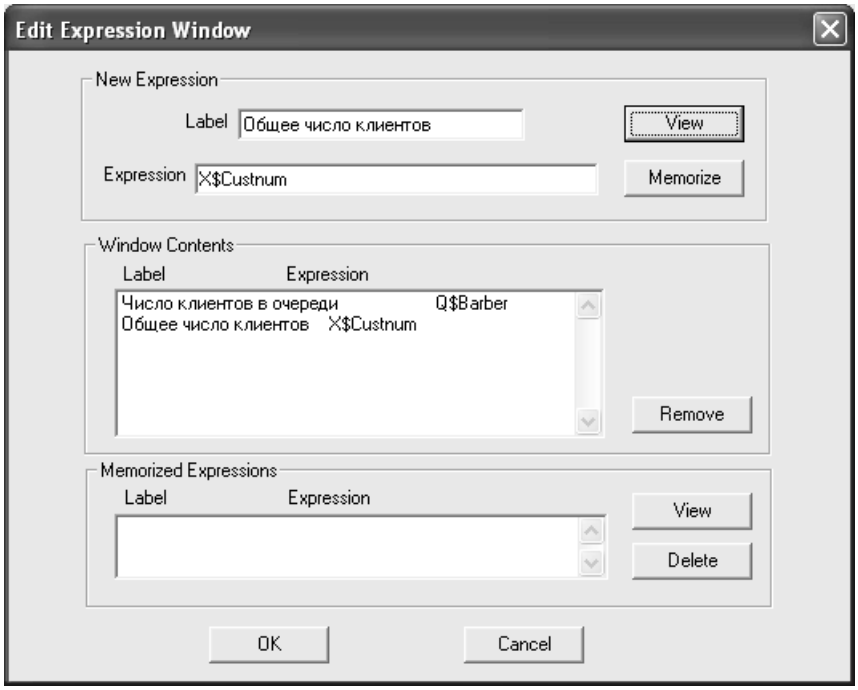


Рисунок 4.17 — Диалоговое окно *Окна выражений* ***Expression Window***

Затем в случае необходимости можно ввести также другие выражения и таким образом составить список выводимых выражений.

Каждое из выражений при вводе можно запомнить, если нажать на кнопку **Memorize** и оно будет показано в строках Memorized Expressions (Рисунок 4.18).

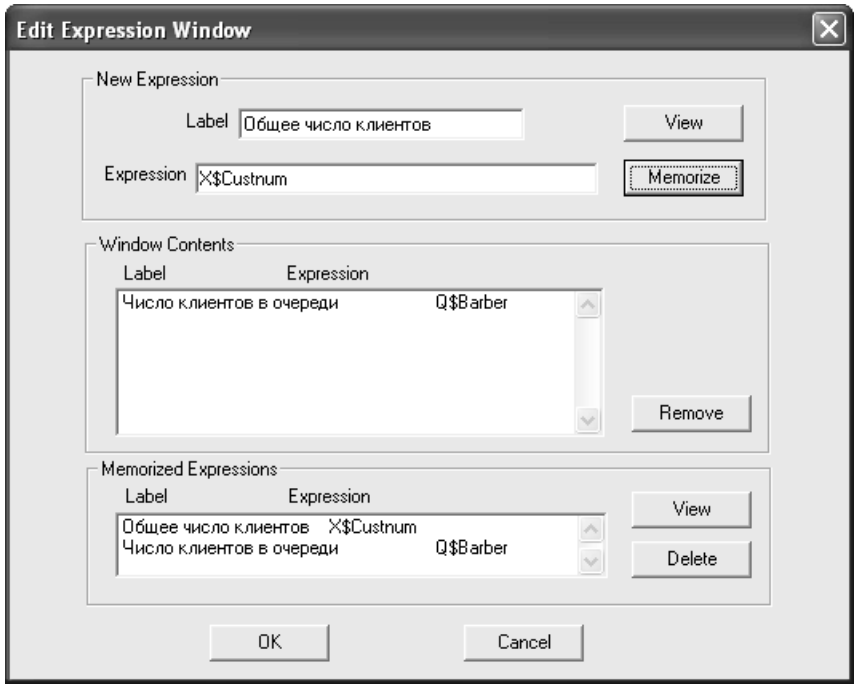


Рисунок 4.18 — Диалоговое окно *Окна выражений* ***Expression Window*** с запоминанием выводимых выражений

Если выражения уже были записаны в строках Window Contents, то нужное выражение высвечивают курсором, а затем также нажимают кнопку **Memorize**. При этом можно указать для вывода не все выражения, которые указаны для запоминания. Запомненные выражения сохраняются для того, чтобы их использовать в следующем сеансе моделирования.

В процессе моделирования в паузах между сеансами можно изменять (дополнять, очищать) *Окно выражений* ***Expression Window***. Для этого используют меню **Edit → Expression Window**, после чего появится окно (Рисунок 4.19).

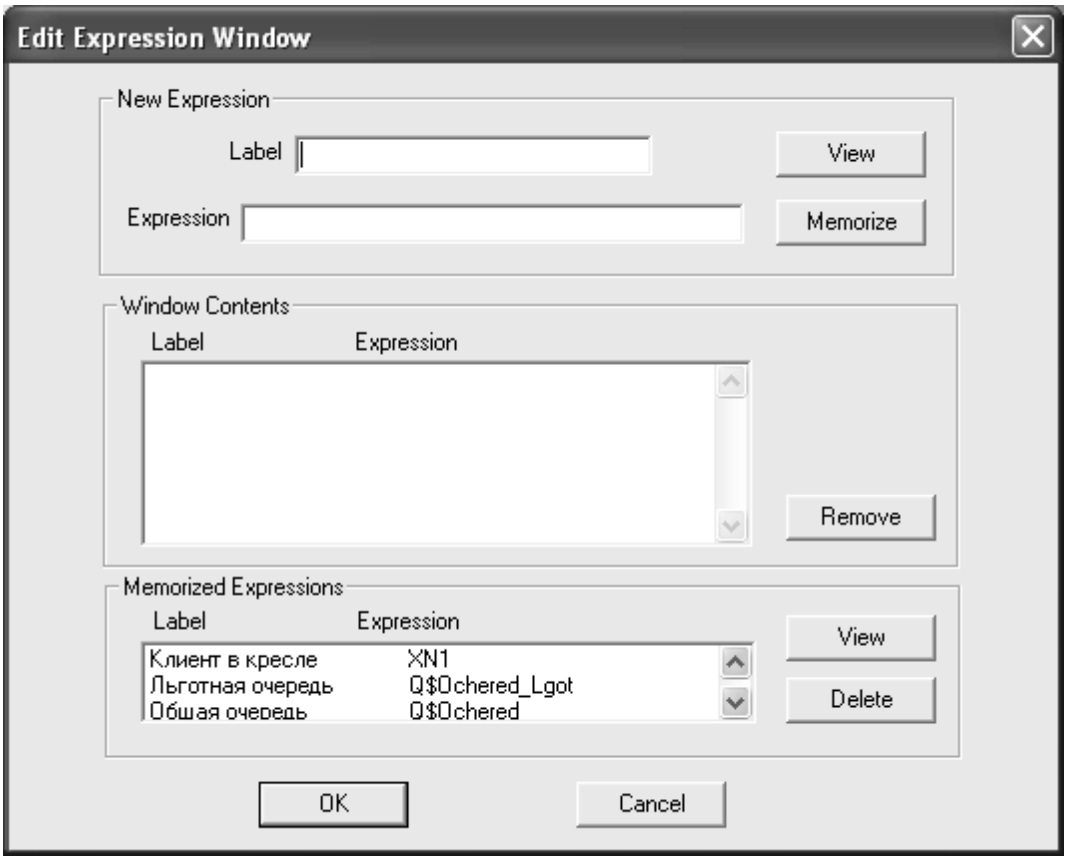


Рисунок 4.19 — Диалоговое окно для редактирования *Окна выражений* ***Expression Window***

Окно (Рисунок 4.19) повторяет окно настройки *Окна выражений* ***Expression Window*** (Рисунок 4.14), и с его помощью можно изменять эти настройки, а именно, убирать ненужные для вывода выражения и вводить нужные.

Для свободного пользования *Окном выражений* ***Expression Window*** необходимо хорошо знать систему *Системных числовых атрибутов* GPSS World (раздел 3.2).

4.4.4 Окно обслуживающих устройств Facilities Window

Facilities Window — *Окно обслуживающих устройств*. *Обслуживающие устройства* — это устройства, в которых выполняется обслуживание (обработка) транзактов. Все транзакты обслуживаются в этих устройствах в течение некоторого времени (но время задаётся не в самих *Обслуживающих устройствах*, а в блоке **ADVANCE**). Можно получить информацию о текущих, максимальных, средних временах обслуживания транзактов. *Окно обслуживающих устройств* ***Facilities Window*** может быть представлено в двух видах — развёрнутом (Рисунок 4.20) и компактном (Рисунок 4.21).

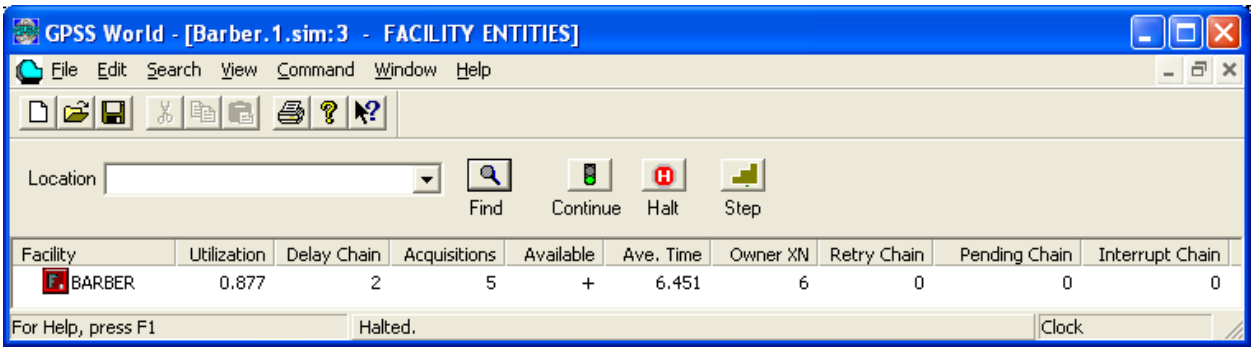


Рисунок 4.20 — *Окно обслуживающих устройств* ***Facilities Window*** в развёрнутой форме

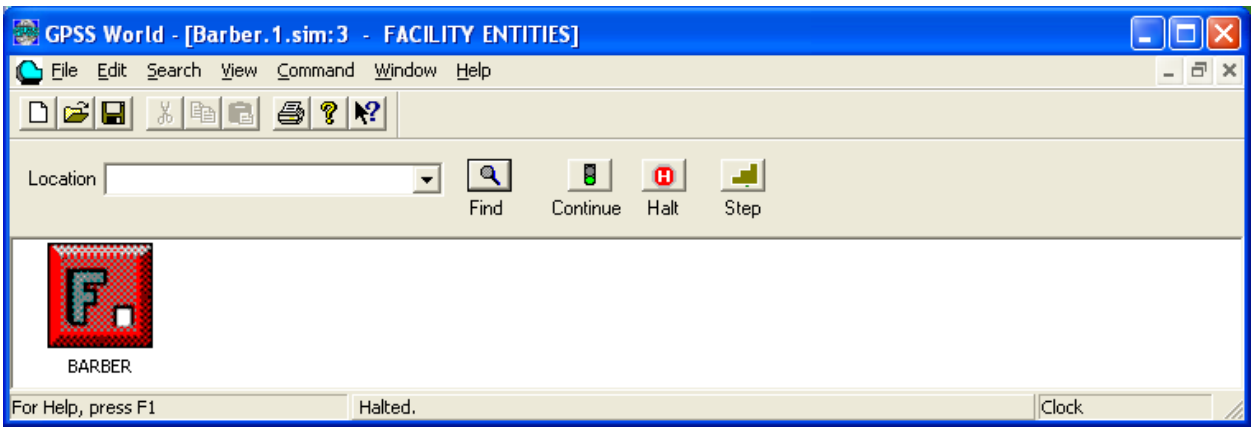


Рисунок 4.21 — *Окно обслуживающих устройств* ***Facilities Window*** в компактном виде

Переход от развёрнутого вида к компактному и наоборот осуществляется, как и в случае с *Окном блоков* ***Blocks Window***, с помощью меню **View → Entities Details**.

