|  |
| --- |
| титул |
| **115172, Москва, ул. Б.Каменщики, д. 7;тел., факс: (495) 134 1234; e-mail: spo-54@edu.mos.ru** |

**МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ**

к лабораторной работе №7:

**Синтаксические конструкции. Блоки условия и цикла.**

по учебной дисциплине

**МДК 01.01 Разработка программных модулей**

Специальность

**09.02.07 «Информационные системы и программирование»**

**Оглавление**

[**Введение** 3](#_Toc148354931)

[**1.** **Программирование - синтаксические конструкции (блок условия)** 10](#_Toc148354932)

[**2.** **Программирование - синтаксические конструкции (блок цикла)** 17](#_Toc148354933)

[**Индивидуальное задание** 24](#_Toc148354934)

[**Контрольные вопросы** 25](#_Toc148354935)

# **Введение**

**Цель работы**: Изучение синтаксических конструкций 1С:Предприятие. Разработка блоков условия и цикла.

**Время выполнения**: 2 часа

**Задание**:

1. Законспектировать теоретические сведения, при наличии.

2. Выполнить лабораторную работу.

3. Выполнить индивидуальное задания, при наличии.

4. Защитить работу, ответив на контрольные вопросы устно.

**Операторы и синтаксические конструкции**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Описание | Синтаксис | Параметры | Пример |
| ? (вычислить выражение по условию) | Позволяет вычислить одно из двух заданных выражений в зависимости от результата вычисления логического выражения. | ?(<Логическое выражение>,   <Выражение 1>, <Выражение 2>) | Логическое выражение, результат вычисления которого определяет одно из результирующих выражений, которые будут вычислены. Если результат его вычисления *Истина*, то будет вычисляться <Выражение 1>. Если результат *Ложь*, то <Выражение 2>. | Статус = ?(ПолучитьСкидку() > 10, “Особый клиент”, “Обычный клиент”); Предупреждение(Статус); |
| ВызватьИсключение (Raise) | При использовании данной формы оператора вызывается новое исключение. | ВызватьИсключение <Выражение>  Англоязычный: Raise <Expression> | Результат вычисления выражения преобразуется к строке, и данная строка используется в качестве описания исключения. | ВызватьИсключение “Документ не может быть проведен” |
| Выполнить (Execute) | Позволяет выполнить фрагмент кода, который передается ему в качестве строкового значения. | Выполнить(<Строка>)  Англоязычный: Execute(<Строка>) | Строка, содержащая текст исполняемого кода. | // Выводит в окно сообщений текущую дату Выполнить(“Сообщить(ТекущаяДата())”); |
| ДобавитьОбработчик (AddHandler) | Добавляет обработчик события. При добавлении обработчика события производится проверка соответствия числа параметров события числу параметров метода, назначаемого в качестве обработчика. | ДобавитьОбработчик <Событие>, <ОбработчикСобытия>;  Англоязычный: AddHandler <Событие>, <ОбработчикСобытия>; | <Событие> — событие, которому добавляется обработчик. Событие задается в форме <Выражение>.<ИмяСобытия>, где: ● <Выражение> – произвольное выражение на встроенном языке. Его результатом должен быть объект, к событию которого добавляется обработчик. ● <ИмяСобытия> – идентификатор (имя) события. Процедура/функция-обработчик события. Обработчиком события может являться метод объекта встроенного языка. Тогда <ОбработчикСобытия> задается как <Выражение>.<ИмяОбработчика>, где: ● <Выражение> – произвольное выражение на встроенном языке. Его результатом должен быть объект, метод которого служит обработчиком события. ● <ИмяОбработчика> – имя метода обработчика события. Также в качестве обработчика события может быть задана процедура/функция, находящаяся в области видимости. В этом случае обработчик события задается как имя процедуры/функции. | Обработка = Обработки.КонтрольДокумента.Создать(); Накладная = Документы.Накладная.СоздатьДокумент(); ДобавитьОбработчик Накладная.ПриЗаписи, Обработка.ПриЗаписиДокумента; msword = Новый COMОбъект(“Word.Application”); ДобавитьОбработчик msword.DocumentChange, ПриИзмененииДокумента; Процедура ПриИзмененииДокумента() Сообщить(“Документ изменен”); КонецПроцедуры  Пример с разными параметрами: // Обработчик без параметров Процедура ОбработкаСобытия() КонецПроцедуры // Обработчик с одним параметром Процедура ОбработкаСобытия2(Параметр) КонецПроцедуры // Объект может генерировать события как с параметром, // так и без параметров Объект = Новый COMОбъект(“Test.Events”); ДобавитьОбработчик Объект.TestEvent, ОбработкаСобытия ДобавитьОбработчик Объект.TestEvent, ОбработкаСобытия2 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Описание | Синтаксис | Параметры | Пример |
| Для (For) | Оператор цикла Для предназначен для циклического повторения операторов, находящихся внутри конструкции  Цикл – КонецЦикла. | Для <Имя переменной> = <Выражение1> По <Выражение2>  Цикл // Операторы [Прервать;] // Операторы [Продолжить;] // Операторы КонецЦикла;  Англоязычный: For <Имя переменной> = <Выражение1> To <Выражение2>  Do // Операторы [Break;] // Операторы [Continue;] // Операторы EndDo; | <Имя переменной> – Идентификатор переменной (счетчика цикла), значение которой автоматически увеличивается на 1 при каждом повторении цикла. Так называемый счетчик цикла. <Выражение 1> – Числовое выражение, которое задает начальное значение, присваиваемое счетчику цикла при первом проходе цикла. По – Синтаксическая связка.  <Выражение 2> – Максимальное значение счетчика цикла. Когда переменная <Имя переменной> становится больше чем <Выражение 2>, выполнение оператора цикла Для прекращается. Цикл  –  операторы, следующие за ключевым словом Цикл, выполняются, пока значение переменной <Имя переменной> меньше или равно значению <Выражение 2>.  // Операторы Прервать – Позволяет прервать выполнение цикла в любой точке. После выполнения этого оператора управление передается оператору, следующему за ключевым словом КонецЦикла. Продолжить – Немедленно передает управление в начало цикла, где производится вычисление и проверка условий выполнения цикла. Операторы, следующие в теле. цикла за ним, на данной итерации обхода не выполняются. КонецЦикла – Ключевое слово, которое завершает структуру оператора цикла. | // Перебор дней текущего месяца ПоследнийДеньМесяца = День(КонецМесяца(ТекущаяДата())); Для ТекДень = 1 по ПоследнийДеньМесяца Цикл Состояние(“Обрабатывается день: “+ ТекДень); // Операторы обработки очередного дня месяца . . . КонецЦикла; |
| Для каждого (For each) | Оператор цикла Для каждого предназначен для циклического обхода коллекций значений.  При каждой итерации цикла возвращается новый элемент коллекции. Обход осуществляется до тех пор, пока не будут перебраны все элементы коллекции, или может быть завершен досрочно при выполнении оператора Прервать. | Для каждого <Имя переменной 1> Из <Имя переменной 2> Цикл // Операторы [Прервать;] // Операторы [Продолжить;] // Операторы КонецЦикла;  Англоязычный:  For each <Имя переменной1>  In <Имя переменной2> Do // Операторы [Break;] // Операторы [Continue;] // Операторы EndDo; | <Имя переменной> -Переменная, которой при каждом повторении цикла присваивается значение очередного элемента коллекции. Из – Синтаксическая связка для параметра <Имя переменной 2>. <Имя переменной 2> -Переменная или выражение, предоставляющее коллекцию. Элементы этой коллекции будут присваиваться параметру <Имя переменной 1>. Цикл – Операторы, следующие за ключевым словом Цикл, выполняются до тех пор, пока не будут перебраны все элементы коллекции.  // Операторы  Исполняемый оператор или последовательность таких операторов. Прервать – Позволяет прервать выполнение цикла в любой точке. После выполнения этого оператора управление передается оператору, следующему за ключевым словом КонецЦикла. Продолжить – Немедленно передает управление в начало цикла, где производится вычисление и проверка условий выполнения цикла. Операторы, следующие в теле цикла за ним, на данной итерации обхода не выполняются. КонецЦикла – Ключевое слово, которое завершает структуру оператора цикла. | // Перебор строк табличной части документа. Документ = Документы.РасходнаяНакладная.НайтиПоКоду(12345); // Проверим, найден нужный нам документ или нет Если Не Документ.Пустая() Тогда Для каждого СтрокаСостава из Документ.Состав Цикл Состояние(“Строка: ” + Документ.Состав.Индекс(СтрокаСостава)+1); // Операторы обработки очередной строки табличной части … КонецЦикла; КонецЕсли; |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Описание | Синтаксис | Параметры | Пример |
| Если (If) | Оператор Если управляет выполнением программы, основываясь на результаты одного или более логических выражений. Оператор может содержать любое количество групп операторов, возглавляемых конструкциями  ИначеЕсли – Тогда. | Если <Логическое выражение> Тогда // Операторы [ИначеЕсли <Логическое выражение> Тогда] // Операторы [Иначе] // Операторы КонецЕсли;  Англоязычный синтаксис: If <Логическое выражение> Then // Операторы [ElsIf <Логическое выражение> Then] // Операторы  [Else] // Операторы EndIf; | <Логическое выражение> – Логическое выражение.  Тогда – Операторы, следующие за Тогда, выполняются, если результатом логического выражения является значение Истина.  // Операторы – Исполняемый оператор или последовательность таких операторов  ИначеЕсли – Логическое выражение, следующее за ключевым словом ИначеЕсли, вычисляется только тогда, когда условия в Если и всех предшествующих ИначеЕсли оказались равны Ложь. Операторы, следующие за конструкцией ИначеЕсли – Тогда, выполняются, если результат логического выражения в данном ИначеЕсли равен Истина.  Иначе – Операторы, следующие за ключевым словом Иначе, выполняются, если результаты логических выражений в конструкции Если и всех предшествующих конструкциях ИначеЕсли оказались равны Ложь.  КонецЕсли–  Ключевое слово, которое завершает структуру оператора условного выполнения. | Если ДеньНедели(ТекущаяДата()) = 6 Тогда Сообщить(“Сегодня суббота.”); ИначеЕсли ДеньНедели(ТекущаяДата()) = 7 Тогда Сообщить(“Сегодня воскресенье.”); Иначе Сообщить(“Сегодня рабочий день.”); КонецЕсли; |
| Новый (New) | Оператор позволяет создать значение указанного типа. Допустим только для тех типов, для которых разрешено создание новых значений. Для прикладных объектов необходимо использовать функциональную форму оператора **Новый** (вариант 2), так как при проверке модулей в конфигураторе  не определены типы для прикладных объектов. | Синтаксис (вариант 1): Новый <Имя типа>[(<Парам 1>, …, <Парам N>)]  Синтаксис (вариант 2): Новый (<Тип>[, <Параметры конструктора>]) | Параметры 1:  Имя типа – Указывается имя типа, значение которого создается.  <Парам 1>, …, <Парам N> –  После имени типа в скобках могут указываться параметры, если они определены в конструкторах для данного типа. Допустимое количество параметров и их назначение указываются в описании конструкторов объекта.  Параметры 2:  Тип – Имя типа или значение типа Тип. <Параметры конструктора> Массив параметров конструктора. | Пример 1: // Пример создания массива из трех элементов. Массив = Новый Массив(3);  Пример 2:  ТипЗначения = Тип(“КвалификаторыСтроки”); Параметры = Новый Массив(2); Параметры[0] = 20; Параметры[1] = ДопустимаяДлина.Переменная; КвалифСтр = Новый(ТипЗначения, Параметры); |
