Глава 3. Основные операторы

Третья глава у нас очень объемная по знаниям, в ней мы научимся работать с конструкцией *Если*, разберем два вида цикла, будем пробовать делать собственные процедуры и функции и освоим такой нужный оператор как *Попытка*.

Часть 1. Условие: Если…Тогда…ИначеЕсли

Начнем мы с очень необходимого оператора, без которого просто никуда. Это оператор условия ***Если…Тогда.***

Данный оператор управляет выполнением программы в зависимости от логического выражения (их может быть одно или несколько).

Простейший вариант данного оператора будет следующим:

Если <Условие> тогда

[Операция1];

……..

[ОперацияN];

……..

КонецЕсли;

Рассмотрим подробнее. Для того чтобы начали выполняться операции внутри конструкции *Если…Тогда*, нужно, чтобы *Условие* принимало значение *Истина*. Если *Условие* принимает значение *Ложь*, то программа не будет заходить внутрь конструкции. Все просто. Теперь пример.

Создайте самостоятельно обработку, форму и в обработчике команды формы напишем следующий код:

```
&НаКлиенте
Процедура ВыполнениеКоманды(Команда)
    Перем Число1;

    ВвестиЧисло(Число1,"ВведитеЧисло");

    Если Число1 > 0 тогда
        Сообщить("Число " + Число1);
        Сообщить("Больше нуля");
    КонецЕсли;
КонецПроцедуры
```

Листинг 3.1.1

В данном коде мы указали, что если введенное число больше нуля, то выводим сообщение. В противном случае ничего не делаем.

Сохраните обработку и посмотрите, как она работает.

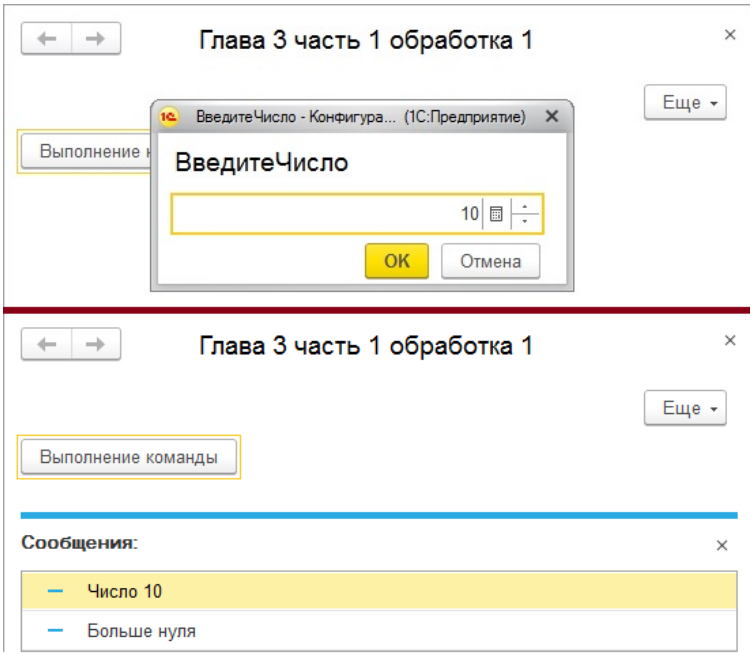

Рис. 3.1.1

Это самый примитивный вариант использования данного оператора.

Условие для данного оператора может быть каким угодно, главное, чтобы возвращалось значение типа Булево.

Например, может быть такой вариант:

Если Число1 > 1 и Число1 < 10 тогда….

Или такой вариант:

Условие = Число1 > 1 и Число1 < 10;

Если Условие тогда…

И так далее, более подробно про работу с булевыми операторами смотрите главу 2, часть 3 (см. стр. 63).

Продолжаем изучать дальше конструкцию «Если…». Что если нам надо выполнить какие-нибудь операции, когда выполняется наше условие, и другие операции, когда оно не выполняется? Для этого нам помогает оператор условия *Иначе*.

Рассмотрим схему:

Если <Условие> тогда

[Операция1];

…….

[ОперацияN];

Иначе

[Операция_2];

…….

[Операция_N]

КонецЕсли;

В данном случае выполнение программы после оператора *Иначе* будет, когда *Условие* примет значение *Ложь*.

Доработайте Ваш пример:

```
&НаКлиенте
Процедура ВыполнениеКоманды2(Команда)
    Перем Число1;

    ВвестиЧисло(Число1,"ВведитеЧисло");

    Если Число1 > 0 тогда
        Сообщить("Число " + Число1);
        Сообщить("Больше нуля");
    иначе
        Сообщить("Число " + Число1);
        Сообщить("Равно нулю или меньше нуля");
    КонецЕсли;

КонецПроцедуры
```

Листинг 3.1.2

В вышеприведенном коде мы вводим некоторое число в программу, после, как в предыдущем примере, пишем условие, что если это число больше нуля, то выводим соответствующее сообщение. Но также пишем оператор *Иначе*, после которого сообщаем, что число меньше или равно нулю. Запустите данную обработку и посмотрите, как она работает.

Опять усложним текущий пример. Что если нам необходимо вывести сообщение, когда число равно нулю, или, к примеру, равно 10?

Для этого мы используем условную конструкцию ***ИначеЕсли*** …***Тогда***.

Рассмотрим схему:

Если <Условие> тогда

[Операция1];

…….

[ОперацияN];

ИначеЕсли <Условие2> тогда

[Операция_2];

…….

[Операция_2N];

ИначеЕсли <Условие3> тогда

[Операция_2];

…….

[Операция_2N];

.

Иначе

[Операция_N];

…….

[Операция_NN]

КонецЕсли;

Операторов *ИначеЕсли…Тогда* внутри условия может быть бесконечное множество, оператор *Иначе* всегда должен быть один и идти после всех операторов *ИначеЕсли…Тогда*.

Теперь усложните Ваш пример:

```
&НаКлиенте
Процедура ВыполнениеКоманды3(Команда)
    Перем Число1;

    ВвестиЧисло(Число1,"ВведитеЧисло");

    Если Число1 >= 1 тогда
        Сообщить("Число " + Число1);
        Сообщить("Больше или равно единицы");
    ИначеЕсли Число1 >0 и Число1 < 1 тогда
        Сообщить("Число " + Число1);
        Сообщить("Меньше 1, но больше 0");
    ИначеЕсли Число1 = 0 тогда
        Сообщить("Число " + Число1);
        Сообщить("Равно нулю");
    иначе
```

```
        Сообщить("Число " + Число1);
        Сообщить("Меньше нуля");
    КонецЕсли;
КонецПроцедуры
```

Листинг 3.1.3

В данном примере мы проверяем, больше или равно единицы введенное нами число, если да, то сообщаем об этом. Потом проверяем, входит ли оно в интервал от 0 до 1, если да, то сообщаем. Следующим шагом проверяем, равно ли оно нулю, если да, то сообщаем. И если число не удовлетворяет всем вышеприведенным условиям, т.е. число меньше нуля, выводим сообщение об этом.