Компактный вид (Рисунок 4.21) не представляет существенного интереса и необходим только для того, чтобы просмотреть весь набор *Обслуживающих устройств*. В основном используется развёрнутый вид. В этом случае окно содержит таблицу с 10 столбцами и числом строк, равным числу *Обслуживающих устройств*.

Столбцы *Окна обслуживающих устройств* ***Facilities Window*** включают следующую информацию:

Facility — имя *Обслуживающего устройства*, данное ему в программе (в примере — имя **BARBER**);

Utilization — коэффициент использования, равный отношению суммарного времени обслуживания к общему времени моделирования (характеризует эффективность работы *Обслуживающего устройства* и может изменяться в диапазоне 0,000…1,000);

Delay Chain — число транзактов, находящихся в *Цепи задержки*;

Acquisitions — число входов транзактов в *Обслуживающее устройство*;

Available — доступность *Обслуживающего устройства* (+ доступно, – недоступно), доступность можно задавать принудительно, имитируя, например, поломку или перерыв в работе;

Ave. Time (average time) — среднее время обслуживания;

Owner XN — номер обслуживаемого в данном устройстве в данный момент транзакта;

Retry Chain — число транзактов в *Цепи повторений*;

Pending Chain — число транзактов в *Цепи продолжения*;

Interrupt Chain — число транзактов в *Цепи прерываний*.

Таким образом, *Окно обслуживающих устройств* ***Facilities Window*** в основном даёт информацию о нахождении транзактов в различных *Цепях* модели (см. соответствующий раздел этого учебного пособия), имеющих отношение к *Обслуживающим устройствам*.

4.4.5 Окно логических ключей Logicswitches Window

Logicswitches Window — *Окно логических ключей*. *Логические ключи* — специфические переменные GPSS, задаваемые специальным оператором **LOGIC**. Они бывают необходимы, чтобы обеспечить задание и учёт условий для условных операций. Транзакты могут переключать логические ключи в одно из двух состояний: 0 или 1. В любом другом месте программы можно проанализировать состояние того или иного ключа и в зависимости от него выполнить то или иное действие. Например, может фиксироваться проход транзакта через какое-то *Обслуживающее устройство* в середине программы и на основании этого приниматься решение в другом месте программы.

Окно логических ключей может быть как в развёрнутом (Рисунок 4.22), так и в компактном (Рисунок 4.23) виде.

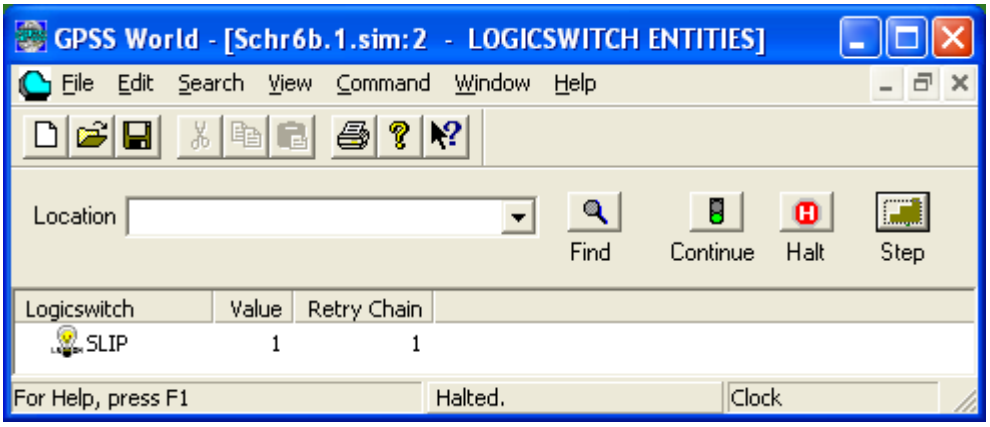


Рисунок 4.22 — *Окно логических ключей* ***Logicswitches Window*** в развёрнутом виде

В развёрнутом виде *Окно логических ключей* содержит таблицу с 3 колонками и количеством строк, равным количеству *Логических ключей*. В первой колонке указывается имя *Логического ключа*, заданное в программе. Во второй колонке — его значение (1 или 0). В третье колонке указывается количество транзактов в *Цепи повторений* ***Retry Chain***, связанной с *Логическими ключами*.

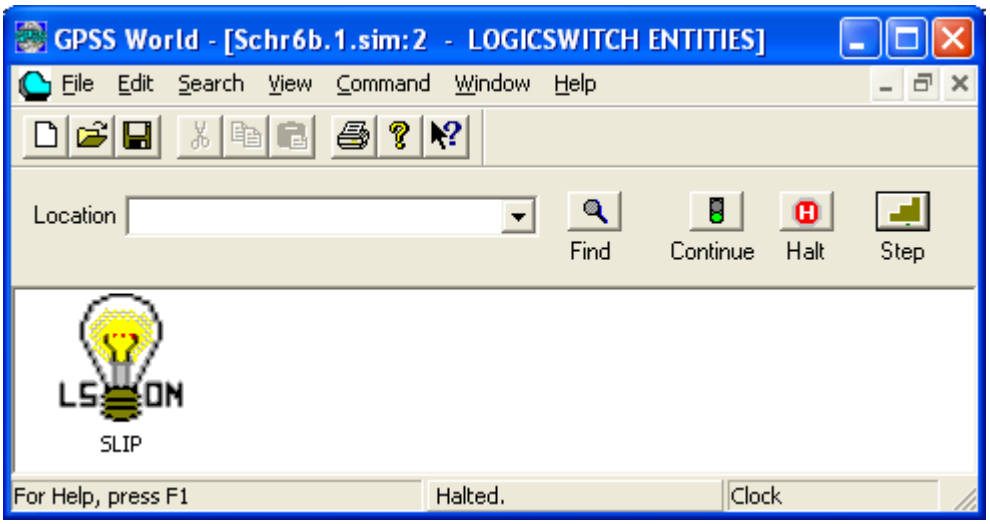


Рисунок 4.23 — *Окно Логических Ключей* ***Logicswitches Window*** в компактном виде

Логических ключей может быть много и они имеют свои имена. *Окно логических ключей* позволяет просматривать их работу. Это необходимо большей частью при отладке программы, но может быть полезно и для получения информации о процессе. *Окно логических ключей* открывается всегда, но заполнено только в том случае, если такие ключи имеются в программе.

Использование *Логических ключей* — специфический приём программирования, поэтому во многих программах их может не быть, так как вместо них можно использовать обычные переменные. Достоинство логических ключей, что их переключение из состояния 1 в 0 и наоборот может происхо-

дить в том числе по команде, обеспечивающей инвертирование значения (т. е. не нужно программно проверять предыдущее значение и подставлять противоположное), а также то, что они занимают одну ячейку памяти.

4.4.6 Окно матриц Matrix Window

Matrix Window — *Окно матриц*. *Матрицы* — разновидности блоков в GPSS World. Они внешне соответствуют матрицам в обычных языках программирования, но имеют и некоторые особенности. *Матрицы* описываются с помощью команды **MATRIX**, а затем заполняются с помощью блока **MSAVEVALUE**. Заполнение *Матрицы* происходит только в тот момент, когда в соответствующий ей блок **MSAVEVALUE** входит транзакт. Что именно и в какую ячейку матрицы записывается, решает программист. *Матриц* может быть много, поэтому каждая *Матрица* имеет уникальное имя или номер. *Матрицы* могут иметь до 6 измерений. При вызове *Окно матриц* откроется только если в программе есть хотя бы одна *Матрица*.

Сразу после вызова *Окна матриц* предлагается выбрать из списка *Матрицу*, вводимую на экран (Рисунок 4.24). Предлагается также выбрать способ вывода матрицы — с нормальным выводам по строкам и столбцам, как это задано в программе, или транспонированную матрицу.

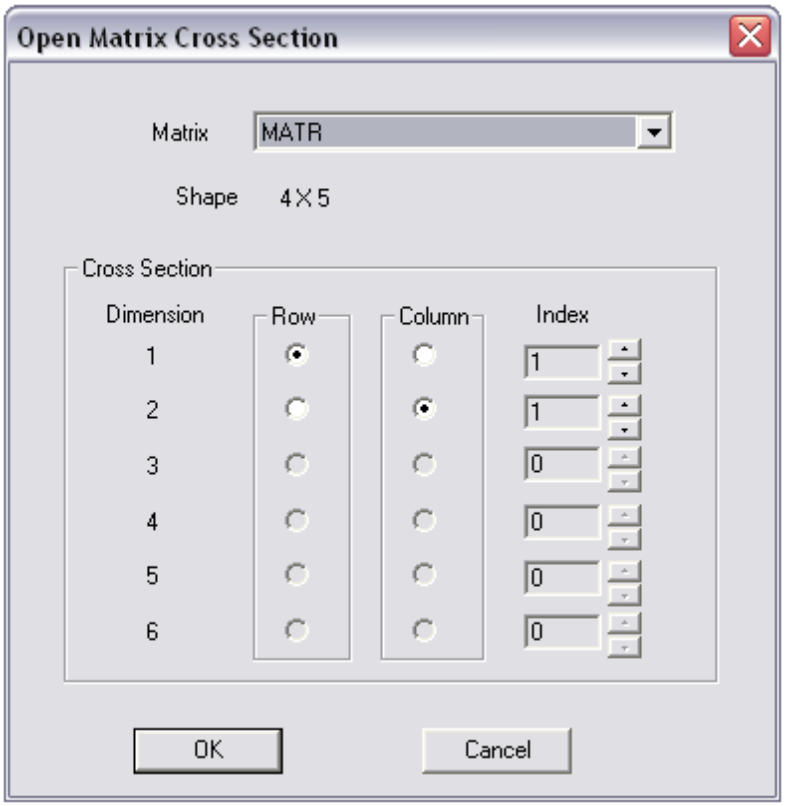


Рисунок 4.24 — Вкладка настройки *Окна матриц* ***Matrix Window***. Задание нормального типа вывода по строкам и столбцам

При выборе нормального типа вывода *Матрицы* она выглядит следующим образом (Рисунок 4.25).