| Перейти (Goto) | Безусловная передача управления на другой оператор программы. Передает управление от одного оператора к другому. Область действия оператора ограничивается программным модулем, процедурой или функцией; он не может передать управление за пределы программного модуля, процедуры или функции. | Перейти <Метка>; | Примечание 1. Метка в этом операторе не должна быть меткой перехода на оператор Процедура или Функция. Примечание 2. Оператор безусловного перехода не может быть использован для передачи управления на операторы, находящиеся внутри конструкций: Пока – КонецЦикла, Для – КонецЦикла, Для каждого – КонецЦикла, Если – КонецЕсли, Попытка – Исключение – КонецПопытки извне этих конструкций. | Перейти ~Метка1; … ~ Метка1: Сообщить(“Осуществлен переход по метке.”); |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Описание | Синтаксис | Параметры | Пример |
| Перем (Var) | Позволяет в явном виде объявить переменную. | Перем <Имя переменной 1> [Экспорт] [, <Имя переменной 2>, …]; Англоязычный: Var <Имя переменной 1> [Export] [, <Имя переменной 2>, …]; | <Имя переменной 1>[, <Имя переменной 2>, …] – Задается имя или имена объявляемых переменных Экспорт – Необязательное ключевое слово. Указывает, что данная переменная доступна при обращении к контексту этого модуля из других модулей. Данное ключевое слово необходимо указывать для каждой объявляемой переменной отдельно. Не имеет смысла при объявлении переменных отдельных процедур или функций. | // Пример объявления одной переменной Перем А Экспорт; Перем Б; // Пример объявления нескольких переменных одним оператором Перем А, Б Экспорт; |
| Пока (While) | Оператор цикла Пока предназначен для циклического повторения операторов, находящихся внутри конструкции Цикл – КонецЦикла. Цикл выполняется,  пока логическое выражение равно Истина. Условие выполнения цикла всегда проверяется вначале, перед выполнением цикла. | Пока <Логическое выражение> Цикл // Операторы [Прервать;] // Операторы [Продолжить;] // Операторы КонецЦикла Англоязычный: While <Логическое выражение> Do // Операторы [Break;] // Операторы [Continue;] // Операторы EndDo; | <Логическое выражение> -Логическое выражение Цикл– Операторы, следующие за ключевым словом Цикл, выполняются, пока результат логического выражения равен Истина. // Операторы – Исполняемый оператор или последовательность таких операторов. Прервать – Позволяет прервать выполнение цикла в любой точке. После выполнения этого оператора управление передается оператору, следующему за ключевым словом КонецЦикла. Продолжить –  Немедленно передает управление в начало цикла, где производится вычисление и проверка условий выполнения цикла. Операторы, следующие в теле цикла за ним, на данной итерации обхода не выполняются. КонецЦикла –  Ключевое слово, которое завершает структуру оператора цикла. | ВыборкаДок = Документы.РасходнаяНакладная.Выбрать(); // Цикл по всем документам Пока ВыборкаДок.Следующий() Цикл // Отобразим Документ в панели состояния Состояние(“Обрабатывается документ №” + ВыборкаДок.Номер); // Операторы выполнения действий над документом КонецЦикла; |
| Попытка (Try) | Оператор Попытка управляет выполнением программы, основываясь на возникающих при выполнении модуля ошибочных (исключительных) ситуациях, и определяет обработку этих ситуаций. В качестве ошибочных (исключительных) ситуаций воспринимаются ошибки времени выполнения модуля. Не предусмотрено определяемых пользователем исключений. | Попытка // Операторы попытки Исключение // Операторы исключения [ВызватьИсключение;] // Операторы исключения КонецПопытки; Англоязычный синтаксис: Try // Операторы попытки Except // Операторы исключения [Raise;] // Операторы исключения EndTry; | // Операторы попытки – Исполняемый оператор или последовательность таких операторов.  Исключение – Операторы, следующие за ключевым словом Исключение, выполняются, если при выполнении последовательности операторов произошла ошибка времени выполнения.  // Операторы исключения – Исполняемый оператор или последовательность операторов, которые обрабатывают исключительную ситуацию.  ВызватьИсключение –  Оператор позволяет вызвать исключение в тех случаях, когда, несмотря на отработку исключительной ситуации, необходимо прервать выполнение модуля с ошибкой времени выполнения. Оператор допустим только внутри операторных скобок Исключение – КонецПопытки. Выполнение данного оператора прекращает выполнение последовательности операторов исключения, и производится поиск более «внешнего» обработчика исключения (при вложенных попытках). Если таковой есть, то управление передается на его первый оператор. Если нет, то исключительная ситуация обрабатывается системно, выдается сообщение о первоначально возникшей ошибке, а выполнение модуля прекращается.  КонецПопытки–  Ключевое слово, которое завершает структуру оператора обработки исключительных ситуаций. | Процедура СформироватьВExcel() Попытка // Пытаемся обратиться к программе MS Excel Табл = Новый ComObject(“Excel.Application”); Исключение Предупреждение(ОписаниеОшибки()); Возврат; КонецПопытки; // Операторы формирования отчета … КонецПроцедуры |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Описание | Синтаксис | Параметры | Пример |
| Процедура (Procedure) | Ключевое слово Процедура начинает секцию исходного текста, выполнение которого можно инициировать из любой точки программного модуля, просто указав ИмяПроцедуры() со списком параметров (если параметры не передаются, то круглые скобки, тем не менее, обязательны). Если в модуле приложения или общем программном модуле в теле описания процедуры использовано ключевое слово Экспорт, то это означает, что данная процедура является доступной из всех других программных модулей конфигурации. При выполнении оператора Возврат процедура заканчивается и возвращает управление в точку вызова. Если в тексте процедуры не встретился оператор Возврат, то после выполнения последнего исполняемого оператора происходит выполнение неявного оператора Возврат. Конец программной секции процедуры определяется по оператору КонецПроцедуры | Процедура <ИмяПроцедуры>([[Знач] <Парам 1> [=<ДефЗнач>], … ,[Знач] <Парам N> [=<ДефЗнач>]])[Экспорт] // Объявления локальных переменных; // Операторы; … [Возврат;] // Операторы; … КонецПроцедуры  Англоязычный: Procedure <ИмяПроцедуры>([[Val] <Парам 1> [=<ДефЗнач>], … ,[Val] <Парам N>[=<ДефЗнач>]])[Export] // Объявления локальных переменных; // Операторы; … [Return;] // Операторы; … EndProcedure | Переменные, объявленные в теле процедуры в разделе Объявления локальных переменных, являются локальными переменными данной процедуры, поэтому доступны только в этой процедуре (за исключением случая передачи их как параметров при вызове других процедур, функций или методов). ПРИМЕЧАНИЕ. Ключевые слова Процедура, КонецПроцедуры являются не операторами, а операторными скобками, поэтому не должны заканчиваться точкой с запятой (это может приводить к ошибкам выполнения модуля). | Перем Глоб; // Описание процедуры Процедура МояПроцедура(Пар1, Пар2, Пар3) Экспорт Глоб = Глоб + Пар1 + Пар2 + Пар3; Возврат; КонецПроцедуры Глоб = 123; МояПроцедура(5, 6, 7); // Вызов процедуры |
| УдалитьОбработчик (RemoveHandler) | Удаляет обработчик события. При удалении обработчика события производится проверка соответствия числа параметров события числу параметров метода, назначенного в качестве обработчика. | Синтаксис: УдалитьОбработчик <Событие>, <ОбработчикСобытия>;  Англоязычный: RemoveHandler <Событие>, <ОбработчикСобытия>; | <Событие> – обработчик события удаляется. Событие задается в форме <Выражение>.<ИмяСобытия>, где: ● <Выражение> – произвольное выражение на встроенном языке. Его результатом должен быть объект, обработчик события которого удаляется. ● <ИмяСобытия> – идентификатор (имя) события.  <ОбработчикСобытия> – Процедура/функция-обработчик события. Обработчиком события может являться метод объекта встроенного языка. Тогда <ОбработчикСобытия> задается как <Выражение>.<ИмяОбработчика>, где: ● <Выражение> – произвольное выражение на встроенном языке. Его результатом должен быть объект, метод которого служит обработчиком события. ● <ИмяОбработчика> – имя метода обработчика события. Также в качестве обработчика события может быть задана процедура/функция, находящаяся в области видимости. В этом случае обработчик события задается как имя процедуры/функции. | УдалитьОбработчик Накладная.ПриЗаписи, Обработка.ПриЗаписиДокумента; |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Описание | Синтаксис | Параметры | Пример |
| Функция (Function) | Ключевое слово Функция начинает секцию исходного текста функции, выполнение которой можно инициировать из любой точки программного модуля, просто указав ИмяФункции со списком параметров (если параметры не передаются, то круглые скобки, тем не менее, обязательны). Если в модуле приложения или общем программном модуле в теле описания функции использовано ключевое слово Экспорт, то это означает, что данная функция является доступной из всех других программных модулей конфигурации. | Функция <ИмяФункции>([[Знач] <Парам 1>[=<ДефЗнач>], … ,[Знач] <Парам N>[=<ДефЗнач>]])[Экспорт] // Объявления локальных переменных; // Операторы; … Возврат <Возвращаемое значение>; // Операторы; … КонецФункции  Англоязычный: Function <ИмяФункции>([[Val] <Парам 1>[=<ДефЗнач>], … ,[Val] <Парам N>[=<ДефЗнач>]])[Export] // Объявления локальных переменных; // Операторы; … Return <Возвращаемое значение>; // Операторы; … EndFunction | Выполнение функции заканчивается оператором Возврат. Функции отличаются от процедур только тем, что возвращают ВозвращаемоеЗначение. Конец программной секции функции определяется по оператору КонецФункции. Вызов любой функции в тексте программного модуля можно записывать как вызов процедуры, т. е. в языке допускается не принимать от функции возвращаемое значение. Если ключевое слово Возврат в теле функции не указано или строка модуля, его содержащая, не выполнена, то функция возвращает значение типа Неопределено. Переменные, объявленные в теле функции в разделе Объявления локальных переменных, являются локальными переменными данной функции, поэтому доступны только в этой функции (за исключением случая передачи их как параметров при вызове других процедур, функций или методов). ПРИМЕЧАНИЕ. Ключевые слова Функция, КонецФункции являются не операторами, а операторными скобками, поэтому не должны заканчиваться точкой с запятой (это может приводить к ошибкам выполнения модуля). | Перем Глоб; // Описание функции Функция МояФункция(Пар1, Пар2, Пар3) Экспорт Глоб = Глоб + Пар1 + Пар2 + Пар3; Возврат Глоб; КонецФункции Глоб = 123; Рез = МояФункция(5, 6, 7);  // Вызов функции |