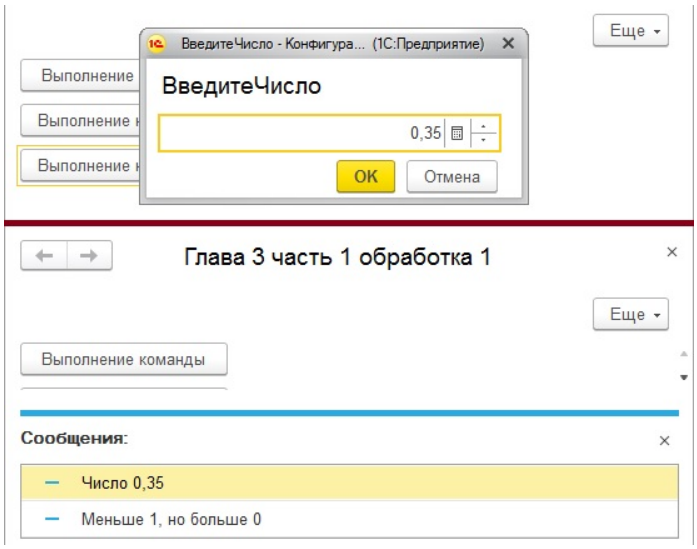

Рис. 3.1.2

Обращаю Ваше внимание, что условия *ИначеЕсли* будут выполняться только в том случае, когда не выполнились все предыдущие условия. Подправьте для примера Ваш код:

```
Перем Число1;
ВвестиЧисло(Число1,"ВведитеЧисло");

Если Число1 > 0 тогда
    Сообщить("Число " + Число1);
    Сообщить("Больше или равно единицы");
ИначеЕсли Число1 >0 и Число1 < 1 тогда
    Сообщить("Число " + Число1);
    Сообщить("Меньше 1, но больше 0");
ИначеЕсли Число1 = 0 тогда
    Сообщить("Число " + Число1);
    Сообщить("Равно нулю");
иначе
    Сообщить("Число " + Число1);
    Сообщить("Меньше нуля");
КонецЕсли;
```

Листинг 3.1.4

Если мы изменим первое условие, что *Число1* больше 0, то программа никогда не зайдет во второе условие. Так как, к примеру, число *0.6* больше нуля, и выполнится код под этим условием, после чего происходит выход из оператора.

Обратите внимание на один момент, если в *Условии* идет сразу переменная типа булево, то не надо писать так:

Если НашеУсловие = Истина тогда…

Если НашеУсловие = Ложь тогда…

Достаточно просто написать условие, например:

Если НашеУсловие тогда…

Или

Если Не НашеУсловие тогда….

Теперь переделайте предыдущий пример. Создайте переменную типа булево с нужным условием и подставляйте ее в конструкцию *Если…Тогда*.

Закончим разбирать данный оператор. Попробуйте сами усложнить последний пример (листинг 3.1.4), добавив различные дополнительные сравнения.

Теперь изучим похожий оператор, но несколько упрощенный – *Вычислить выражение по условию*. Он имеет следующий синтаксис:

?(<Условие>,<Выражение1>,<Выражение2>)

Где:

«Условие» – это логическое выражение, в зависимости от результата которого будет возвращено либо *Выражение1*, либо *Выражение2*;

«*Выражение1»* – возвращается, если логическое выражение *Истина*;

«Выражение2» – возвращается, если логическое выражение *Ложь*;

Рассмотрим пример. Вводим некоторое число, и надо вывести сообщение, больше оно нуля или меньше.

```
&НаКлиенте
Процедура ВыполнениеКоманды4(Команда)
    Перем Число1;

    ВвестиЧисло(Число1,"Введите число");

    Сообщение = ?(Число1 > 0,"Число " + Строка(Число1) + " больше 0",
                             "Число " + Строка(Число1) + " меньше или равно
0");
```

```
    Сообщить(Сообщение);
КонецПроцедуры
```

Листинг 3.1.5

В изучаемом примере первый параметр оператора «?» это наше условие: *Число1 >0*, второй параметр – строка с текстом, что наше число больше нуля, оно будет возвращаться, если первый параметр будет *Истина*. И третий параметр – строка с текстом, что наше число ноль или меньше нуля, оно будет возвращаться, если первый параметр будет *Ложь*.

Посмотрите, как она работает.

Сделайте еще один пример: пользователь вводит некоторое число, нужно извлечь из него корень, для этого оно всегда должно быть положительным.

```
&НаКлиенте
Процедура ВыполнениеКоманды5(Команда)
    Перем Число1;

    ВвестиЧисло(Число1,"Введите число");
    ЧислоДляКорня = ?(Число1 < 0, Число1*(-1),Число1);
    Корень = Окр(Sqrt(ЧислоДляКорня),5);

    Сообщить("Корень из модуля числа " + Строка(Число1) + " равен " +
Строка(Корень));
КонецПроцедуры
```

Листинг 3.1.6

Разберите данный пример самостоятельно.

Резюме

В этой части Вы изучили один из основных операторов языка программирования 1С. Если Вам не понятно его действие, то смотрите, как работает программа в отладке. Вы должны хорошо владеть данным оператором, чтобы перейти к дальнейшему изучению языка программирования 1С.

Часть 2. Цикл: Для…Цикл

Итак, начнем изучать циклы. Что такое вообще *Цикл*? ***Цикл*** – действие, выполняемое определенное количество раз. Под действием понимается один или несколько операторов.

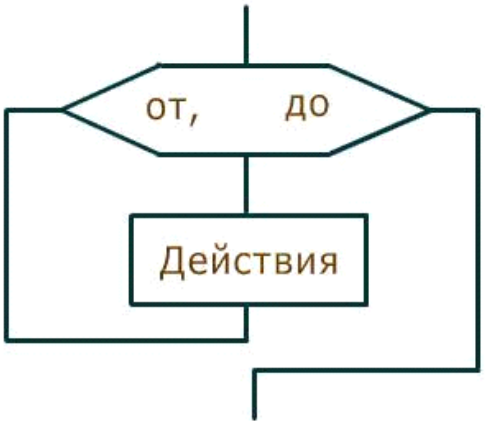

Рис. 3.2.1

Изучите рисунок 3.2.1, он как раз применим к осваиваемой теме. Пока идет перечисление какого-нибудь параметра от одного значения до другого, то выполняются определенные действия. Как только параметр достигает максимального значения, мы выходим из цикла и продолжаем выполнение программы.