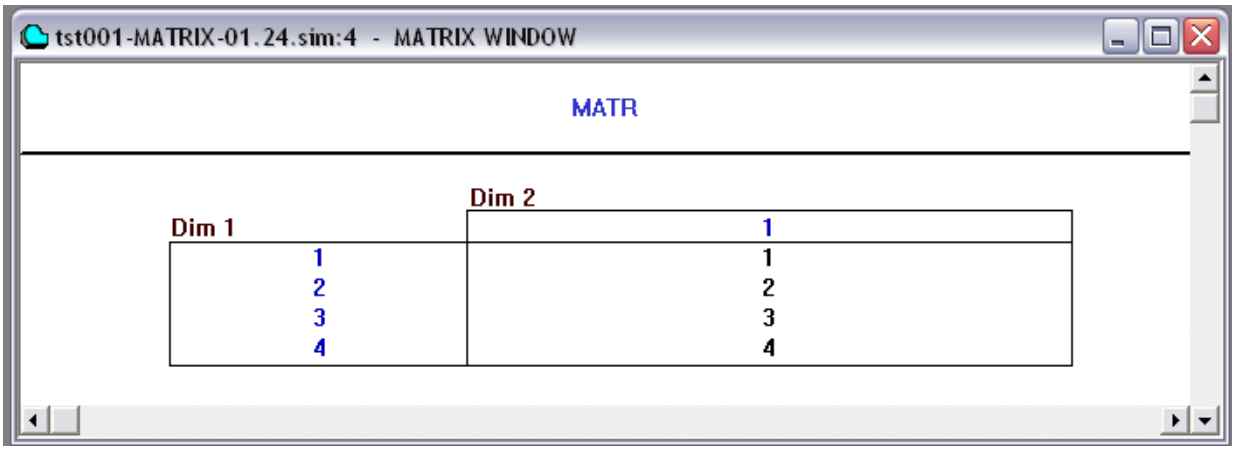


Рисунок 4.25 — Вывод *Матрицы* с нормальным расположением столбцов и строк

Можно поменять указания для **Row** и **Column** (Рисунок 4.26).

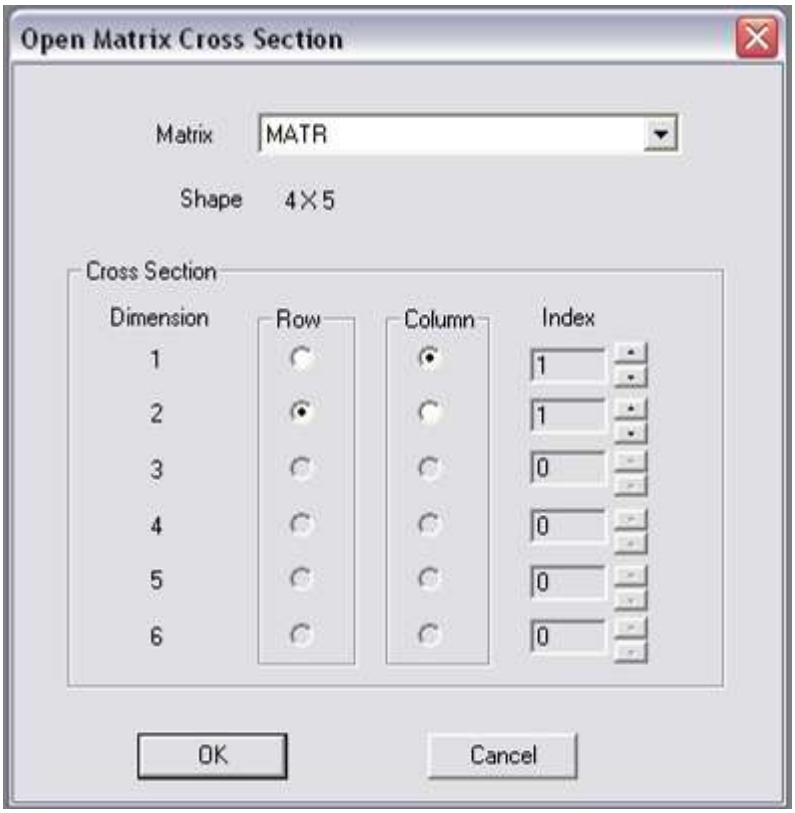


Рисунок 4.26 — Вкладка настройки *Окна матриц* ***Matrix Window***. Задание вывода транспонированной *Матрицы*

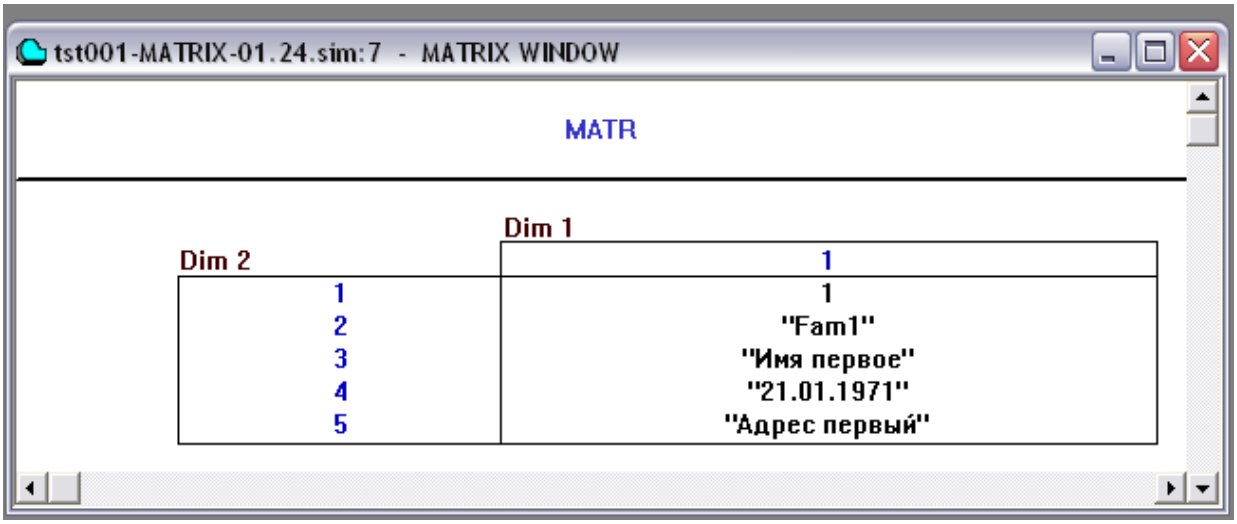


Рисунок 4.27 — Вывод транспонированной *Матрицы*

Матрицы могут просматриваться как в конце сеанса моделирования, так и в процессе моделирования — в этом случае можно наблюдать динамику процессов, отражаемых в них.

4.4.7 Окно графиков Plot Window

Plot Window — окно графиков. Предназначено для визуализации результатов моделирования с помощью СЧА. Графики имеют стандартный вид, который практически невозможно менять.

Используя пункт меню **Window → Simulation Window → Plot Window…**, можно построить до восьми таких графиков. Для организации вывода информации в графическом виде в GPSS Word необходимо выполнить следующие шаги:

1. Создать модель и оттранслировать её, выбрав пункт меню **Command → Creat Simulation**.
2. Выбрать пункт меню **Window → Simulation Window → Plot Window…**
3. Заполнить поля в диалоговом окне Edit Plot Window (Рисунок 4.28) и нажать кнопку **OK**.
4. Запустить процесс имитации, выбрав пункт меню **Command → START**.

Поля диалогового окна Edit Plot Window (Рисунок 4.29) имеют следующий смысл.

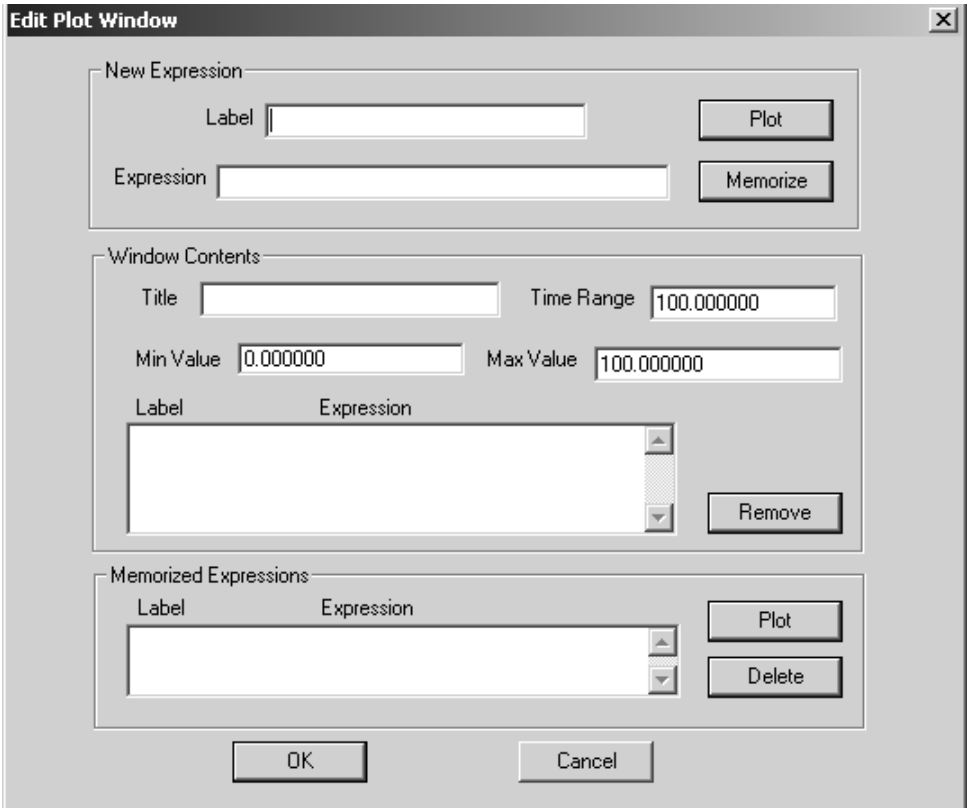


Рисунок 4.28 — *Окно графиков* ***Plot Window***

Группа New Expression (новое выражение) предназначена для добавления нового выражения в список отображаемых выражений. После заполнения этой группы можно нажать кнопку **Plot** для добавления введённого выражения в список отображаемых выражений. Можно также нажать кнопку **Memorize** (запомнить) для сохранения выражения с целью дальнейшего использования. В поле Label задается имя выражения, а в поле Expression — само выражение пользователя.

В группе Window Contents (содержимое окна) отображается список выражений и задается ряд глобальных настроек графика, а в поле Time Range — длительность временно го интервала, отображаемого на графике. Поля Min Value и Max Value определяют соответственно минимальное и максимальное значения отображаемой величины. Кнопка **Remove** используется для удаления выражения.

Группа Memorize Expressions (сохранение выражения) содержит перечень сохраненных выражений.

Для заполнения полей диалогового окна необходимо установить курсор в начале соответствующей строки. Можно также использовать **Tab**, чтобы двигаться от поля к полю. Нельзя использовать **Enter**, чтобы перейти к следующему полю, так как это закроет диалоговое окно, и появится транзакт об ошибке. Используйте **Enter** или выбирайте **OK**, когда вся информация в поле заполнена.