# **Программирование - синтаксические конструкции (блок условия)**

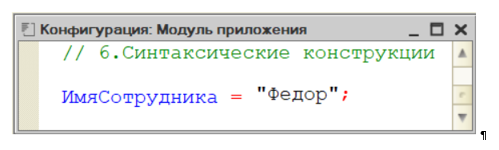
Теперь рассмотрим такую важную часть написания программного кода, как «Синтаксические конструкции».

Вообще синтактических конструкций достаточно много в программировании на любом языке. Самые ключевые синтаксические конструкции: условия, циклы ‚ функции и процедуры. Есть. также и менее популярные конструкции, например, попытка и исключение.

Основное назначение синтактических конструкций заключается в том, чтобы каким-то образом менять логику исполняемого алгоритма, например, разветвлять ее.

Все алгоритмы, которые мы писали до этого, были линейные, то есть они исполнялись строчка за строчкой.

Бывают такие ситуации, когда нужно в зависимости от входных данных поступать тем или иным образом. Такие алгоритмы называются разветвленны ми: появляется развилка на «Да» и «Нет».Создадим переменную «ИмяСотрудника» и присвоим значение «Федор».



Задача: в зависимости от имени сотрудника здороваться с ним по-разному.

Для таких целей существует синтактическая конструкция, которая называется «блок условия». Это первая синтактическая конструкция, с которой мы познакомимся.

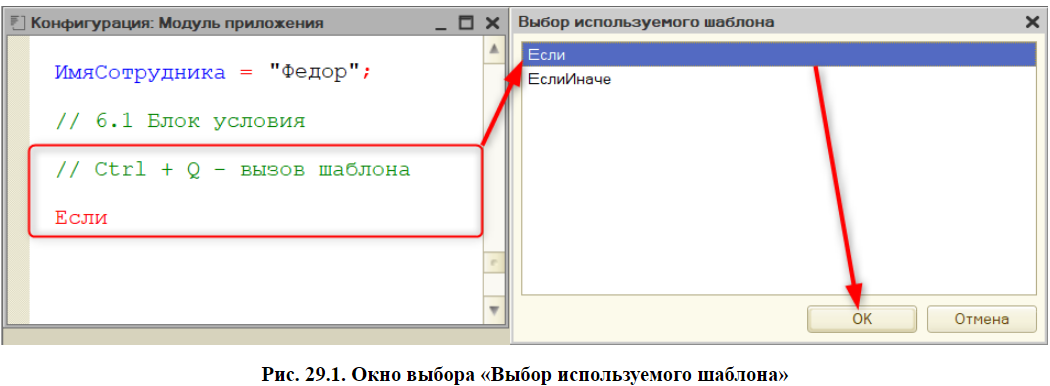
Блок условия позволяет проверить с помощью выражения наличие той или иной информации в алгоритме.

Мы можем проверить переменную, содержащую имя сотрудника, на предмет соответствия значению, например, «Василий».

Выражение сравнения вернет нам тип «Булево» («Истина» или «Ложь»). То есть блок условия всегда будет разветвлять нашу информацию по принципу «истина или ложь».

Для написания ‘условной конструкции воспользуемся специальным шаблоном, который можно вызвать, написав «Если» и нажав на комбинацию клавиш «CTRL + Q».

При нажатии на эту комбинацию клавиш у нас открывается окно «Выбор используемого шаблона» (рис. 29.1).

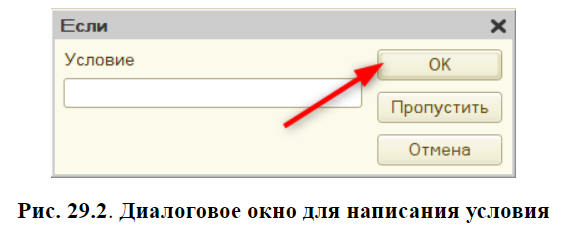


**Рисунок 29.1 Окно выбора «Выбор используемого шаблона»**

Шаблоны работают не всегда, а только если ввести ключевую фразу.

В данном случае мы написали ключевое слово «Если» и нажали комбинацию клавиш — открылось окно выбора используемого шаблона. Выберем первую конструкцию.