Первый цикл который мы изучим, будет цикл ***Для…Цикл.***

Данный оператор имеет следующий синтаксис (пока мы его рассмотрим в самом общем виде, а углублять будем по мере изучения):

Для ИмяПеременной = Выражение1 по Выражение2 цикл

//операторы

КонецЦикла.

Где:

«ИмяПеременной» – это переменная счетчик цикла, его значение увеличивается на единицу при каждом повторении цикла;

«Выражение1» – числовое выражение, оно задает начальное значение, которое присваивается счетчику цикла при первом обходе. *Может быть любое целое число;*

«Выражение2» – максимальное значение счетчика цикла. Когда переменная *ИмяПеременной* становится больше *Выражение2*, программа выходит из цикла, т.е. операторы внутри конструкции *Для …Цикл* больше не выполняются. *Может быть любое целое число;*

«Цикл» – ключевое слово, которое открывает тело цикла;

«КонецЦикла» – ключевое слово, которое закрывает тело цикла;

Все, что находится между ключевыми словами *Цикл* и *КонецЦикла,* будет выполняться, пока *ИмяПеременной* находится в интервале от *Выражение1* до *Выражение2*.

Обращаю внимание, что *Выражение1* всегда должно быть меньше или равно *Выражение2.*

Создайте самостоятельно обработку, форму, и в обработчике команды формы напишите следующий код:

```
&НаКлиенте
Процедура ВыполнитьКоманду1(Команда)

    Для  н = 1 по 10 цикл;
        Сообщить("н = " + н);
    КонецЦикла;

КонецПроцедуры
```

Листинг 3.2.1

В данном примере мы создали цикл *Для…Цикл*, где счетчик цикла называется *«н»*, и начальное его значение равно *1*, а конечное его значение равно *10*.

И при каждом обходе цикла, выводим сообщение о том, чему равно *«н»*.

Сохраните обработку, запустите и посмотрите, что у Вас выйдет в окно сообщений.

Циклы можно вкладывать друг в друга. Сделайте следующий пример:

```
&НаКлиенте
Процедура ВыполнитьКоманду2(Команда)
    Для  н = 1 по 5 цикл

        Для к = 1 по 3 Цикл
            Сообщить("н.к = " + н + "." + к );
        КонецЦикла;

    КонецЦикла;
КонецПроцедуры
```

Листинг 3.2.2

Как работает такой цикл? Сначала выполняется внешний цикл, т.е. переменной *«н»* присваивается значение *1*, потом полностью выполняется внутренний цикл, т.е. значение *«к»* проходит от *1* до *7*, после этого управление передается опять на внешний цикл.

Сохраните обработку и посмотрите, что получится.

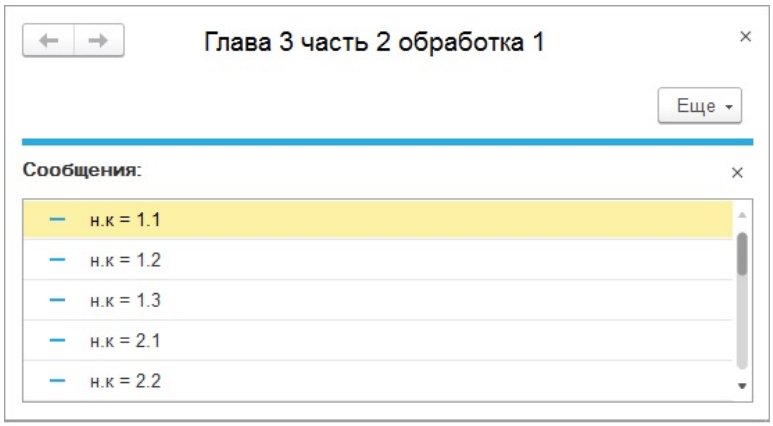

Рис. 3.2.2

Если Вам трудно понять очередность внешнего и внутреннего цикла, поставьте самостоятельно сообщения перед внутренним циклом, и после него, тогда все встанет на свои места.

Значения *Выражение1* и *Выражение2* могут задаваться пользователем в виде переменных.

Сделайте следующий пример:

```
&НаКлиенте
Процедура ВыполнитьКоманду3(Команда)
    Перем Выражение1, Выражение2;

    ВвестиЧисло(Выражение1,"Введите выражение1",2,0);
    ВвестиЧисло(Выражение2,"Введите выражение2 (больше выражение1)",2,0);

    Если Выражение1 > Выражение2 тогда

        Предупреждение("Выражение1 должно быть больше выражение2");

    иначе

        Для н = Выражение1 по Выражение2 цикл

            Сообщить(" н = " + н);

        КонецЦикла;

    КонецЕсли;
КонецПроцедуры
```

Листинг 3.2.3

Разберите его самостоятельно и посмотрите, как он работает.

Продолжим изучение цикла *Для..Цикл.* И углубим нашу схему, введя оператор *Прервать.*

Данный оператор прерывает выполнение цикла в любой его точке. После выполнения данной команды управление передается оператору, следующему после ключевого слова *КонецЦикла*.

С данным оператором конструкция цикла будет иметь следующий вид:

Для ИмяПеременной = Выражение1 по Выражение2 цикл

//операторы

Прервать.

//операторы

КонецЦикла.

Сделайте пример, который нам объяснит использование данного оператора.

```
&НаКлиенте
Процедура ВыполнитьКоманду4(Команда)

    Для н = 1 по 10 цикл
        к = 1/н;
        Сообщить("к = " + Окр(к,3));
        Прервать;
    КонецЦикла;

КонецПроцедуры
```

Листинг 3.2.4

Сохраните обработку и посмотрите, как она работает. Должен быть следующий результат:

Рис. 3.2.3

Как видно из результата выполнения данной обработки, произошел только первый шаг, дальше цикл не стал продолжаться. Как правило, данный оператор ставят внутри определенного условия.

Смотрим еще один пример:

```
&НаКлиенте
Процедура ВыполнитьКоманду5(Команда)

    Для н = - 10 по 10 цикл
        Если н = 0 тогда
            Прервать;
        КонецЕсли;
        к = 1/н;
        Сообщить("к = " + Окр(к,3));
    КонецЦикла;

КонецПроцедуры
```

Листинг 3.2.5

В этом примере наш цикл следует от *-10* до *10*, в том случае, когда счетчик цикла равняется нулю, мы выходим из цикла, чтобы не сгенерировалась исключительная ситуация.

Напоследок, мы разберем оператор *Продолжить*. Еще углубим схему конструкции.