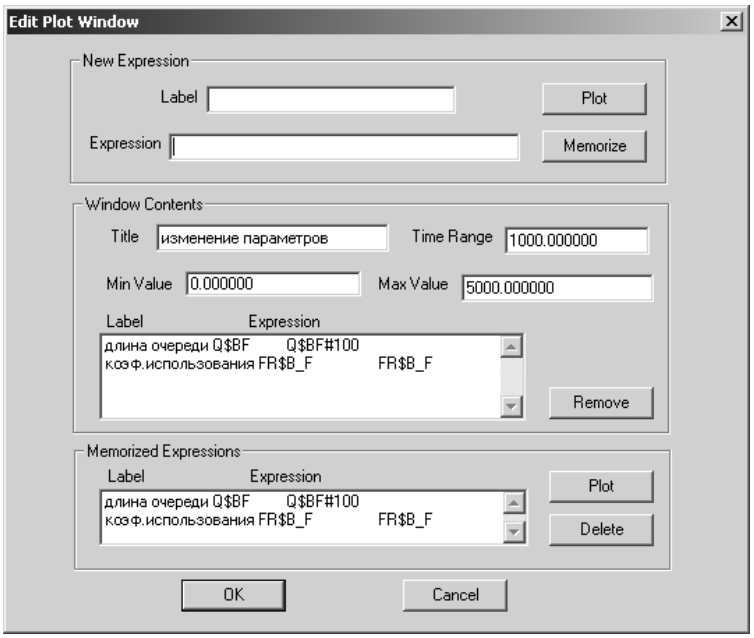


Рисунок 4.29 — Назначение полей диалогового окна ***Edit Plot Window***

Если значения отображаемого выражения выходят за пределы, указанные в параметрах Min Value и Max Value, либо длительность времени имитации превышает Time Range, можно просмотреть интересующие значения выражений, прокрутив изображение с помощью горизонтальных и вертикальных полос прокрутки.

Поле Label содержит имя выражения, которое используется в легенде внизу графика. Напечатайте в поле Label **`Storage in Use,`** в поле Expression — **`S$Pool`**

В поле Title напишем имя, которое опишет оба элемента, из которых мы хотим составить график, а именно: размер очереди и среднее время в очереди, которое будет общим временем от начала до завершения процесса моделирования.

Напечатайте **`Storage in Use & Process Time`** в поле Title.

Теперь определим диапазон времени для оси *x*. В поле Time Range напечатайте **`10000`**. Значения оси *y* имеют значения по умолчанию 0 и 100. Изменим только значение Max Value на 200.

Нажимаем кнопку **Plot** для добавления введённого выражения в список отображаемых выражений. Можно также нажать кнопку **Memorize** (запомнить) для сохранения выражения с целью дальнейшего использования.

Теперь вводим второй набор значений для того же самого графика, не забывая ставить курсор в нужное поле и не использовать клавишу **Enter** между полями. Напечатайте в поле Label текст **`Process Time`**, а в поле Expression — текст **`QT$Process_Time`**. Далее следует нажать кнопку **Plot** и кнопку Memorize. Затем **OK** (рисунок).

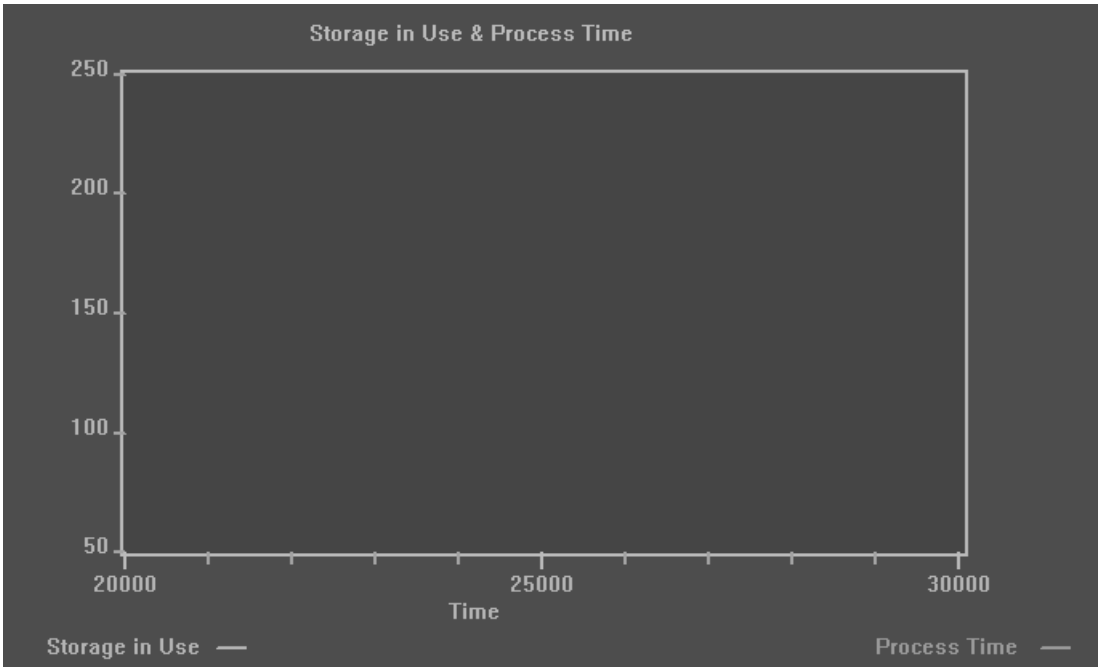


Рисунок 4.30 — Графическое окно

Теперь запускаем процесс моделирования **`START 10000`** (рисунок).

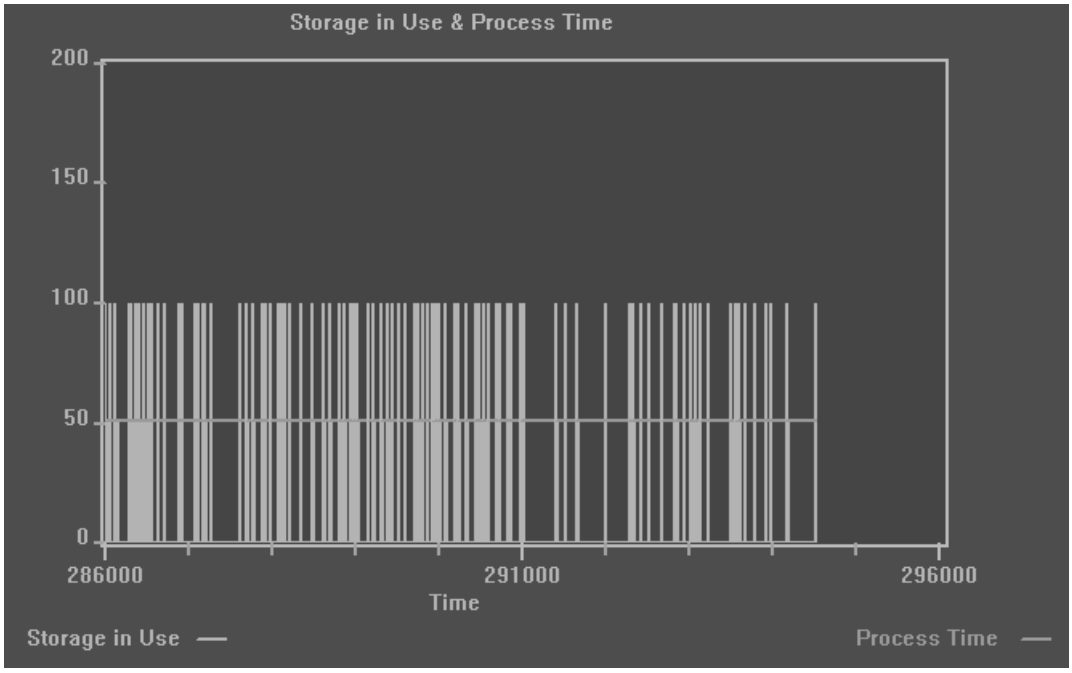


Рисунок 4.31 — *Графическое окно* с двумя выражениями, составляющими график

Внешний вид графиков весьма примитивен, но позволяет оценить характер процесса. Если требуется более качественное графическое представление, можно записать выводимые значения в файл, а затем построить графики с помощью более совершенных графических инструментов.

4.4.8 Окно очередей Queues Window

Queues Window — *Окно очередей*. *Очереди* — разновидность устройств в GPSS World. Очереди возникают, если *Обслуживающее устройство* ***Facilities*** не готово принять транзакт по каким-то причинам (занято, у транзакта не тот приоритет и др.). *Очереди* в программе могут быть или не быть заданы — не зависимо от того, имеются ли они в действительности. Задание имени очереди с помощью оператора **`QUEUE`** (*Очередь*) означает, что в процессе моделирования для неё начинает накапливаться информация, которую и можно посмотреть в *Окне очередей* (Рисунок 4.32).

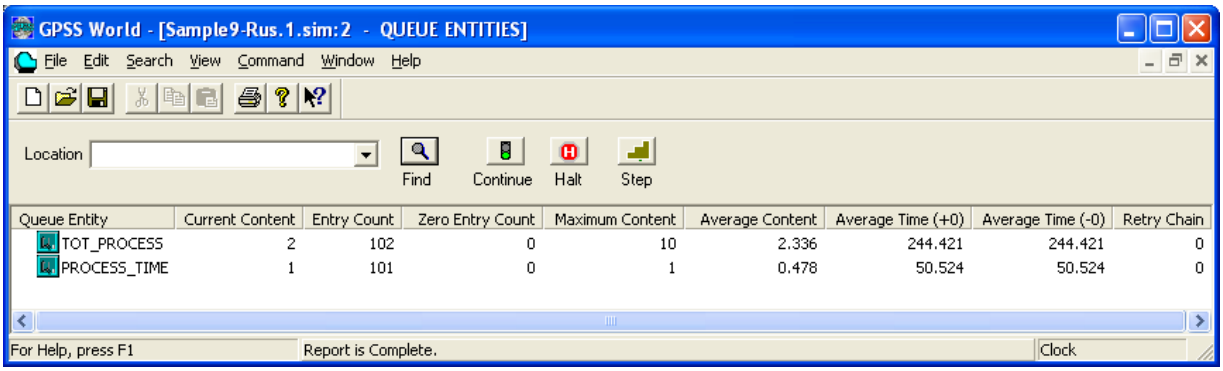


Рисунок 4.32 — *Окно очередей* ***Queues Window***

Окно очередей представляет собой таблицу, в первом столбце которой перечислены названия очередей, включённых в программу на GPSS. Другие столбцы имеют следующее содержание:

Current Content — текущее содержание (число транзактов, находящихся в данный момент в данной очереди).

Entry Count — число входов (общее число входов транзактов с момента начала моделирования).

Zero Entry Count — число "нулевых" входов (число транзактов, которые вошли в очередь, но не задержались в ней по разным причинам и тотчас прошли на выход; это может происходить из-за отсутствия в очереди транзактов, из-за высокого приоритета вошедшего транзакта).

Maximum Content — наибольшая величина (наибольшее число транзактов, которые одновременно были в очереди).

Average Content — среднее количество (средняя величина количества транзактов в очереди за время моделирования).

Average Time (+) — среднее время ожидания транзакта в очереди.

Average Time (–) — среднее время ожидания транзакта в очереди.

Retry Chain — наличие (и их количество) транзактов в *Цепи повторений* (если имеются транзакты, которые не могут войти в очередь, например, из-за того, что она заполнена, они ожидают такой возможности в *Цепи повторений*).