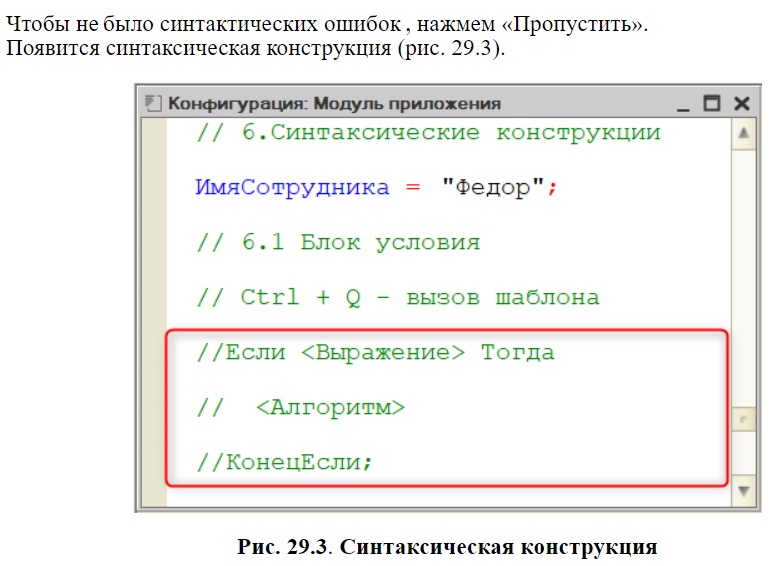
Выбрав нужный вариант, нажмем «Enter», либо кнопку «ОК». Откроется следующее диалоговое окно (рис. 29.2), которое предлагает написать условие (в качестве условия принимается выражение).



**Рисунок 29.2 Диалоговое окно для написания условия**

Чтобы не было синтактических ошибок ‚ нажмем «Пропустить».

Появится синтаксическая конструкция (рис. 29.3).



**Рисунок 29.3 Синтаксическая конструкция**

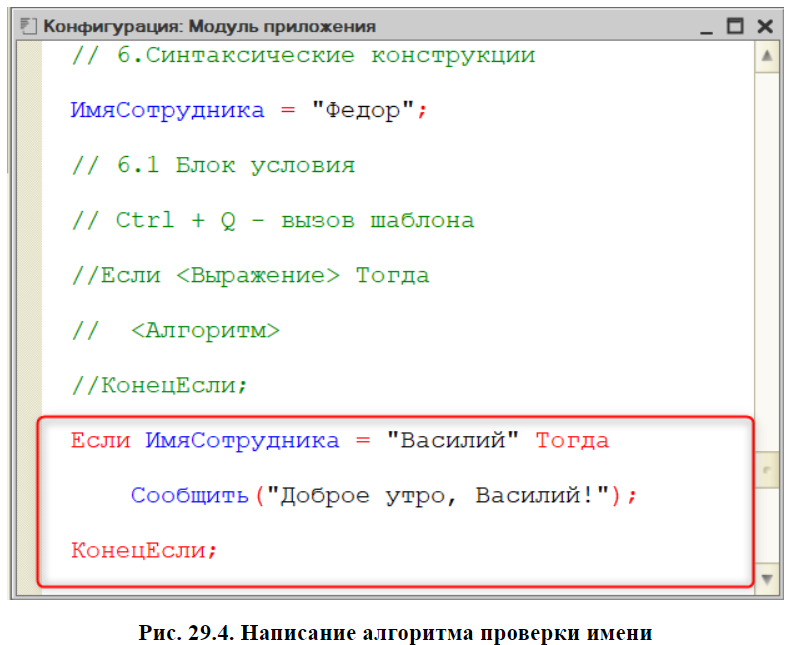
После ключевого слова «Если» указывается выражение, возвращающее тип «Булево».

Далее идет алгоритм, который выполнится, если выражение в условии принимает значение «Истина». Если это выражение возвращает «Ложь», то алгоритм не выполнится.

Попробуем описать такой алгоритм. Напишем еще раз синтаксическую конструкцию. Проверим значение нашей переменной на соответствие значению «Василий».

После ключевого слова «Если» напишем «ИмяСотрудника = “Василий”».

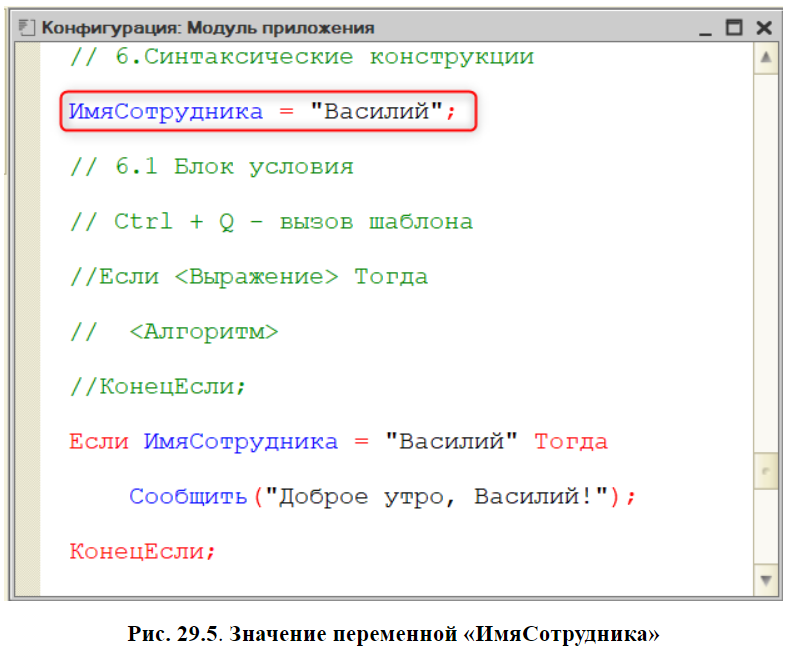
Если условие выполняется, тогда выполнится команда «Сообщить (“Доброе утро , Василий!”)» (рис. 29.4).



**Рисунок 29.4 Написание алгоритма проверки имени**

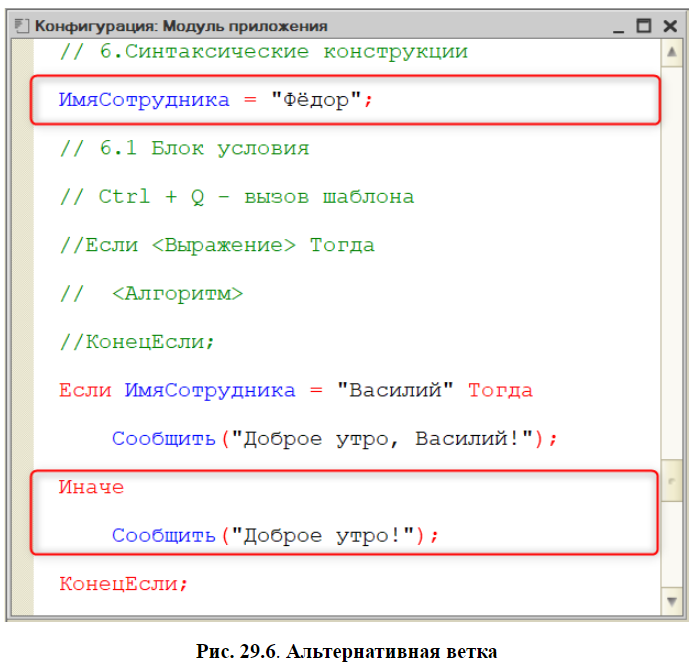
Обновим конфигурацию и запустим пользовательский режим. Вывод сообщения отсутствует: прописанное условие не выполнилось.

Значение переменной «ИмяСотрудника» равно не «Василий», а «Федор» (рис. 29.5).



**Рисунок 29.5 Значение переменной «ИмяСотрудника»**

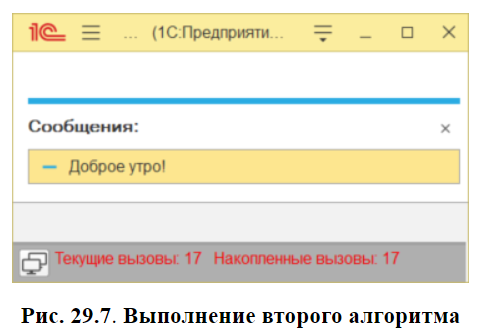
Усложним алгоритм: напишем альтернативно ветку, которая будет выполняться всегда, когда не срабатывает основное условие. За это отвечает блок «Иначе» (рис. 29.6).