Для ИмяПеременной = Выражение1 по Выражение2 цикл

//операторы

Продолжить;

//операторы

Прервать.

//операторы

КонецЦикла.

Оператор *Продолжить* передает управление в начало цикла, после этого происходит вычисление и проверка условий выполнения цикла. Все, что следует за данным оператором, на этой итерации цикла выполняться не будет.

Операторы Прервать и Продолжить применимы во всех циклах платформы 1С!

Сделайте следующий пример, и посмотрите результат его работы:

```
&НаКлиенте
Процедура ВыполнитьКоманду6(Команда)
    Для н = 1 по 10 цикл
        к = 1/н;
        р = 1/(Pow(н,2));
        Сообщить("1/н   = " + Окр(к,3));
    КонецЦикла;
КонецПроцедуры
```

Листинг 3.2.6

А теперь подправьте наш пример:

```
&НаКлиенте
Процедура ВыполнитьКоманду6(Команда)

    Для н = 1 по 10 цикл
        к = 1/н;
        р = 1/(Pow(н,2));
        Сообщить("1/н   = " + Окр(к,3));
        Продолжить;
        Сообщить("--------------------");
        Сообщить("1/н^2 = " + Окр(р,3));
    КонецЦикла;

КонецПроцедуры
```

Листинг 3.2.7

Выполните обработку. Как видите, сообщение о делении единицы на *«н» в квадрате* не выходит, потому что оператор *Продолжить* передал управление в начало цикла.

Также оператор *Продолжить* можно выполнять внутри условия.

Сделайте следующий пример:

```
&НаКлиенте
Процедура ВыполнитьКоманду7(Команда)

    Для н = - 10 по 10 цикл
        Если н = 0 тогда
            Продолжить;;
        КонецЕсли;
        к = 1/н;
        Сообщить("1/н = " + Окр(к,3));
    КонецЦикла;

КонецПроцедуры
```

Листинг 3.2.8

В данном примере при *«н»* равном нулю, мы не прерываем цикл, как это делали в предыдущем уроке, а просто начинаем его заново, чтобы не сгенерировалась ошибка.

Только что мы с Вами изучили самый простой вид цикла, который есть в языке программирования 1С. Он простой, но основополагающий, научитесь хорошо его понимать и с легкостью им пользоваться. Только после этого приступайте к изучению следующих частей данного урока.

Часть 3. Цикл: Пока…Цикл. Рекурсия переменных

Рассмотрим еще один цикл, это цикл с условием. Принцип его работы мы разберем на следующей схеме:

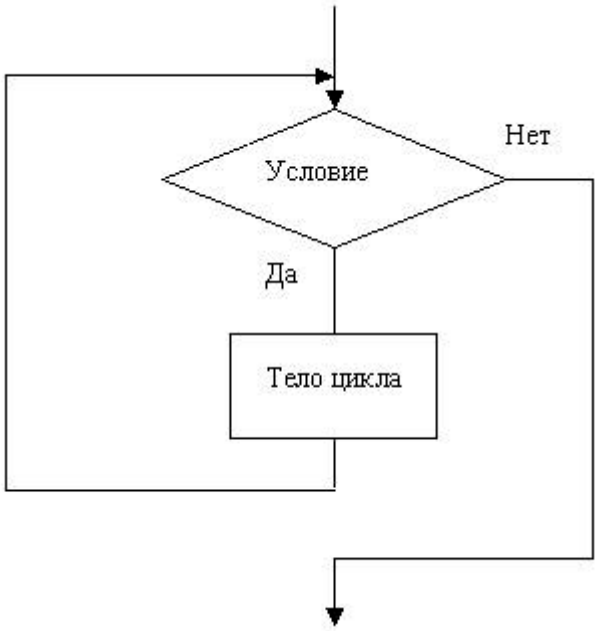

Рис. 3.3.1

Посмотрите на данную схему: перед телом цикла задается какое-нибудь условие, пока данное условие истинно, выполняется тело цикла, в противном случае программа выходит из цикла. Рассмотрим синтаксис данного оператора в программе 1С.

```
Пока <Условие> Цикл

//операторы

Продолжить;

//операторы

Прервать;

КонецЦикла;
```

Разберем данный оператор. Ключевое слово *Пока* задает некоторое условия, оно должно иметь тип булево, после данного условия следует ключевое слово *Цикл*, открывающее тело цикла, а закрывает тело цикла ключевое слово *КонецЦикла*.

Операторы *Продолжить* и *Прервать* обладают теми же функциями, как и в предыдущем цикле *Для…Цикл*. Это переход в начало цикла и выход из цикла соответственно.

Перед тем, как рассматривать примеры цикла *Пока...Цикл*, мы изучим ***Рекурсию переменных***.

Рекурсия – определение нового значения параметров при помощи старого значения, используя единый идентификатор.

Посмотрим, как это выглядит.

и = 0;

и = и + 1;

В этом случае переменная «и» увеличивает сама себя на 1.

Сделаем самый элементарный пример, и на нем же изучим рекурсию и цикл *Пока…Цикл.* Пример таков: нам необходимо вывести ряд чисел от *0* до *100* с шагом *5 (0,5,10,15 и т.д.)*.

Циклом *Для…Цикл*, который мы изучали в предыдущей части, сделать этот пример можно: нужно при помощи оператора % определять, кратен ли текущий счетчик цикла 5 или нет, и исходя из этого выводить сообщение. Но гораздо проще и понятнее будет сделать данную задачу, используя цикл *Пока...Цикл*.

Создайте самостоятельно обработку, форму и в обработчике команды формы напишем следующий код:

```
&НаКлиенте
Процедура ВыполнитьКоманду1(Команда)
    н = 0;
    Шаг = 5;

    Пока н <= 100 Цикл
        Сообщить("н =" + н);
        н = н + 5
    КонецЦикла;
КонецПроцедуры
```

Листинг 3.3.1

Разберем данный пример. Сначала задаем начальное значение выводимой переменной, потом шаг, с которым она будет выходить на экран.

После этого мы задаем условие цикла: что он будет выполняться, пока переменная *«н»* меньше или равна ста. То есть если переменная *«н»* меньше или равна *100*, то программа будет заходить в тело цикла и выполнять операторы в цикле. Как только данное условие перестанет выполняться, то будут исполняться операторы после ключевого слова *КонецЦикла*. Выводим сообщение чему равно *«н»*. Последний шаг – выполняем рекурсию переменной, увеличивая ее на 5.

Сохраните обработку, запустите и посмотрите, как работает данный код.

Рекурсивно можно вычитать, умножать и делить переменные (для типа число). Рекурсивный вызов возможен также для строк и дат.