4.4.9 Окно сохраняемых величин Savevalues Window

Savevalues Window — *Окно сохраняемых величин*. В процессе моделирования переменные модели меняют свои значения. Некоторые переменные относятся к определённым классам (*Очереди* ***Queue***, одноканальные *Обслуживающие устройства* ***Facilities***, *Многоканальные устройства* ***Storage***) и для них уже предусмотрены окна представления результатов моделирования. Но в программе могут быть и другие переменные, значения которых могут интересовать пользователя модели. Тогда необходимо значения этих переменных сохранять, чтобы затем построить графики или вывести статистически обработанные значения (среднее, максимальное и др.). Именно эти переменные и появляются в окне ***Savevalues Window***.

Внешний виде *Окна сохраняемых величин* (Рисунок 4.33) типичен для GPSS World.

Окно сохраняемых величин открывается всегда, даже если в программе нет самих *Сохраняемых величин*. Но *Сохраняемые величины* появляются в *Окне* по мере того, как они принимают какое-либо значения. *Сохраняемые величины*, которые не получили никаких значений, не выводятся (Рисунок 4.33).

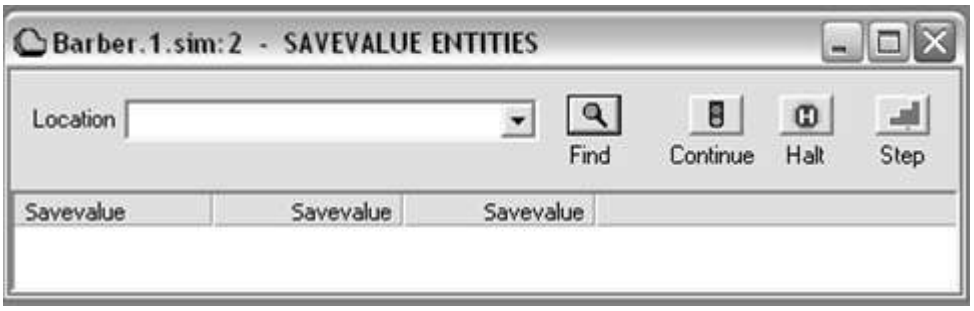


Рисунок 4.33 — *Окно сохраняемых величин* ***Savevalues Window***, когда ни одна *Сохраняемая величина* ещё не получила значения

Если запустить программу на счёт (или несколько раз нажать кнопку Step), в *Окне* появятся имя и значения *Сохраняемых величин* (Рисунок 4.34).

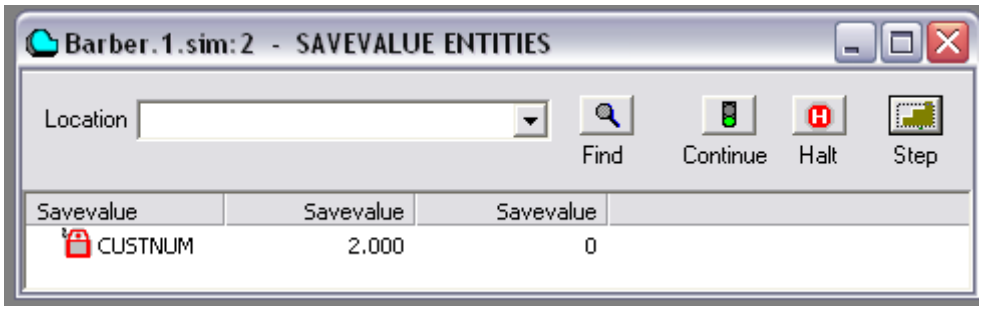


Рисунок 4.34 — *Окно сохраняемых величин* ***Savevalues Window***, когда первая *Сохраняемая величина* не получила значение

При последующей работе модели будут появляться другие *Сохраняемые величины* — в порядке очереди. Поэтому, если для целей удобства наблюдения требуется расположить *Сохраняемые величины* определённым образом, необходимо в такой точно последовательности задать их начальные значения оператором **INITIAL**.

4.4.10 Окно Многоканальных устройств Storages Window

Storages Window — *Окно многоканальных устройств*. Это многоканальные устройства обслуживания, у которых все каналы идентичные. Вместо них можно использовать соответствующее количество одноканальных *Обслуживающих устройств*, но, во-первых, при большом количестве каналов (десятки, сотни, тысячи) получится необозримая программа; во-вторых, чтобы собирать и обрабатывать информацию в этих каналах, придётся писать дополнительные фрагменты программы большого размера. Поэтому для *Многоканальных устройств* используется специальный оператор **STORAGE**. Если в программе такие операторы отсутствуют, то окно *Многоканальных устройств* будет пусто. Если они есть, то в окне *Многоканальных устройств* появится таблица. В первой колонке содержится список *Многоканальных устройств* из программы, а рядом с ним — некоторая информация.

Внешний вид *Окна многоканальных устройств* () типичен для GPSS World.

4.4.11 Окно гистограмм Table Window

Table Window — *Окно гистограмм*. Гистограмма — это график, на котором отражена зависимость частоты встречаемости какого-то параметра от величины этого параметра. Например, гистограмма может отражать, как много транзактов находились в очереди от 0 до 5 минут, от 5 до 10 минут, от 10 до 15 минут, от 15 до 20 минут и т. д. Чтобы построить гистограмму, необходимо предварительно собрать такую информацию. Для этого существуют специальные операторы: **TABLE** (для любых указанных данных) и **QTABLE** (для данных об очередях). Очереди выделены потому, что именно время нахождения в них транзактов часто представляет интерес для исследователя. Поэтому использование специализирующего оператора **QTABLE** предусматривает минимальное число настроек. Универсальный оператор **TABLE** требует для заполнения специального оператора **TABULATE**. Каждая таблица гистограмм имеет своё название. Поэтому при открытии *Окна гистограмм* предлагается выбрать конкретную гистограмму из их списка — GPSS World знает, какие именно таблицы собираются в модели (Рисунок 4.35).

Рисунок 4.35 — Выбор имени выводимой на экран гистограммы в *Окне гистограмм* ***Table Window***

Если работа модели завершилась, то гистограмма отражает конечную статистику исследуемого процесса. Если запустить программу далее, то в процессе её работы будет меняться и выводимая гистограмма, что позволяет наблюдать динамику процесса. Для управления моделированием используются кнопки на панели *Окна гистограмм*, аналогичные таковым в других *Окнах*.

4.5 Совместное использование нескольких окон

Цель моделирования может быть как в определении средних показателей процессов, так и наблюдение за отдельными процессами при их пошаговом моделировании. В последнем случае удобно открыть сразу несколько окон и расположить их рядом на экране дисплея (Рисунок 4.36).

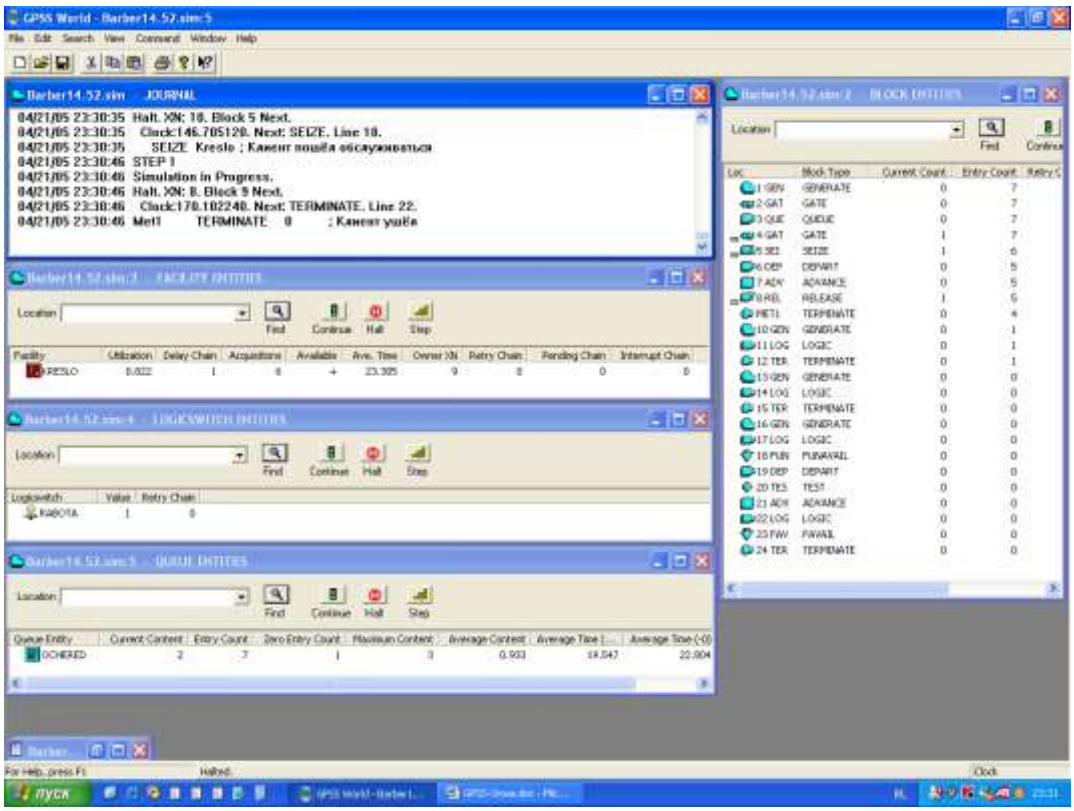


Рисунок 4.36 — Расположение окон для одновременного наблюдения при пошаговом моделировании

При нажатии функциональной клавиши **F5** будет делаться 1 шаг (т. е. совершаться сдвиг на одно ближайшее событие). При этом обычно перемещается один из транзактов, так как именно с ними связаны все события во всех операторах GPSS.

5 Некоторые приёмы программирования в GPSS World

5.1 Общая характеристика программирования

GPSS World позволяет одни и те же задачи решать разными способами, но при этом проявляются различные ограничения или дополнительные возможности.

Необходимо помнить, что *блоки* в GPSS World — это фактически подпрограммы (процедуры), которые выполняют много различных функций (поэтому их и не называют *операторами*, как в других языках). Их последовательное выстраивание представляет собой краткую запись многих операций, скрытых от программиста. Поэтому, хотя некоторые из них на первый взгляд имеют схожие основные функции, второстепенные операции могут быть существенно разными.

Например, существуют различные возможности задавать блоки — с помощью имён или с помощью номеров, а также добавлять к ним метки. Задание номеров позволяет выполнять с их помощью арифметические операции, например, вычислять номер какого-либо блока через номера других блоков или номера транзактов и т. д. Метки также можно задать с помощью имён, а затем имена перенумеровать, и тогда уже метки можно вычислять с помощью арифметических операций. Вообще, в GPSS World предусмотрено много возможностей перехода от имён к числам и наоборот. В то же время, необходимо помнить об основной особенности программ на GPSS World: все блоки "срабатывают" только тогда, когда в них входят активные транзакты. В этом как раз проявляется отличие блоков от операторов. Фактически, перед каждым блоком как бы располагается условный оператор, который проверяет наличие активного транзакта. Это существенно усложняет написание программы, так как транзакты должны обеспечивать "срабатывание" в том числе таких блоков, как запись в файл или чтение из файла, которые прочно ассоциируются с обычными операторами универсальных языков программирования. Это приводит к тому, что для работы некоторых блоков приходится генерировать отдельные транзакты, не имеющие физического смысла, а затем их уничтожать. В связи с такой существенной ролью транзактов, в GPSS World имеется несколько блоков, позволяющих задавать условие для их прохождения — **`TEST`**, **`TRANSFER`**, **`GATE`** и др. Каждый из них имеет особенности в выполнении своих функций, но в некоторых случаях их применение оказывается практически эквивалентным.