**Рисунок 29.6 Альтернативная ветка**

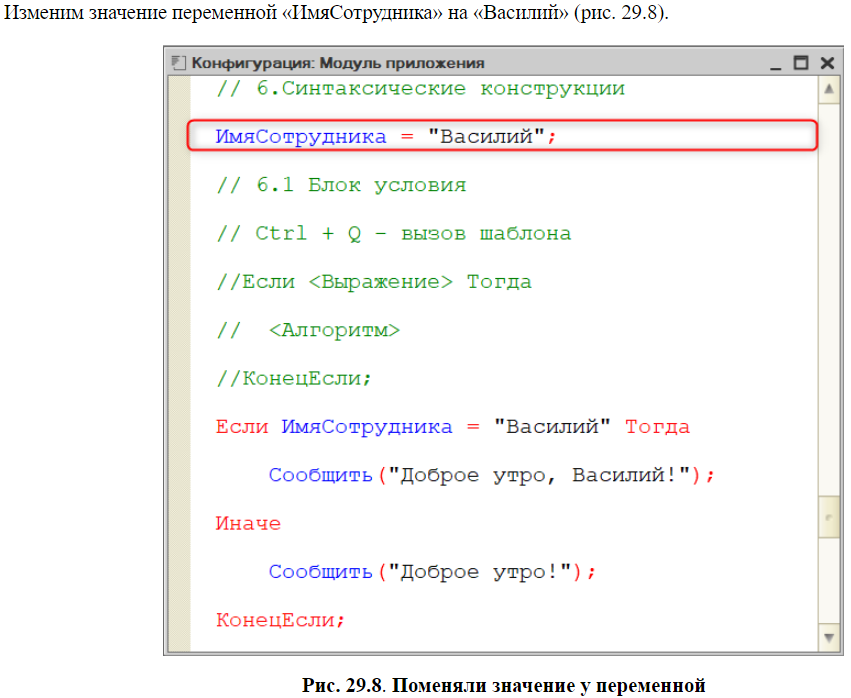
Если основное условие не выполняется, то будет в любом случае срабатывать алгоритм «Сообщить “Доброе утро!”)».

Обновим конфигурацию и запустим пользовательский режим. Выводится сообщение «Доброе утро!» — выполнялся алгоритм, идущий после конструкции «Иначе» (рис. 29.7).



**Рисунок 29.7 Выполнение второго алгоритма**

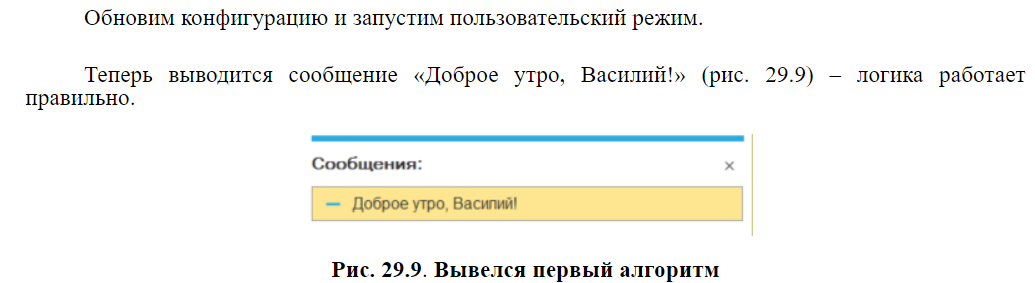
Изменим значение переменной «ИмяСотрудника» на «Василий» (рис. 29.8)



**Рисунок 29.8 Поменяли значение у переменной**

Обновим конфигурацию и запустим пользовательский режим.

Теперь выводится сообщение «Доброе утро, Василий!» (рис. 29.9)-логика работает правильно



**Рисунок 29.9 Вывелся первый алгоритм**

Так как сработало условие, то у нас вывелся первый алгоритм.

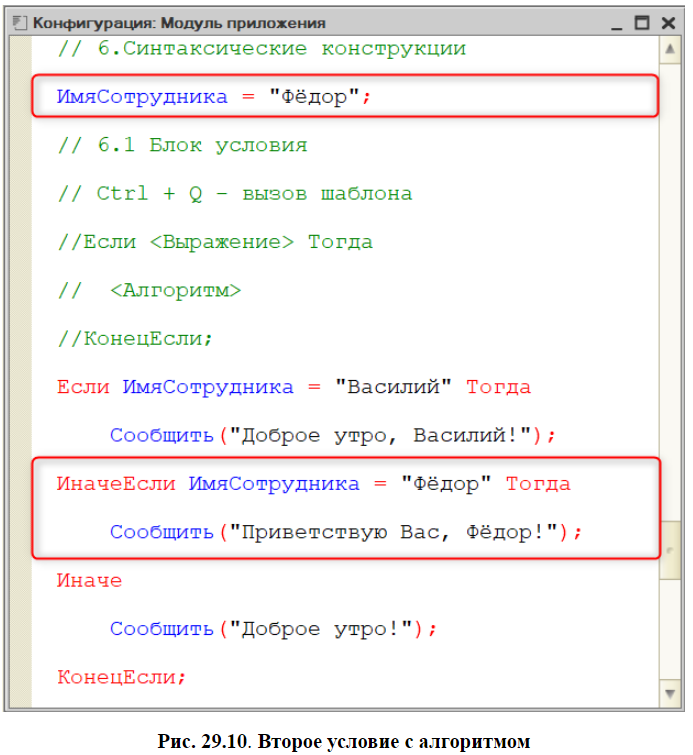
Бывают ситуации, когда одного условия недостаточно, и нужно проверять сразу несколько условий.

Например, если не выполнилось первое условие, то выполняется второе. Если не выполнилось второе - то третье ит. д.

Для этого предназначен блок «ИначеЕсли», и этот блок работает по такому же принципу, как и блок «Если».

Если ни один из блоков не срабатывает, тогда выполняется блок «Иначе».

Опишем такую конструкцию: первым условием будем проверять имя сотрудника на соответствие значению «Василий». Если это не так, тогда произойдет вторая проверка: на соответствие значению «Фёдор». Если и второе условие не выполнится, тогда отработает ветка «Иначе» (рис. 29.10).

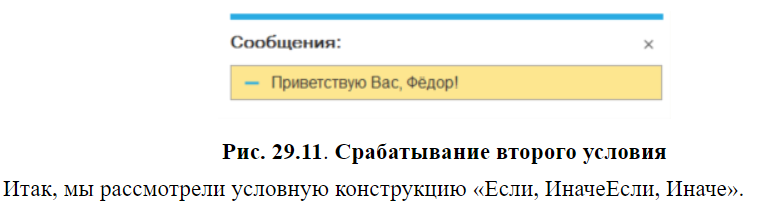


**Рисунок 29.10 Второе условие с алгоритмом**

Обновим конфигурацию и запустим пользовательский режим.

Так как значение переменной не поменялось, то сработает первое условие.

Если изменить значение переменной на «Фёдор», то сработает второе условие и выполнится алгоритм приветствия Фёдора (рис. 29.11).



**Рисунок 29.11 Срабатывание второго условия**

Итак, мы рассмотрели условную конструкцию «Если, ИначеЕсли, Иначе».

# **Программирование - синтаксические конструкции (блок цикла)**

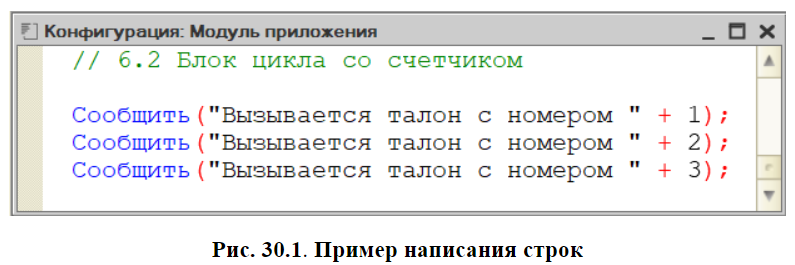
Очень часто возникает такая ситуация, что нужно в алгоритме итеративно выполнять какой- то порядок действий.

Рассмотрим ситуацию: электронная очередь.

По голосовой связи сообщается номер талона, человек с которым должен подойти к тому или иному окну. Значение номера талона должно увеличиваться в зависимости от того, как двигается очередь.