Еще один пример на рекурсию. Нам надо получить *«н»* факториал.

5! = 1*2*3*4*5

```
&НаКлиенте
Процедура ВычислениеФакторила(Команда)
    Перем н;
    ВвелиЧисло = ВвестиЧисло(н,"Введите число",1,0);

    Если ВвелиЧисло тогда
        Если н > 0 тогда
            НачалоФакториала = 1;
            Итерация = 0;
            Пока Итерация < н цикл
                Итерация = Итерация + 1;
                НачалоФакториала = НачалоФакториала*Итерация;
            КонецЦикла;
            Сообщить(Строка(н) + "! = " +
                        Строка(НачалоФакториала));
        КонецЕсли;
    КонецЕсли;

КонецПроцедуры
```

Листинг 3.3.2

Разберем его. Получаем интерактивно некоторое целое число, проверяем, ввели мы его или нет, записываем введенное число в переменную *«н»*.

Если число ввели, и если оно больше нуля, задаем переменную *НачалоФакториала*, которая равна единице и которая будет рекурсивно увеличиваться, и переменную *Итерация*.

Определяем наш цикл, он будет работать пока переменная *Итерация* меньше переменной *«н»*. Т.е. если мы *«н»* задали равным 5, то цикл будет выполняться при следующих значениях переменной *Итерация* : *0,1,2,3,4.*

Сразу же в начале тела цикла увеличиваем переменную *Итерация* на *1*.

И последний шаг: внутри цикла рекурсивно вызываем переменную *Факториал,* умножая ее на переменную *Итерация*.

Т.е. в первом шаге цикла: *НачалоФакториала = 1 * 1*.

Во втором шаге цикла: *НачалоФакториала = 1 * 2*.

В третьем шаге цикла: *НачалоФакториала = 2 * 3*.

И так далее.

В конце выводим сообщение.

Разберите данный пример, если он Вам не понятен. Разберите его в отладке, посмотрите, как изменяются переменные.

Самостоятельно измените пример так, чтобы *Итерация* начиналась с *1*, поменяйте условие цикла или место рекурсии переменной *Итерация*, чтобы пример работал правильно.

Сделайте еще один пример на цикл *Пока…Цикл*.

Мы имеем две даты, вторая дата старше первой, надо вывести все даты в промежутке между ними.

```
&НаКлиенте
Процедура ВыводДат(Команда)

    Перем ДатаНач, ДатаКон;

    ДатаНачВведена = ВвестиДату(ДатаНач,"Введите начальную дату",
                                ЧастиДаты.Дата);
    ДатаКонВведена = ВвестиДату(ДатаКон,"Введите конечную дату",
                                ЧастиДаты.Дата);

    Если ДатаНачВведена и ДатаКонВведена тогда
        Если ДатаНач < ДатаКон тогда
            Пока ДатаНач <= ДатаКон цикл
                Сообщить(Формат(ДатаНач,"ДЛФ = ДД"));
                ДатаНач = ДатаНач + 3600*24;
            КонецЦикла;
        иначе
            Предупреждение("Начальная дата должна быть раньше конечной
даты");
        КонецЕсли;
    иначе
        Предупреждение("Вы не ввели одну из дат");
    КонецЕсли;

КонецПроцедуры
```

Листинг 3.3.3

Разберем данный пример.

Первым делом мы задаем и интерактивно вводим переменные. Для того чтобы исключить ситуацию, когда пользователь не ввел переменную, мы делаем проверку. Если Вы забыли, то напоминаю, что функции *ВвестиДату, ВвестиСтроку, ВвестиЧисло* возвращают *Истину*, если пользователь ввел строку, число или дату, и возвращают *Ложь*, если нет.

В том случае, если обе переменные *ДатаНачВведена* и *ДатаКонВведена* принимают значение *Истина*, то мы заходим в условие, а если нет, то выдаем пользователю предупреждение.

Следующим шагом мы проверяем то, что дата начальная должна быть всегда меньше даты конечной, иначе также выдаем пользователю сообщение.

После того как проверка прошла, начинает выполняться цикл. В условии цикла мы проверяем то, что дата начальная меньше или равна дате конечной. Если условие истинно, то заходим в тело цикла. В теле цикла мы первым делом выводим на экран начальную дату. А потом - внимание! - мы делаем то, что называется рекурсией переменных (о чем я вам рассказывал в начале этой части главы), но только применимо к датам.

После данных манипуляций, *ДатаНач* увеличивается на один день. Опять проверяем наше условие, и если *ДатаНач* все еще меньше или равно *ДатаКон*, то идем в цикл, иначе выходим из него.

Запустите обработку и посмотрите, как она работает.

На этом мы закончим изучение циклов. Всего в языке программирования 1С существует три вида циклов. Два из них мы изучили в этом уроке. Третий цикл будем изучать в седьмой главе, когда будем проходить универсальные коллекции значений. Пока изучите и научитесь работать с циклом *Пока…Цикл*, изучите и освойте рекурсию переменных. Поскольку в практической работе Вам часто придется сталкиваться с данными вопросами.

Часть 4. Работа с процедурами и функциями

В данной части изучим работу с процедурами и функциями. Что такое процедура?

Процедура - это идентификатор, выполняющий некоторые действия, определенные пользователем или системные. Говоря простым языком: если мы пропишем внутри нашего кода некоторую процедуру (не важно, нами написанную или процедуру программы 1С), то когда выполнение программы дойдет до этой строки, тогда будут выполнены определенные действия. Рассмотрим уже известные нам процедуры *Сообщить* и *Предупреждение*. Из предыдущих примеров Вы видели, что при выполнении процедуры *Сообщить* в окно сообщений выходит текст, введенный в качестве первого параметра данной процедуры. А при выполнении процедуры *Предупреждение* появляется всплывающее окно с описанием текста, заданного в качестве параметра данной процедуры.

Что такое функция?

Функция - это идентификатор, выполняющий некоторые действия, определенные пользователем или системные, и возвращающий значение определенного типа. Как видите, функции отличаются от процедур только тем, что возвращают определенное значение. К примеру, функция *СтрДлина* вычисляет длину строки, которая передается в нее в качестве параметра, и возвращает это значение пользователю в виде числа. Или функция *ТекущаяДата* определяет системную дату компьютера и возвращает это значение в виде даты.

В языке программирования 1С разработчик или внедренец может самостоятельно разрабатывать любые процедуры и функции, которые необходимы для выполнения алгоритма программы.