Особые проблемы возникают при написании больших программ, содержащих тысячи блоков. В универсальных языках программирования предусмотрена возможность структурирования программного кода с помощью подпрограмм (процедур) пользователя. В этом случае воздаётся библиотека

подпрограмм, которые вызываются в основной программе, что существенно уменьшает её объём. В GPSS World такая возможность частично также предусмотрена в рамках использования встроенного алгоритмического языка PLUS. Но для кода, состоящего собственно из блоков GPSS World, такой возможности нет. Структурирование выполняется с помощью включения в текст одной программы текста другой оператором **INCLUDE**. Но это — совершенно другая технология. В данном случае включаемый программный код представляет собой фрагмент программы и необходимо учитывать основной код до него и после него. Фактически, можно представить себе единый текст программы, из которого вырезаны фрагменты, сохранены в отдельных файлах, а затем вновь вставлены в программу оператором **INCLUDE.** Здесь сложно в разных местах включать одну и ту же последовательность строк, так как необходимо тщательно следить за тем, чтобы включаемый текст был инвариантен относительно места включения (т. е. во всех местах включения выполнял свои функции).

5.2 Использование имён и номеров блоков

В ряде случаев объект моделирования представляется несколькими однотипными фрагментами программы, включающими устройства, очереди, окружающие их операторы. При этом транзакты распределяются по этим фрагментам программы тем или иным образом (случайно, вследствие выполнения каких-либо условий и т. д.). Если таких фрагментов немного, то можно каждый из них записать отдельно (расположив последовательно в программе), давая каждому устройству своё имя, а затем эти имена использовать для управления движением транзактов. Но если фрагментов много или требуется менять их количество, то желательно заменить линейную программу циклической, где каждый фрагмент программы имеет свой номер и именно этот номер используется при управлении движением транзакта (номер можно вычислить, а имя вычислить невозможно). Для этого в GPSS предусмотрены соответствующие механизмы.

Некоторые устройства могут иметь как имена, так и номера: обслуживающие устройства (задаваемые операторами **SEIZE** и **RELEASE**), очереди (задаваемые операторами **QUEUE** и **DEPART**), сохраняемые величины (оператор **SAVEVALUE**).

Пример программы с использованием имён устройств приведён ниже.

```
;-------------------------------------------------------------------------
; GPSS World: tst001-имена_блоков-02.gps
;-------------------------------------------------------------------------
; Работа с блоками, заданными именами.
; Устройства и очереди имеют одинаковые имена Nam.
;-------------------------------------------------------------------------
```

```
GENERATE     2,1
QUEUE        Nam
SEIZE        Nam
DEPART       Nam
ADVANCE      4,3
RELEASE      Nam
TERMINATE    1
```

Рисунок 5.1 — Листинг программы с блоками, заданными именами. Очередь и обслуживающее устройство имеют одинаковые имена

В программе (Рисунок 5.1) оператор **GENERATE** создаёт поток транзактов, генерируя новый транзакт через каждые 2±1 единицу времени. Оператор **QUEUE** включает транзакты в очередь с именем ***Nam***. Оператор **SEIZE** направляет очередной транзакт в обслуживающее устройство с именем ***Vic***, при этом оператор **DEPART** освобождает очередь ***Nam*** на этот вышедший из неё транзакт. Оператор **ADVANCE** задерживает транзакт в обслуживающем устройстве ***Vic*** на время 4±3 единицы времени, из-за чего другие транзакты не могут войти в обслуживающее устройство ***Nam***. После окончания задержки оператор **RELEASE** освобождает устройство ***Nam***. Следующий оператор **TERMINATE** ликвидирует один транзакт. В данном случае *Очередь* и *Обслуживающее устройство* имеют одно и то же имя, но транслятор GPSS World никогда их не перепутает, так как операции с *Очередями* и *Обслуживающими устройствами* задаются разными блоками: : для очередей это — **QUEUE** и **DEPART**, а для обслуживающих устройств — **SEIZE** и **RELEASE**.

```
;-------------------------------------------------------------------------------
; GPSS World: tst001-имена_блоков-01.gps
;-------------------------------------------------------------------------------
; Работа с блоками, заданными именами
; Устройства и очереди имеют разные имена.
;-------------------------------------------------------------------------------
      GENERATE     2,1
      QUEUE        Vic
      SEIZE        Serv
      DEPART       Vic
      ADVANCE      4,3
      RELEASE      Serv
      TERMINATE    1
```

Рисунок 5.2 — Листинг программы с блоками, заданными именами. Очередь и обслуживающее устройство имеют разные имена

В программе (Рисунок 5.2) те же блоки, что и в предыдущей программе (Рисунок 5.1), в данном случае обслуживающее устройство имеет имя ***Serv***, а очередь — ***Vic***. В результате несколько проще читать текст программы, но следует помнить, к какому блоку относится соответствующее имя.

Кроме имён, можно использовать номера блоков.

Пример программы с использованием номеров блоков приведён ниже (Рисунок 5.3).

```
;-------------------------------------------------------------------------------
; GPSS World: tst01-02.gps
;-------------------------------------------------------------------------------
```

```
; Работа с блоками, заданными номерами
;-------------------------------------------------------------------------
          GENERATE      2,1
          QUEUE         1
          SEIZE         1
          DEPART        1
          ADVANCE       4,3
          RELEASE       1
          TERMINATE     1
```

Рисунок 5.3 — Листинг программы с блоками, заданными номерами. Очередь и обслуживающее устройство имеют один номер

В программе (Рисунок 5.3) оператор **GENERATE** создаёт поток транзактов, генерируя новый транзакт через каждые 2±1 единицу времени. Оператор **QUEUE** включает транзакты в очередь с номером 1. Оператор **SEIZE** направляет очередной транзакт в обслуживающее устройство с номером 1, при этом оператор **DEPART** освобождает очередь 1 на этот вышедший из неё транзакт. Оператор **ADVANCE** задерживает транзакт в обслуживающем устройстве 1 на время 4±3 единицы времени, из-за чего другие транзакты не могут войти в обслуживающее устройство 1. После окончания задержки оператор **RELEASE** освобождает устройство 1. Следующий оператор **TERMINATE** ликвидирует один транзакт.

Таким образом, цифра 1 в программе имеет разное значение. В операторах **QUEUE** и связанном с ним **DEPART** цифра 1 означает номер очереди (1-я очередь среди всех других очередей, если бы были другие). В операторах **SEIZE** и связанном с ним **RELEASE** цифра 1 означает номер обслуживающего устройства. Очередь номер 1 и обслуживающее устройство номер 1 — это совершенно разные блоки. Для программы нет проблем различить их, так как обращение к ним осуществляется различными операторами: для очередей это — **QUEUE** и **DEPART**, а для обслуживающих устройств — **SEIZE** и **RELEASE**.

Не обязательно нумерацию начинать с цифры 1 (так как это — условный номер, а не номер "по порядку"). Например (Рисунок 5.4). В программе (Рисунок 5.4) использованы различные номера для очереди и обслуживающего устройства. Использование различных или одинаковых номеров для них имеет свои достоинства или недостатки.

Разные номера позволяют лучше различать в тексте очереди и обслуживающие устройства, особенно, если они не связаны друг с другом (например, имеется общая очередь для двух обслуживающих устройств).

```
;-------------------------------------------------------------------------
; GPSS World: tst01-02.gps
;-------------------------------------------------------------------------
; Работа с блоками, заданными номерами
;-------------------------------------------------------------------------
          GENERATE      2,1
          QUEUE         7
```

```
SEIZE         5
DEPART        7
ADVANCE       4,3
RELEASE       5
TERMINATE     1
```

Рисунок 5.4 — Листинг программы с блоками, заданными номерами. Очередь и обслуживающее устройство имеют различные номера

Одинаковые номера позволяют увязать между собой обслуживающие устройства и соответствующие им очереди, если такая связь есть. Наиболее удобно использовать одинаковые номера в случае вычисления номеров очередей и обслуживающих устройств (Рисунок 5.5).

```
;-------------------------------------------------------------------------
; GPSS World: tst01-03
;-------------------------------------------------------------------------
; Распределение транзактов по различным устройствам с использованием
; номеров вместо имён устройств
;-------------------------------------------------------------------------
        GENERATE   2,1                  ; Генерирование очередного транзакта
        ASSIGN     1,(DUniForm(3,1,10)) ; Выбор направления перемещения
                                        ; Номер направления записан в 1-й параметр
                                        ; сгенерированного транзакта
        QUEUE      P1                   ; Очередь с заданным номером, взятым из
                                        ; 1-го параметра идущего транзакта
        SEIZE      P1                   ; Обслуживающее устройство
        DEPART     P1                   ; Освобождение очереди
        ADVANCE    7,2                  ; Время обработки (одно для всех)
        RELEASE    P1                   ; Освобождение обслуживающего устройства
        TERMINATE  1                    ; Уничтожение транзакта
```

Рисунок 5.5 — Листинг программы с блоками, заданными номерами. Очередь и обслуживающее устройство имеют одинаковые номера, задаваемые через параметр транзакта

В программе (Рисунок 5.5) оператор **GENERATE** создаёт через каждые 2±1 единицу времени новый транзакт. С помощью функций генерирования последовательностей случайных чисел ***DUniForm*** задаётся случайный номер в диапазоне от 1 до 10 (при этом используется генератор случайных чисел № 3, на что указывает первый операнд в скобках функции), который оператором **ASSIGN** присваивается параметру № 1 этого транзакта. Затем транзакт поступает в оператор **QUEUE**, который направляет его в очередь с номером, записанным в параметр № 1 этого транзакта (об этом говорит операнд ***P1***, который распадается на две части: ***P*** — СЧА, означающий параметр; ***1*** — номер параметра). Затем последовательно занимается обслуживающее устройство и освобождается очередь с номером, записанным в параметр № 1 транзакта. После задержки оператором **ADVANCE** на 7±2 единицы времени транзакт покинет своё обслуживающее устройство с номером, записанным в параметре № 1 транзакта. Таким образом, у каждого транзакта, сразу после его появления в программе, с помощью оператора **ASSIGN** создаётся параметр с номером 1, в котором хранится информация о том, в какой группе очередей и обслуживающих устройств он должен обслуживаться.