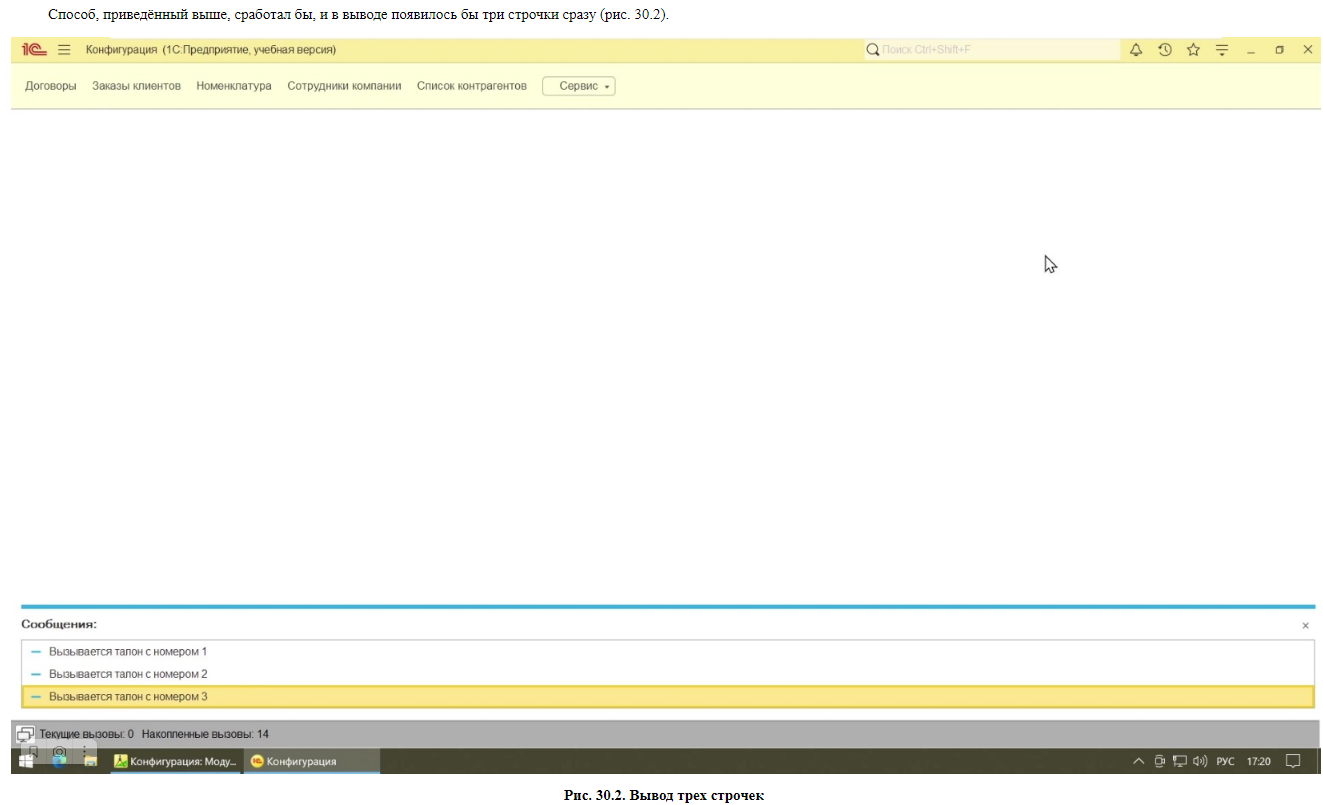
Предположим, что должно выводиться сообщение: «Вызывается талон с номером №.

Получается, что нам нужно сделать вызов талонов последовательно. Пока, не зная циклов, можно было бы сделать так: прописать три строки с разными цифрами (рис. 30.1).

****

**Рисунок 30.1 Пример написания строк**

Способ, приведенный выше, сработал бы, и в выводе появилось бы три строчки сразу (рис. 30.2).

****

**Рисунок 30.2 Вывод трех строчек**

Все усложняется тем, что таких выводов для каждого талона может быть много, или мы вовсе можем не знать их конечное количество.

Поэтому правильно будет воспользоваться синтаксической конструкцией «Циклы».

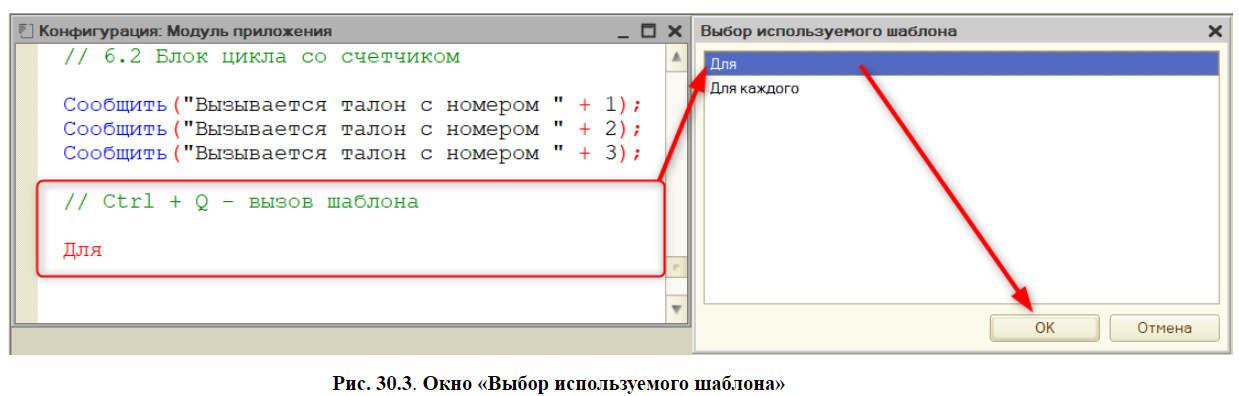
В 1С есть всего три типа циклов. Мы рассмотрим два из них: со счетчиком и с условием.

Начнем со счетчика. Цикл начинается со слов «Для» по аналогии из других языков программирования «For».

Введем это ключевое слово и с помощью комбинации клавиш «СTRL+ Q» вызовем окно «Выбор используемого шаблона».

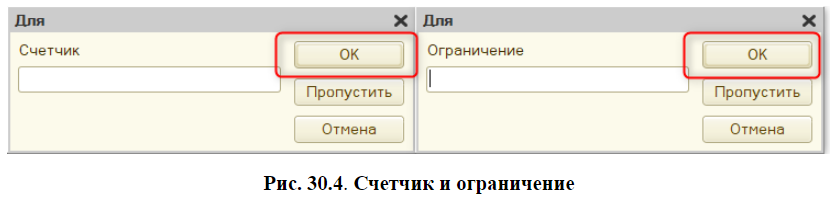
Перед нами появятся два варианта: «Для» и «Для каждого» (рис. 30.3).

Сейчас мы рассмотрим цикл «Для»: цикл со счетчиком.

****

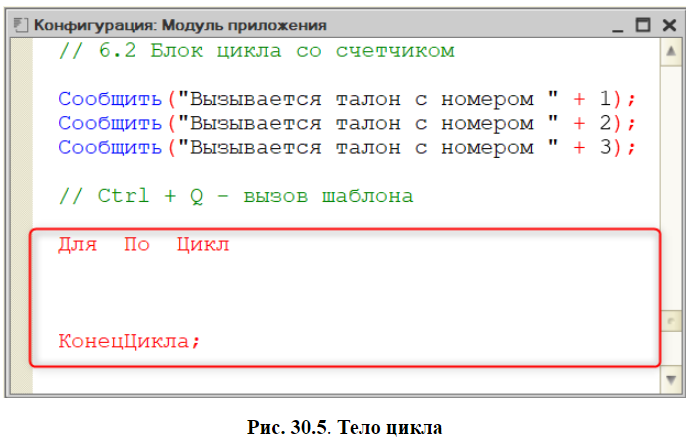
**Рисунок 30.3 Окно «Выбор используемого шаблона»**

После выбора варианта появляется окно ввода счетчика (рис. 30.4). Игнорируем это поле и нажимаем «Пропустить». «Ограничение» счетчика также пропустим — укажем его в модуле с использованием контекстной подсказки (рис. 30.4).

****

**Рисунок 30.4 Счетчик и ограничение**

В результате мы получили заготовку – тело нашего цикла (рис. 30.5)

****

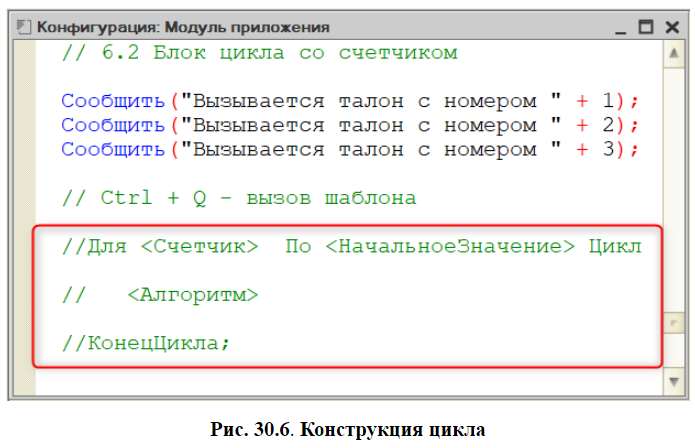
**Рисунок 30.5 Тело цикла**

Наш алгоритм должен выполняться итеративно, то есть несколько раз.

Для начала создадим счетчик после конструкции «Для» и присвоим ему начальное значение.

После ключевого слова «По» мы укажем конечное значение — до какого числа увеличиваться нашему счетчику.

Получится следующая конструкция (рис. 30.6).