У многих может возникнуть вопрос: для чего необходимо самим писать процедуры или функции? Во-первых, Вы упростите свой код, если он будет описан с помощью процедур, такой код будет хорошо читаем и понятен для следующего разбора, причем, возможно, и Вами. Во-вторых, чтобы не писать много раз одно и то же в коде. Если у Вас один и тот же код повторяется несколько раз, то лучше его написать в процедуре или функции, а потом просто использовать данную процедуру или функцию в нужных местах. Так Вы не только сделаете хорошо читаемый код, но и подстрахуете себя от возможных ошибок. Понятно, что одно дело исправить ошибку в одном месте внутри процедуры или функции, а другое дело в 10 местах: Вы можете исправить в 9 местах, а в 10-м забыть или пропустить, в результате качество Вашей работы будет страдать.

Это была теоретическая часть, сейчас мы изучим синтаксис процедур в самом простом для начинающих варианте.

```
Процедура ИмяПроцедуры()

//оператор

КонецПроцедуры
```

Разберем этот синтаксис. Ключевое слово *Процедура* идет всегда перед названием. После данного ключевого слова всегда идет название процедуры. Все названия процедур внутри одного модуля должны быть уникальны. Поскольку в данном примере синтаксиса процедура

представлена без параметров (их мы разберем позже), то просто пишем левую и правую скобку вместе. После этих скобок идут операторы, которые описывают те или иные действия внутри процедуры (для удобного чтения я рекомендую начинать писать их со следующей строки и делать отступ с помощью клавиши «*Tab*»). Ключевое слово *КонецПроцедуры* завершает код процедуры. Что бы Вы ни написали после этого ключевого слова, все это к этой процедуре не будет иметь никакого отношения.

Так же в предыдущих листингах Вы должны были обратить внимание на конструкцию *&НаКлиенте*. Это так называемая *директива компиляции*, с её помощью программист может задать, где будет выполняться его код, на сервере или на клиенте. Более подробно работу этой директивы мы разберем в пятой главе, посвященной работе с управляемыми формами. Пока же просто устанавливайте эту директиву перед началом всех процедур и функций, которые Вы используете в модуле формы. На данном этапе обучения Вам не стоит задумываться о её сути.

Создайте пример. В этом примере пользователь вводит два числа, и оба раза вычисляется квадратный корень из данных чисел. Если число меньше нуля, то пользователю выходит целый ряд сообщений. Вывод этого ряда сообщений мы и создадим в процедуре.

Создайте обработку, форму и в модуле формы напишите следующую процедуру:

```
&НаКлиенте
Процедура СообщениеОНеправильномЧисле()
    Сообщить("Введенное число меньше нуля");
    Сообщить("Вычисление не будет произведено");
    Сообщить("Задайте другие числа");
КонецПроцедуры
```

Листинг 3.4.1

А в обработчике команды формы нижеприведенный код:

```
&НаКлиенте
Процедура ВыполнитьКоманду(Команда)
    Перем А, Б;
    Если ВвестиЧисло(А,"Введите число А") тогда
        Если А >= 0 тогда
            КореньА = Sqrt(А);
            Сообщить("Корень из числа А = " + Окр(КореньА,5));
        иначе
            СообщениеОНеправильномЧисле();
        КонецЕсли;
    КонецЕсли;
    Если ВвестиЧисло(Б,"Введите число Б") тогда
        Если Б >= 0 тогда
            КореньБ = Sqrt(Б);
            Сообщить("Корень из числа Б = " + Окр(КореньБ,5));
        иначе
            СообщениеОНеправильномЧисле();
        КонецЕсли;
    КонецЕсли;
КонецПроцедуры
```

Листинг 3.4.2

Разберем данный пример.

Мы вводим некоторое первое число *А*, проверяем на то, что оно введено. Я сразу написал функцию *ВвестиЧисло* как параметр оператора *Если*, чтобы оптимизировать код программы. Потом мы проверяем больше или равно нуля наше число, если да - то вычисляем корень, если нет - пишем вновь созданную процедуру, которая выводит целый ряд сообщений. Точно такие же действия проделываем с числом *Б*.

Запустите обработку и посмотрите, как работает данный код.

Сейчас пытливый читатель спросит: ведь эти два отрезка кода, когда вводим первое число и вычисляем корень, а потом второе число и вычисляем корень, очень похожи. Нельзя ли и их каким-нибудь образом сделать в виде процедуры? Да, можно.

Напишите в модуле формы следующую процедуру:

```
&НаКлиенте
Процедура ИнтерактивноВВестиКореньИВычислить()
    Перем А;

    Если ВвестиЧисло(А,"Введите число") тогда
        Если А >= 0 тогда
            КореньА = Sqrt(А);
            Сообщить("Корень из числа " +
                     А + " = " + Окр(КореньА,5));
        иначе
            Сообщить("Введенное число меньше нуля");
            Сообщить("Вычисление не будет произведено");
            Сообщить("Задайте другие числа");
        КонецЕсли;
    КонецЕсли;
КонецПроцедуры
```

Листинг 3.4.3

А в обработчике команды код:

```
&НаКлиенте
Процедура ВыполнитьКоманду2(Команда)
    ИнтерактивноВВестиКореньИВычислить();
    ИнтерактивноВВестиКореньИВычислить();
КонецПроцедуры
```

Листинг 3.4.4

Посмотрите, код будет выполняться точно так же, как и в предыдущей обработке. Но написание кода в обработчике команды заметно упростилось.

Посмотрев на это, пытливый читатель может задать новый вопрос: а если необходимо одно число задать интерактивно, а одно в программе, как мы поступим в этом случае, как это число передать в процедуру?

Сделать это можно двумя способами. Первый: сделать эту переменную глобальной и передавать в процедуру. О глобальных и локальных переменных мы узнаем в четвертой главе. Либо передать эту переменную в качестве параметра.

А для этого мы рассмотрим наш более усложненный синтаксис. Обратите внимание: все, что касается параметров, справедливо и для процедур, и для функций. Просто сейчас мы рассмотрим данный вопрос на примере процедур.

Процедура ИмяПроцедуры(*Параметр1*, *Параметр2*, ….,*ПараметрN*)

КонецПроцедуры

После названия процедуры внутри скобок идет описание параметров, которые будут передаваться в процедуру. Из основного кода параметры должны передаваться в той последовательности, в которой описаны в процедуре. Названия параметров внутри процедуры должны совпадать с названиями в скобках.

Переделайте Ваш пример.