В данной программе время обслуживания в обслуживающем устройстве, заданное оператором **ADVANCE**, составляет 7±2 единиц времени — одну и ту же величину для всех обслуживающих устройств. Чтобы задать для каждого обслуживающего устройства своё время обслуживания, можно использовать матрицу или (что проще) — функцию (Рисунок 5.6).

```
;-------------------------------------------------------------------------
; GPSS World: tst01-04
;-------------------------------------------------------------------------
; Распределение транзактов по различным устройствам с использованием
; номеров вместо имён устройств
;-------------------------------------------------------------------------
Tmp       FUNCTION     P1,L10                ; Функция задания времени обслуживания
1,5./2,4./3,3.5/4,4.5/5,5./6,3.5/7,4.2/8,5.2/9,3.8/10,3.1
TmpS      FUNCTION     P1,L10                ; Фунция задания разброса времени
1,1./2,2./3,1.5/4,2.5/5,1.8/6,1.1/7,2.1/8,1.7/9,0.8/10,2.1
;---------
          GENERATE     1                     ; Генерирование очередного транзакта
          ASSIGN       1,(DUniForm(3,1,10))  ; Выбор направления перемещения
          QUEUE        P1                    ; Очередь с заданным номером
          SEIZE        P1                    ; Обслуживающее устройство
          DEPART       P1                    ; Освобождение очереди
          ADVANCE      FN$Tmp,FN$TmpS        ; Время обработки (разное)
          RELEASE      P1                    ; Освобождение обслуживающего устройства
          TERMINATE    1                     ; Уничтожение транзакта
```

Рисунок 5.6 — Листинг программы с блоками, заданными номерами. Задание аргумента функции через параметр

В программе (Рисунок 5.6) вычисляются не только номера очереди (**QUEUE**), обслуживающего устройства (**SEIZE**) и сохраняемой величины (**SAVEVALUE**), но также величина времени обслуживания и диапазон его разброса (**ADVANCE**). Для этого используются две функции: время обслуживания задаётся функцией ***Tmp***, а диапазон его разброса — функцией ***TmpS***. Функции задаются с помощью оператора описания **FUNCTION**. В поле метки слева от него записывается имя функции. Справа — два операнда. Первый операнд (в данном случае ***P1***) содержит указание на ту переменную, которая будет задавать аргумент функции (в данном случае это — параметр № 1 транзакта). Второй операнд (в данном случае ***L10***) содержит указание на тип функции (буква ***L***) и количество пар "аргумент — функция" (в данном случае 10). Ниже располагается строчка, содержащая пары "аргумент — функция", разделённые косой чертой. В данном случае на месте аргументов стоят последовательные цифры от 1 до 10, а на месте функций — значения времён обслуживания (у ***Tmp***) и их разброса (у ***TmpS***). Работа функций в операторе **ADVANCE** обеспечивается записью имени функции на соответствующем месте с использованием СЧА ***FN*** и выглядит как ***FN$Tmp*** и ***FN$TmpS***. Когда очередной транзакт входит в оператор **ADVANCE**, вызываются функции ***Tmp*** и ***TmpS***. При этом в соответствии с их описанием вычисляется значение параметра № 1 вошедшего транзакта (***P1***) — оно лежит в диапазоне от 1 до 10 (в соответствии со сработанной выше функцией ***DUniForm***). Выбирается та

пара "аргумент — функция", в которой аргумент равен значению параметра № 1, а затем выбирается соответствующее значение функции и подставляется на место функции.

Аргумент функции не обязательно задаётся параметром: может быть любой другой СЧА, например, сохраняемая величина (**SAVEVALUE**) (Рисунок 5.7).

```
;-------------------------------------------------------------------------------
; GPSS World: tst01-05
;-------------------------------------------------------------------------------
; Распределение транзактов по различным устройствам с использованием
; номеров вместо имён устройств
;-------------------------------------------------------------------------------
Tmp       FUNCTION     P1,L10
1,5./2,4./3,3.5/4,4.5/5,5./6,3.5/7,4.2/8,5.2/9,3.8/10,3.1
TmpS      FUNCTION     X1,L10
1,1./2,2./3,1.5/4,2.5/5,1.8/6,1.1/7,2.1/8,1.7/9,0.8/10,2.1
;---------
          GENERATE     1                      ; Генерирование очередного транзакта
          ASSIGN       1,(DUniForm(3,1,10))   ; Выбор направления перемещения
          QUEUE        P1                     ; Очередь с заданным номером
          SEIZE        P1                     ; Обслуживающее устройство
          DEPART       P1                     ; Освобождение очереди
          SAVEVALUE    1,P1                   ; Определение длины покинутой очереди
          ADVANCE      FN$Tmp,FN$TmpS         ; Время обработки (разное)
          RELEASE      P1                     ; Освобождение обслуживающего устройства
          TERMINATE    1                      ; Уничтожение транзакта
```

Рисунок 5.7 — Листинг программы с блоками, заданными номерами. Задание аргумента функции через параметр P1 и сохраняемую величину X1

В программе (Рисунок 5.7) аргумент функции ***Tmp*** задаётся параметром транзакта (записано ***Tmp*** **FUNCTION** ***P1,L10***), а функции ***TmpS*** — сохраняемой величиной (записано ***TmpS*** **FUNCTION** ***X1,L10***), что отражено в их описании вверху программы. В данном случае значение сохраняемой величины равно значению параметра, что обеспечивается оператором **SAVEVALUE** ***1,P1***, но это — не обязательно.

Один и тот же транзакт может последовательно проходить через разные обслуживающие устройства (Рисунок 5.8).

В программе (Рисунок 5.8) организован цикл между меткой ***t1000*** и оператором **TEST** ***E P1,10,t10000***. Оператор **GENERATE** генерирует один транзакт (на это указывает его четвёртый операнд), после чего оператором **ASSIGN** создаётся его параметр № 1, которому присваивается значение 0. Далее стоит ещё один оператор **ASSIGN**. Каждый раз, когда транзакт входит в этот оператор, значение его первого параметра увеличивается на 1 (**ASSIGN** ***1+,1***): первый операнд задаёт номер параметра (1), после него стоит знак +, означающий увеличение, второй операнд — число 1, означающее величину приращения. Таким образом единственный сгенерированный транзакт пройдёт последовательно через все 10 обслуживающих устройств. Цикл закончится, когда переменная цикла (параметр ***P1***) станет равна 10 (**TEST** ***E P1,10,t1000***).

```
;-------------------------------------------------------------------------
; GPSS World: tst01-06
;-------------------------------------------------------------------------
; Распределение транзактов по различным устройствам с использованием
; номеров вместо имён устройств
;-------------------------------------------------------------------------
Tmp      FUNCTION   P1,L10          ; Функция задания времени обслуживания
1,5./2,4./3,3.5/4,4.5/5,5./6,3.5/7,4.2/8,5.2/9,3.8/10,3.1
TmpS     FUNCTION   X1,L10          ; Фунция задания разброса времени
1,1./2,2./3,1.5/4,2.5/5,1.8/6,1.1/7,2.1/8,1.7/9,0.8/10,2.1
;---------
         GENERATE   ,,,1            ; Генерирование единственного транзакта
         ASSIGN     1,0             ; Обнуление счётчика цикла
t1000    ASSIGN     1+,1            ; Определение номера прохода в цикле
         QUEUE      P1              ; Очередь с заданным номером
         SEIZE      P1              ; Обслуживающее устройство
         DEPART     P1              ; Освобождение очереди
         SAVEVALUE  1,P1            ; Определение длины покинутой очереди
         ADVANCE    FN$Tmp,FN$TmpS  ; Время обработки (разное)
         RELEASE    P1              ; Освобождение обслуживающего устройства
         TEST E     P1,10,t1000     ; Проверка на окончание цикла
         TERMINATE  1               ; Уничтожение транзакта
```

Рисунок 5.8 — Листинг программы с блоками, заданными номерами. Прохождение одного транзакта через группу последовательных устройств

Можно смоделировать ситуацию со многими транзактами и заставить каждый из них последовательно пройти все обслуживающие устройства (возможно, ожидая в соответствующей очереди, пока очередное устройство освободится) (Рисунок 5.9).

```
;-------------------------------------------------------------------------
; GPSS World: tst01-07
;-------------------------------------------------------------------------
; Распределение транзактов по различным устройствам с использованием
; номеров вместо имён устройств
;-------------------------------------------------------------------------
Tmp      FUNCTION   P1,L10          ; Функция задания времени обслуживания
1,5./2,4./3,3.5/4,4.5/5,5./6,3.5/7,4.2/8,5.2/9,3.8/10,3.1
TmpS     FUNCTION   X1,L10          ; Фунция задания разброса времени
1,1./2,2./3,1.5/4,2.5/5,1.8/6,1.1/7,2.1/8,1.7/9,0.8/10,2.1
;---------
         GENERATE   1               ; Генерирование единственного транзакта
         ASSIGN     1,0             ; Обнуление счётчика цикла
t1000    ASSIGN     1+,1            ; Определение номера прохода в цикле
         QUEUE      P1              ; Очередь с заданным номером
         SEIZE      P1              ; Обслуживающее устройство
         DEPART     P1              ; Освобождение очереди
         SAVEVALUE  1,P1            ; Определение длины покинутой очереди
         ADVANCE    FN$Tmp,FN$TmpS  ; Время обработки (разное)
         RELEASE    P1              ; Освобождение обслуживающего устройства
         TEST E     P1,10,t1000     ; Проверка на окончание цикла
         TERMINATE  1               ; Уничтожение транзакта
```

Рисунок 5.9 — Листинг программы с блоками, заданными номерами. Прохождение нескольких транзактов через группу последовательных устройств

В программе (Рисунок 5.9) непрерывно генерируется поток транзактов (новый транзакт появляется через каждую 1 единицу времени). Затем выполняются те же операции, что и в программе (Рисунок 5.8), но для каждого транзакта. Транзакты двигаются один за другим, переходя последовательно

от одного обслуживающего устройства к другому и скапливаясь в соответствующих очередях перед ними.

Если вместо одноканальных обслуживающих устройств используются многоканальные устройства, то возникает проблема, связанная с тем, что многоканальные устройства обязательно должны иметь имена — они не могут быть пронумерованы в операторе описания **STORAGE**, так как имя стоит в поле метки, а метка не может быть числом. Поэтому для использования многоканальных устройств в циклах используют специальный вид функций, в которых аргументами являются числа, а функциями — имена (Рисунок 5.10).

```
;-------------------------------------------------------------------------
; GPSS World: tst01-08
;-------------------------------------------------------------------------
; Распределение транзактов по различным устройствам с использованием
; номеров вместо имён устройств
;-------------------------------------------------------------------------
Tmp       FUNCTION   P1,L10              ; Функция задания времени обслуживания
1,5./2,4./3,3.5/4,4.5/5,5./6,3.5/7,4.2/8,5.2/9,3.8/10,3.1
TmpS      FUNCTION   P1,L10              ; Фунция задания разброса времени
1,1./2,2./3,1.5/4,2.5/5,1.8/6,1.1/7,2.1/8,1.7/9,0.8/10,2.1
Serv      FUNCTION   P1,L10              ; Функция задания имен МКУ
1,Serv01/2,Serv02/3,Serv03/4,Serv04/5,Serv05/6,Serv06/7,Serv07/
8,Serv08/9,Serv09/10,Serv10
;
Serv01    STORAGE    4                  ; Описание многоканального устройства
Serv02    STORAGE    4                  ; Описание многоканального устройства
Serv03    STORAGE    4                  ; Описание многоканального устройства
Serv04    STORAGE    4                  ; Описание многоканального устройства
Serv05    STORAGE    4                  ; Описание многоканального устройства
Serv06    STORAGE    4                  ; Описание многоканального устройства
Serv07    STORAGE    4                  ; Описание многоканального устройства
Serv08    STORAGE    4                  ; Описание многоканального устройства
Serv09    STORAGE    4                  ; Описание многоканального устройства
Serv10    STORAGE    4                  ; Описание многоканального устройства
;---------
          GENERATE   1                  ; Генерирование потока транзактов
          ASSIGN     1,0                ; Обнуление счётчика цикла
t10       ASSIGN     1+,1               ; Определение номера прохода в цикле
          QUEUE      P1                 ; Очередь с заданным номером
          ENTER      FN$Serv            ; Многоканальное устройство занято
          DEPART     P1                 ; Освобождение очереди
          ADVANCE    FN$Tmp,FN$TmpS     ; Время обработки (разное)
          LEAVE      FN$Serv            ; Многоканальное устройство освобождено
          TEST E     P1,10,t10          ; Проверка на окончание цикла
          TERMINATE  1                  ; Уничтожение транзакта
```

Рисунок 5.10 — Листинг программы с блоками, заданными номерами. Прохождение нескольких транзактов через группу многоканальных устройств

В программе (Рисунок 5.10) имеется 10 многоканальных устройств (вместо ранее рассмотренных 10 одноканальных обслуживающих устройств). Каждое многоканальное устройство должно быть описано оператором **STORAGE**, и это сделано в верхней части программы. Справа от оператора **STORAGE** стоит число каналов (везде по 4), а слева (в поле метки) — имя: ***Serv01*** и т. д. Для того, чтобы эти имена можно было заменить числами, используется описанная в начале программы функция ***Serv*** (по аналогии с двумя другими функциями). У неё в качестве аргумента используются числа

(например, номера многоканальных устройств), а в качестве функций — имена ***Serv01*** и т. д. — т. е. те же имена, что и при описании в операторах **STORAGE**. Поскольку все пары "аргумент — функция" не вмещаются в одну строку, они записаны в две строки. При обращении к многоканальному устройству с помощью операторов **ENTER** и **LEAVE** используется вызов этой функции в виде ***FN$Serv***, где, как и ранее, ***FN*** является соответствующим СЧА.