****

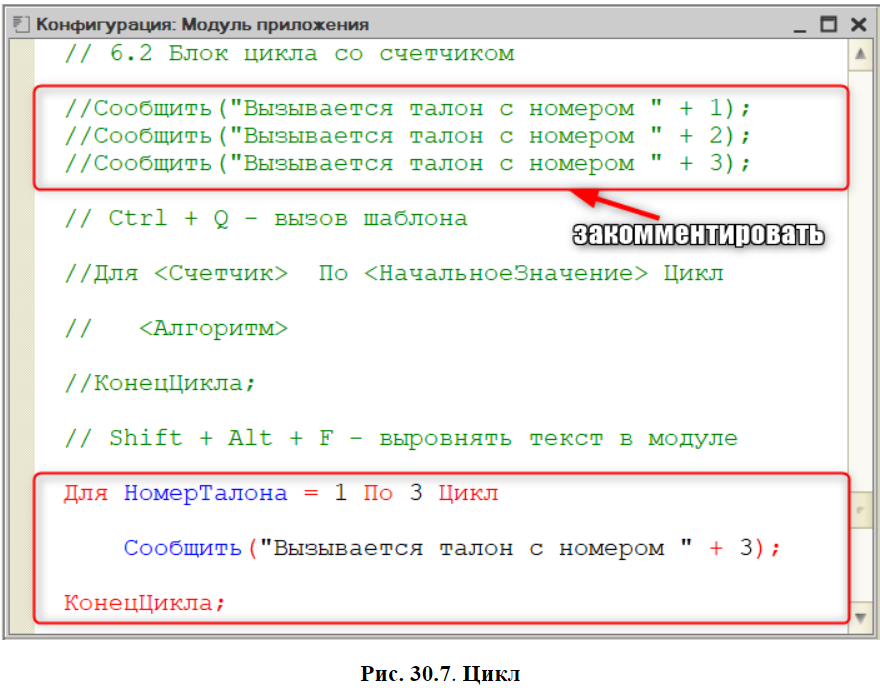
**Рисунок 30.6 Конструкция цикла**

Далее приступим к формированию нашего цикла. Счетчик нужно как-то назвать — это переменная, которая существует только в рамках цикла. За рамками цикла этой переменой не будет. Поскольку у нас речь идет о номере талона, назовем эту переменную «Номер Талона» и зададим начальное значение. В нашем случае очередь начинается с первого талона, значит, ставим единицу.

Дальше нужно прописать конечное значение. В нашем случае это три.

В итоге алгоритм будет выполняться столько раз, сколько указано шагов в счетчике цикла. И на каждой итерации счетчик автоматически будет увеличиваться ровно на единицу.

Пропишем алгоритм (рис. 30.7).

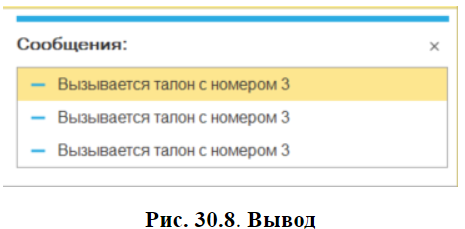
****

**Рисунок 30.7 Цикл**

Для удобства чтения принято писать строчки кода внутри синтаксических конструкций со сдвигом на один отступ табуляции. Сделать такой отступ можно, нажав на клавишу «Таb».

Произвести выравнивание можно сразу по нескольким строкам кода: для этого необходимо и выделить, и нажать на сочетание клавиш «SHIFT+ALT+F». Тогда произойдет автоматическое форматирование программного кода.

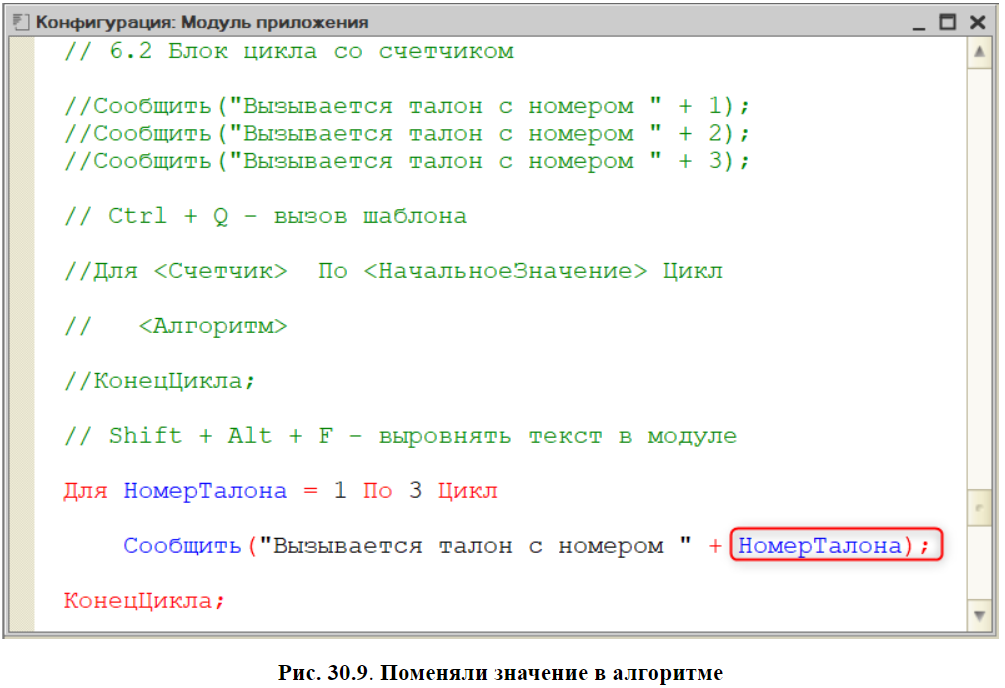
Проверим выполнение написанного алгоритма в пользовательском режиме (рис. 30.8).

****

**Рисунок 30.8 Вывод**

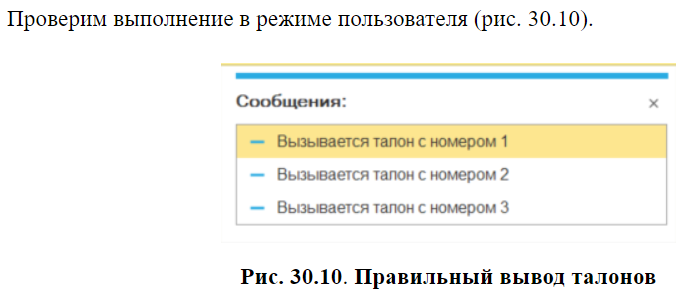
Так как в алгоритме цикла в сообщении указано фиксированное значение «три», то в пользовательском режиме так и выводится.

Нам нужно заменить фиксированное значение номера талона на значение, которое будет меняться с каждым шагом цикла. Такое значение у нас уже есть — это переменная «НомерТалона» (рис. 30.9).

****

**Рисунок 30.9 Поменяли значение в алгоритме**

Проверим выполнение в режиме пользователя (рис. 30.10)

****

**Рисунок 30.10 Правильный вывод талонов**

Рассмотрим еще одну синтаксическую конструкцию цикла: цикл с условием.

Предположим, у нас есть десять конфет. Требуется выполнять действия до тех пор, пока не останется 0 конфет. Когда они закончатся, появится сообщение «Все конфеты съели».

Для таких ситуаций есть цикл «Пока».

Для начала создадим переменную «ОсталосьКонфет» и присвоим ей значение 10.

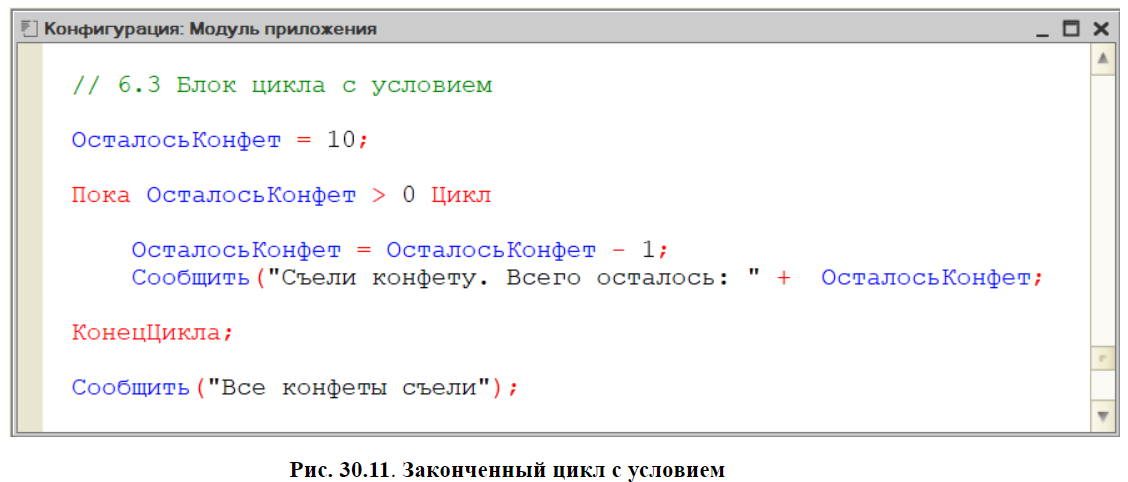
Затем напишем ключевое слово «Пока» и воспользуемся комбинацией клавиш «CTRL+Q» для вызова шаблона.