Измените процедуру *ВычислитьКореньИВывести* следующим образом:

```
&НаКлиенте
Процедура ВычислитьКореньИВывести(ЧислоПодКорнем, НазваниеЧисла)
    Если ЧислоПодКорнем >= 0 тогда
        КореньА = Sqrt(ЧислоПодКорнем);
        Сообщить("Корень из числа " + НазваниеЧисла + " = " +
Окр(КореньА,5));
    иначе
        Сообщить("Число " + НазваниеЧисла + " меньше нуля");
        Сообщить("Вычисление не будет произведено");
        Сообщить("Задайте другие числа");
    КонецЕсли;
КонецПроцедуры
```

Листинг 3.4.5

И создадим новую команду формы, в обработчике которой напишем следующий код:

```
&НаКлиенте
Процедура ВычислитьКорень(Команда)
    Перем А,Б;

    А = 10.11;

    ВычислитьКореньИВывести(А,"А");

    Если ВвестиЧисло(Б, "Введите число Б") Тогда
        ВычислитьКореньИВывести(Б,"Б");
    КонецЕсли;
КонецПроцедуры
```

Листинг 3.4.6

Во вновь созданную процедуру мы передаем два параметра: это число, которое будет под корнем, и название данного числа. Внутри процедуры мы совершаем с этими параметрами все необходимые действия. Как видите, внутри процедуры используются те же названия переменных, что и были заданы в скобках.

Переходим к коду, где непосредственно используется наша процедура в обработчике *«ВычислитьКорень»*. Задаем две переменных, одной присваиваем число явно в коде. И передаем эту переменную в процедуру *ВычислитьКореньИВывести*, также вторым параметром передаем название нашего числа, переменная типа строка. Как понятно из примера, эти параметры нельзя путать. Если мы напишем так:

ВычислитьКореньИВывести("А",А);

то мы будем пытаться извлечь квадратный корень из строки, компилятор выдаст ошибку. Очень часто начинающие программисты путают местами параметры, что приводит к различным проблемам.

Еще один момент - это то, что количество параметров заданных в процедуре при разработке, должно соответствовать количеству параметров при использовании данной процедуры в коде.

Т.е. нельзя написать так:

ВычислитьКореньИВывести(А);

Компилятор выдаст ошибку.

С параметрами должно быть понятно, но есть один момент. Когда мы передаем параметр в процедуру, то мы передаем не просто само по себе число, строку или какой-то объект, а, по сути, передаем ссылку на данное число, строку или объект. Это значит, что если внутри процедуры мы изменим данный параметр, то поменяется и сама та переменная, которая передается в виде параметра в нашу процедуру.

Звучит несколько сложно, но один простой пример прояснит картину.

Сделаем новую процедуру *ВычислитьКореньПоМодулюИВывести*:

```
&НаКлиенте
Процедура ВычислитьКореньПоМодулюИВывести(ЧислоПодКорнем, НазваниеЧисла)

    ЧислоПодКорнем = ?(ЧислоПодКорнем >=0,ЧислоПодКорнем,ЧислоПодКорнем*(-
1));
    КореньА = Sqrt(ЧислоПодКорнем);
    Сообщить("Корень из модуля числа " +
             НазваниеЧисла + " = " +
             Окр(КореньА,5));

КонецПроцедуры
```

Листинг 3.4.7

Создадим новую команду *«ВычислитьКореньПоМодулю»* и в её обработчике напишем следующий код:

```
&НаКлиенте
Процедура ВычислитьКореньПоМодулю(Команда)
    Перем А;
```

```
    Если ВвестиЧисло(А, "Введите число А") Тогда
        Сообщить("Число А = " + А);
        ВычислитьКореньПоМодулюИВывести(А,"А");
        Сообщить("Число А = " + А);
    КонецЕсли;
КонецПроцедуры
```

Листинг 3.4.8

В данном примере если *ЧислоПодКорнем* меньше нуля, то мы его рекурсивно умножаем на минус единицу.

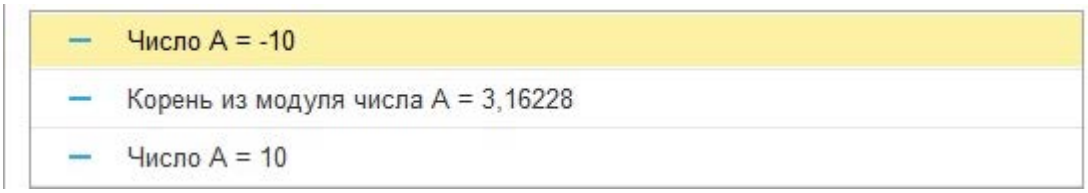

Рис. 3.4.1

Запустите данный пример. И при вводе отрицательного числа будет изменено само число, которое Вы ввели.

В каких-то случаях Вас это не будет сильно волновать, но в других случаях Вы захотите, чтобы переменная, которую передаете в процедуру или функцию, оставалась прежней. Как это сделать? Смотрим следующий синтаксис:

Процедура ИмяПроцедуры(*Знач Параметр1, Знач Параметр2, ….,Знач ПараметрN*)

КонецПроцедуры

Ключевое слово *Знач* перед параметром означает, что данный параметр передается в процедуру по значению, т.е. изменение параметра внутри процедуры никак не повлияет на значение фактического параметра, который был передан в процедуру. Ключевое слово *Знач* является необязательным, т.е. переменная может передаваться как ссылочно, так и по значению.

Теперь измените Ваш предыдущий пример. Подправьте процедуру *«ВычислитьКореньПоМодулюИВывести»*

```
Процедура ВычислитьКореньПоМодулюИВывести(Знач ЧислоПодКорнем, НазваниеЧисла)
    ЧислоПодКорнем = ?(ЧислоПодКорнем >=0,ЧислоПодКорнем,ЧислоПодКорнем*(-1));
    КореньА = Sqrt(ЧислоПодКорнем);
    Сообщить("Корень из модуля числа " +
             НазваниеЧисла + " = " +
             Окр(КореньА,5));
КонецПроцедуры
```

Листинг 3.4.9

Обработчик команды *«ВычислитьКореньПоМодулю»* не изменится.

Запустите обработку, и при вводе отрицательного числа посмотрите, изменится ли само введенное число или нет.

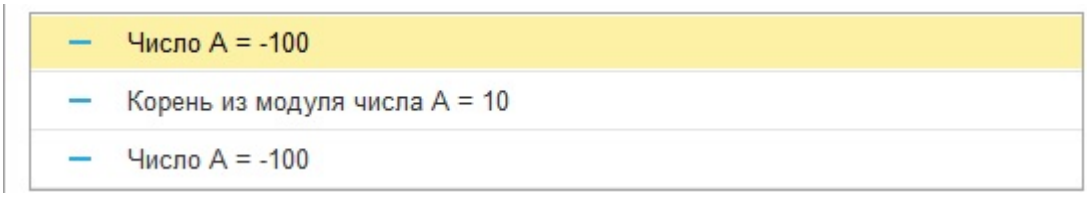

Рис. 3.4.2

Продолжим рассматривать передачу параметров в процедуры и функции. Мы уже знаем, что все параметры, которые мы передаем в процедуру или функцию, обязательны, т.е. когда Вы в коде задаете процедуру, то обязаны перечислить их все, но есть возможность создать параметр по умолчанию, тогда у Вас не будет обязанности перечислять его в процедуре.