Таким же образом можно отправлять транзакты в очереди и обслуживающие устройства, давая им имена, но при этом вместо имён во всех отчётах будут присутствовать внутренние номера соответствующих очередей и обслуживающих устройств, данные им транслятором (начинаются с 10000). Поэтому нет смысла усложнять программу таким образом.

Вычислять можно не только номер устройства, но и номер метки (Рисунок 5.11), используя это, например, для распределения транзактов по различным фрагментам программы.

В программе (Рисунок 5.11) с помощью функции ***NamMet*** задаётся последовательность пар "аргумент — функция", где аргументами являются числа (например, номера), а функциями — названия меток, расположенные ниже в программе. К этим меткам можно обращаться по имени, а также с помощью вызова функции ***NamMet***. В описании этой функции задана переменная, через которую в функцию должно передаваться значение аргумента, это — параметр транзакта с именем ***Num*** (выражение ***P$Num*** состоит из двух частей: буква ***P*** является СЧА, означающим параметр транзакта, а имя ***Num*** является заданным именем этого параметра, оно может быть выбрано любым). Ранее для этих же целей использовался номер транзакта, но можно использовать и имя.

После создания очередного транзакта оператором **GENERATE** он входит в оператор **ASSIGN**, где его параметру с именем ***Num*** придаётся значение, вычисленное с помощью функции ***DUniForm*** (случайное целое число в диапазоне от 1 до 5, с использованием генератора случайных числе № 2 — номер генератора может быть выбран любым от1 до 7). Оператор **TRANSFER** отправляет этот транзакт на один из фрагментов программы с метками ***t100*** … ***t500***. Чтобы перейти от числа (записанного в параметре ***Num***) к метке, используется определённая ранее функция ***NamMet***. Для этого в операторе **TRANSFER** на месте того операнда, где должна стоять метка перехода, записывается вызов функции в виде ***FN***$***NamMet***. Эта функция вызывается, проверяется значение параметра Num, по нему среди пар "аргумент — функция" выбирается пара, где аргументом является соответствующее число, и для него берётся соответствующее ему имя метки, на которую и отправляется транзакт.

```
;-------------------------------------------------------------------------------
; GPSS World: tst01-09.gps
;-------------------------------------------------------------------------------
; Переход по вычисленной метке
;===============================================================================
NamMet    FUNCTION     P$Num,L5              ; Функция задания метки
1,t100/2,t200/3,t300/4,t400/5,t500
;-------------------------------------------------------------------------------
; Движение транзактов
;-------------------------------------------------------------------------------
          GENERATE     1                     ; Генерация транзакта
          ASSIGN       Num,(DUniform(2,1,5))  ; Случайный выбор метки
          TRANSFER     ,FN$NamMet            ; Переход на сгенерированную метку
;-----------------------------------------------
t100      QUEUE        1                     ; Пришёл на метку 1
          SAVEVALUE    NamMet1,XN1           ; Отметка на метке 1
          ADVANCE      1                     ; Находится в метке 1
          DEPART       1                     ; Освободил очередь 1
          TRANSFER     ,t10                  ; Ушёл с метки 1
;-----------------------------------------------
t200      QUEUE        2                     ; Пришёл на метку 2
          SAVEVALUE    NamMet2,XN1           ; Отметка на метке 2
          ADVANCE      1                     ; Находится в метке 2
          DEPART       2                     ; Освободил очередь 2
          TRANSFER     ,t10                  ; Ушёл с метки 2
;-----------------------------------------------
t300      QUEUE        3                     ; Пришёл на метку 3
          SAVEVALUE    NamMet3,XN1           ; Отметка на метке 3
          ADVANCE      1                     ; Находится в метке 3
          DEPART       3                     ; Освободил очередь 3
          TRANSFER     ,t10                  ; Ушёл с метки 3
;-----------------------------------------------
t400      QUEUE        4                     ; Пришёл на метку 4
          SAVEVALUE    NamMet4,XN1           ; Отметка на метке 4
          ADVANCE      1                     ; Находится в метке 4
          DEPART       4                     ; Освободил очередь 4
          TRANSFER     ,t10                  ; Ушёл с метки 4
;-----------------------------------------------
t500      QUEUE        5                     ; Пришёл на метку 5
          SAVEVALUE    NamMet5,XN1           ; Отметка на метке 5
          ADVANCE      1                     ; Находится в метке 5
          DEPART       5                     ; Освободил очередь 5
          TRANSFER     ,t10                  ; Ушёл с метки 5
;-----------------------------------------------
t10       TERMINATE    1                     ; Транзакт ушёл
```

Рисунок 5.11 — Листинг программы с блоками, заданными номерами. Распределение транзактов по фрагментам программы с заменой меток числами

6 Задания для самостоятельной работы

Задания предназначены для развития **навыков** использования конкретных приёмов программирования в GPSS World. Каждый студент должен выполнить по 11 заданий с последовательным усложнением задачи и программной модели. Объект моделирования задаётся преподавателем:

Задание 1. Моделирование одноканального устройства без очереди.

Задание 2. Моделирование одноканального и многоканального устройств без очереди.

Задание 3. Моделирование одноканального и многоканального устройств с очередью.

Задание 4. Моделирование одноканального и многоканального устройств с очередью и продвижением времени.

Задание 5. Моделирование одноканального и многоканального устройств с очередью, продвижением времени и изменением траектории движения транзактов.

Задание 6. Моделирование одноканального и многоканального устройств с очередью, продвижением времени, изменением траектории движения транзактов, с включением и выключением обоих обслуживающих устройств.

Задание 7. Моделирование одноканального и многоканального устройств с очередью, продвижением времени, изменением траектории движения транзактов, с включением и выключением обоих обслуживающих устройств, принудительным освобождением обслуживающего устройства.

Задание 8. Моделирование одноканального и многоканального устройств с очередью, продвижением времени, изменением траектории движения транзактов, с включением и выключением обоих обслуживающих устройств, принудительным освобождением обслуживающего устройства, заданием параметров транзактов.

Задание 9. Моделирование одноканального и многоканального устройств с очередью, продвижением времени, изменением траектории движения транзактов, с включением и выключением обоих обслуживающих устройств, принудительным освобождением обслуживающего устройства, с заданием параметров транзактов и работой в цикле.

Задание 10. Моделирование одноканального и многоканального устройств с очередью, продвижением времени, изменением траектории движения транзактов, с включением и выключением обоих обслуживающих устройств, принудительным освобождением обслуживающего устройства, с заданием параметров транзактов и работой в цикле, с записью результатов в таблицу;

Задание 11. Моделирование одноканального и многоканального устройств с очередью, продвижением времени, изменением траектории движения транзактов, с включением и выключением обоих обслуживающих устройств, принудительным освобождением обслуживающего устройства, с за-

данием параметров транзактов и работой в цикле, с записью результатов в таблицу и выводом в файл.

Задание защищается преподавателю с подробным объяснением всех использованных приёмов программирования.

Задания 1, 2, 3 защищаются не позднее 5-й недели, задания 4, 5, 6, 7 — не позднее 10-й недели, задания 8, 9, 10, 11 — не позднее 16-й недели. Защита задания осуществляется во время лабораторных занятий в компьютерном классе.

Заключение

Данное учебное пособие предназначено для самостоятельного изучения программирования на GPSS World, поэтому оно носит явную практическую направленность. Реально научиться программировать можно только, имея возможность заниматься этим в спокойных условиях, например, дома. Изучение программирования, в том числе имитационного, в классе не позволяет сосредоточиться, проверить много вариантов и т. д. Поэтому программирование в домашних условиях является наилучшей формой изучения такого материала.

Невозможно осветить все возможности имитационного программирования, поэтому в данном учебном пособии рассмотрены только некоторые начальные сведения, которые могут помочь обучающемуся на начальном этапе обучения. Поэтому во многих случаях приводимый материал является справочным.

Дальнейшая работа должна быть направлена на самостоятельную постановку задач имитационного моделирования и поиск конкретных программистских решений. При этом только непосредственная практика может помочь приобрести необходимый опыт.

Библиографический список

Основная литература:

1 Боев, В. Д. Моделирование систем. Инструментальные средства GPSS World : учеб. пособие / В. Д. Боев. — СПб.: БХВ-Петербург, 2007. — 368 с. — ISBN 5-94157-515-7.

2 Кудрявцев, Е. М. GPSS World. Основы имитационного моделирования различных систем / Е. М. Кудрявцев. — М.: ДМК Пресс, 2008. — 320 с. — (Серия "Проектирование"). — ISBN 5-94074-219-X.

3 Томашевский, В. Н. Имитационное моделирование в среде GPSS / В. Н. Томашевский, Е. Г. Жданова. — М.: Бестселлер, 2006. — 416 с. — ISBN 5-98158-004-6.

4 Рыжиков, Ю. И. Имитационное моделирование. Теория и технологии / Ю. И. Рыжиков. — СПб.: КОРОНА принт; М.: Альтекск-А, 2009. — 384 с. — ISBN 5-94271-021-X; 5-7931-0278-7.

Дополнительная литература:

5 Шрайбер, Т. Дж. Моделирование на GPSS / Т. Дж. Шрайбер. — М.: Машиностроение, 1979. — 592 с.

6 Советов, Б. Я. Моделирование систем. Практикум / Б. Я. Советов, С. А. Яковлев. — М.: Высшая школа, 1999. — 224 с.

7 GPSS World. Reference Manual [Электронное издание] / Minuteman Software. — 4th Edition. — Holly Springs, NC, U.S.A., 2001. — 305 p.

8 GPSS World. Tutorial Manual [Электронное издание] / Minuteman Software. —Holly Springs, NC, U.S.A., 2001. — 277 p.

9 Система программного обеспечения для имитационного моделирования на языке GPSS (СПО GPSS/PC) [Электронное издание] / Научно-производственное объединение "ЦЕНТРПРОГРАММСИСТЕМ". — Калинин, 1989. — 183 с.