Идея такого цикла заключается в том, что действия выполняются до тех пор, пока выполняется условие: если условие не выполняется — цикл заканчивается.

***Важно!***

***Если в условии цикла указано выражение, всегда принимающее значение «Истина», то такой цикл будет выполняться бесконечно, и компьютер зависнет.***

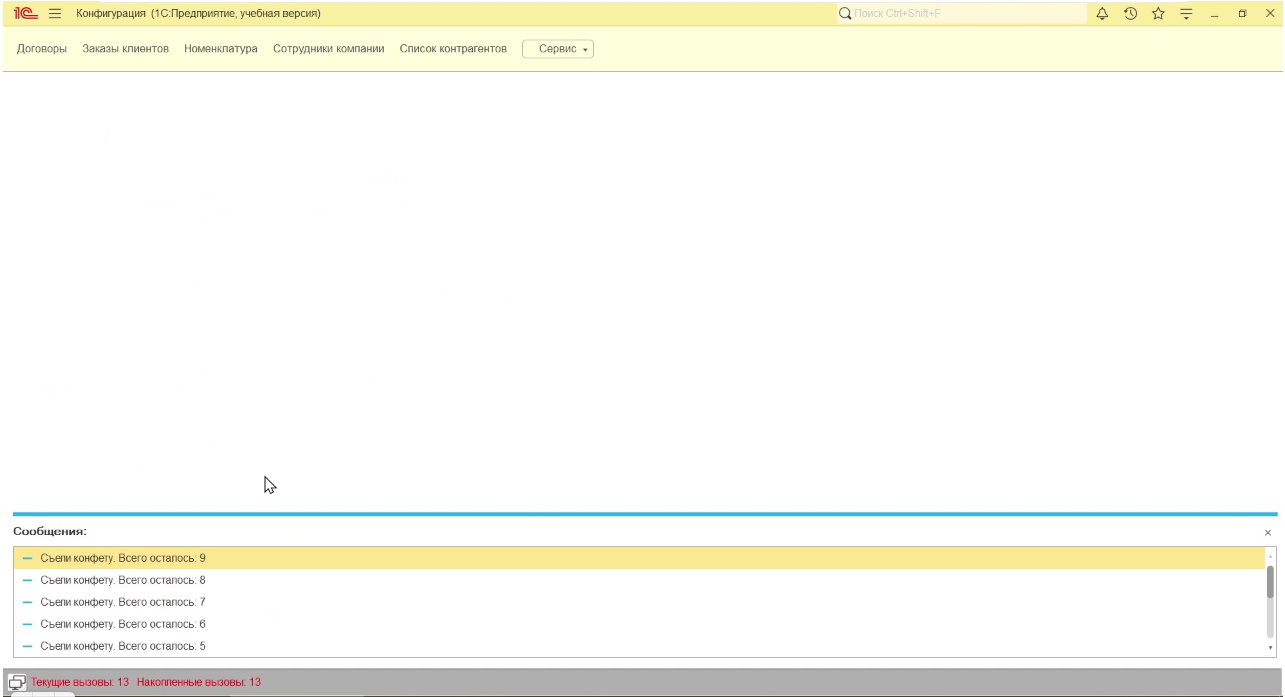
Чтобы цикл имел конечное количество итераций, пропишем уменьшение конфет в алгоритме цикла. Также будем выводить сообщение о количестве оставшихся конфет (рис. 30.11).

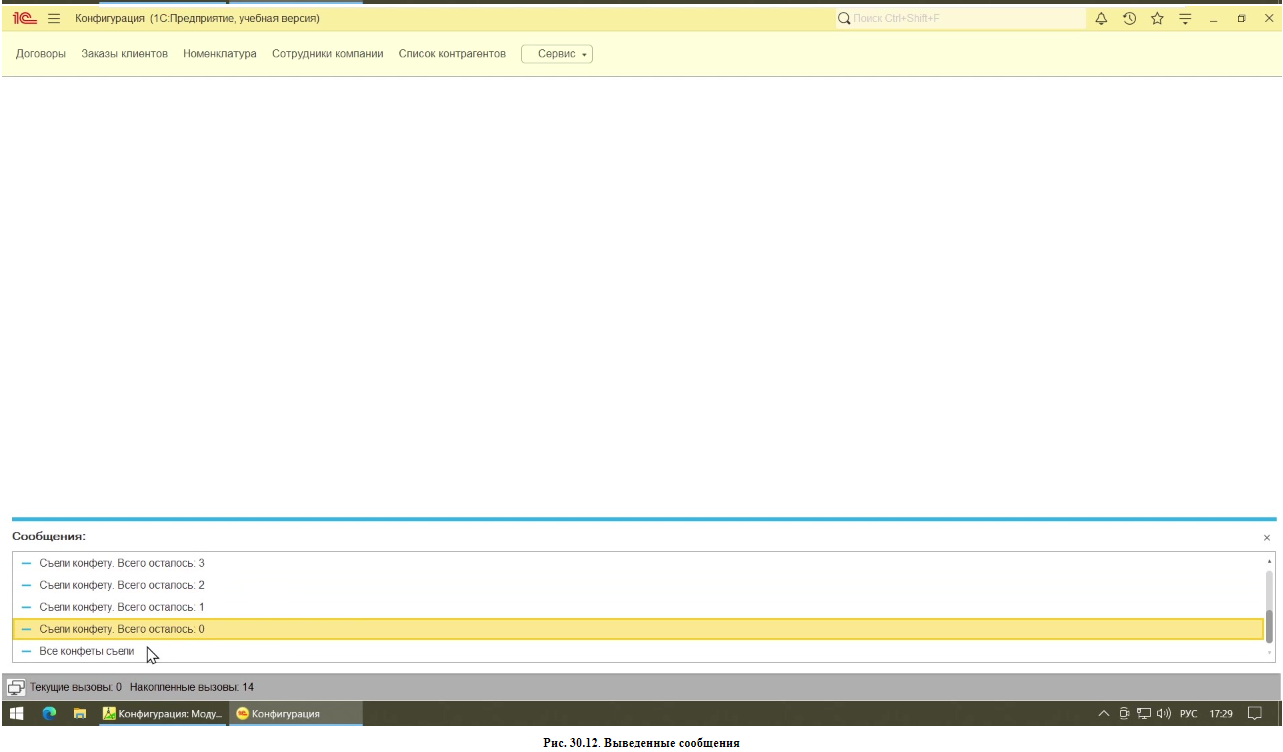
****

**Рисунок 30.11 Законченный цикл с условием**

Цикл со счетчиком закомментируем, чтобы он нам не мешал. Обновим конфигурацию и запустим пользовательский режим.

Алгоритм работает, и все сообщения выводятся корректно (рис. 30.12).



****

**Рисунок 30.12 Выведенные сообщения**

Цикл с условием встречается не так часто, как цикл «Для каждого» или цикл со счетчиком. Тем не менее он используется в логике.

# **Индивидуальное задание**

**№ 12 Алгоритмы – «Сколько времени?»**

Необходимо определить 1 переменную:

1. Текущий день – текущее время на компьютере.

Просчитать количество выходных дней, которые прошли с начала года (учитывать только субботу и воскресенье).

**Пример:**

Текущий день = 01.01.2023 00:00:00

«С начала года прошло 24 выходных дня».

Примечание: для расчета нужно использовать функции для работы с типом Дата, например НачалоДня(), НачалоГода(), НеделяГода().

**№ 13 Алгоритмы – «Угадай число»**

Вася и Петя решили сыграть с игру «угадай число». Один из игроков загадывает число в пределах от 0 до 1000, а другой должен его угадать.

Если второй игрок назвал число, которое близко к загадываемому в пределах 10, значит второй игрок победил. В ином случае побеждает первый игрок.

Если первый игрок загадал число не в заданном пределе, то первый игрок автоматически проигрывает.

**!Все действия должны сопровождать информационным сообщением!**

**№ 14 Алгоритмы – «Здравствуйте, очень приятно!»**

Компания по автоматизации роботов просила заказ на олгоритм для роборта, который стоит в фойе гостиницы.

По этому алгоритму робот должен приветствовать каждого клиента и считать при этом количество человек, вошедших в гостиницу.

Если человек, который прошел через дверь является 100, 1 000, 10 000, то должно звучать оповещение и поздравление, что этот клиент получает возможность поселиться в номере ЛЮКС БЕСПЛАТНО!

# **Контрольные вопросы**

1. Что произойдет с пользовательским режимом при выполнении бесконечного цикла «Пока»?
2. Опишите конструкцию цикла «Для».