Рассмотрим синтаксис.

Процедура ИмяПроцедуры(*Параметр1 = Умол1, Параметр2 = Умол2, ….,ПараметрN = УмолN)*

КонецПроцедуры

Разберем данный синтаксис. Такая форма описания параметра означает, что данный параметр имеет значение по умолчанию и тем самым он становится необязательным, т.е. когда мы будем писать данную процедуру в модуле, то вольны не указывать этот параметр.

Доработайте пример. В этот раз мы вычисляем корень числа и выводим его на экран с точностью до 5-го знака, но пользователь может менять точность.

Создадим новую процедуру *«ВычислитьКореньИВывестиСОкруглением»*

```
&НаКлиенте
Процедура ВычислитьКореньИВывестиСОкруглением(Знач ЧислоПодКорнем,
НазваниеЧисла, Точность = 5)

    ЧислоПодКорнем = ?(ЧислоПодКорнем >=0,ЧислоПодКорнем,ЧислоПодКорнем*(-
1));
    КореньА = Окр(Sqrt(ЧислоПодКорнем),Точность);
    Сообщить("Корень из модуля числа " + НазваниеЧисла + " = " + КореньА);

КонецПроцедуры
```

Листинг 3.4.10

Опять создадим новую команду «*ВычислитьКореньСОкруглением»*, и в её обработчике напишем следующий код:

```
&НаКлиенте
Процедура ВычислитьКореньСОкруглением(Команда)
    Перем А;

    Если ВвестиЧисло(А, "Введите число А") Тогда
        ВычислитьКореньИВывестиСОкруглением(А,"А");
```

```
        ВычислитьКореньИВывестиСОкруглением(А,"А",7);
    КонецЕсли;
КонецПроцедуры
```

Листинг 3.4.11

Разберем данный пример. Мы переделали нашу процедуру *ВычислитьКореньСОкруглением*, добавив параметр *Точность*, который по умолчанию равен *5*. Его мы используем для округления числа, которое получилось при извлечении квадратного корня. После этого мы используем процедуру два раза. Первый раз без третьего параметра, а второй раз с третьим параметром.

Сохраните обработку и запустите ее. И введите любое число.

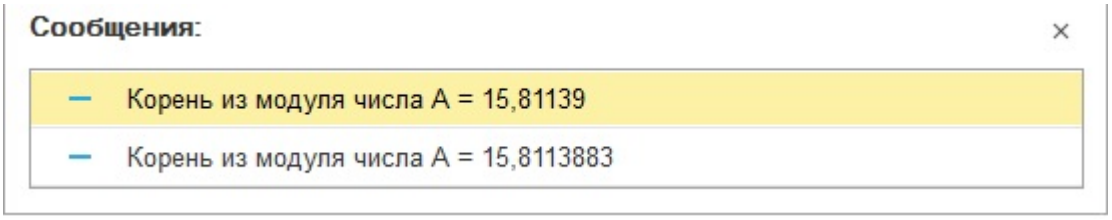

Рис. 3.4.3

Как Вы видите, в первый раз значение вышло с точностью до 5-го знака, а второй до 7-го.

Обращаю Ваше внимание, что параметр, который передается по значению, также может быть и параметром по умолчанию.

Рассмотрим следующий синтаксис:

Процедура ИмяПроцедуры(*Знач Параметр1 = Умол1*, *Знач Параметр2 = Умол2*, …., *Знач ПараметрN = УмолN*)

КонецПроцедуры

Данный синтаксис нам говорит, что параметр по умолчанию будет передаваться по значению. Сделайте следующий пример.

В этом примере, если *Точность* будет меньше *5*, то будем выводить все равно с точностью равной *5*.

Создадим новую процедуру *«ВычислитьКореньИВывестиТочностьПять»*:

```
&НаКлиенте
Процедура ВычислитьКореньИВывестиТочностьПять(Знач ЧислоПодКорнем,
НазваниеЧисла, Знач Точность = 5)

    ЧислоПодКорнем = ?(ЧислоПодКорнем >=0,ЧислоПодКорнем,ЧислоПодКорнем*(-
```

```
1));
    Точность = ?(Точность < 5, 5, Точность);
    КореньА = Sqrt(ЧислоПодКорнем);
    Сообщить("Корень из модуля числа " +
              НазваниеЧисла + " = " +
              Окр(КореньА,Точность));

КонецПроцедуры
```

Листинг 3.4.12

Опять создадим новую команду «*ВычислитьКореньСТочностьюПять*», и в её обработчике напишем следующий код:

```
&НаКлиенте
Процедура ВычислитьКореньСТочностьюПять(Команда)

    Перем А;

    Если ВвестиЧисло(А, "Введите число А") Тогда
        ВычислитьКореньИВывестиТочностьПять(А,"А");
        ВычислитьКореньИВывестиТочностьПять(А,"А",3);
    КонецЕсли;
КонецПроцедуры
```

Листинг 3.4.13

Разберите данный пример самостоятельно.

Рассмотрим последний оператор, который применим просто к процедурам, а не к их параметрам. Это оператор *Возврат*.

Рассмотрим синтаксис.

Процедура ИмяПроцедуры(*Параметр1*, *Параметр2*, ….,*ПараметрN*)

///операторы

Возврат;

//операторы

КонецПроцедуры

В данном синтаксисе, параметры могут задаваться каким угодно из вышеперечисленных способов.

После того, как мы написали имя процедуры, перечислили параметры, пишем наши действия. И когда в ходе выполнения процедуры встретится ключевое слово *Возврат*, то программа тут же выйдет из нее и ее выполнение прекратится. После этого действия работать будут операторы, следующие после ключевого слова *КонецПроцедуры*.

Сделайте пример. В этом примере Вы также будете вычислять квадратный корень числа, но если число отрицательное, то будет выполнен выход из процедуры.

Создадим процедуру *«ВычислитьКореньТолькоПоложительныхЧисла»*:

```
&НаКлиенте
Процедура ВычислитьКореньТолькоПоложительныхЧисла(ЧислоПодКорнем,
НазваниеЧисла)
    Если ЧислоПодКорнем < 0 тогда
        Предупреждение("Нельзя извлекать квадратный корень из отрицательного
числа!");
        Возврат;
    КонецЕсли;
    КореньА = Sqrt(ЧислоПодКорнем);
    Сообщить("Корень из числа " + НазваниеЧисла + " = " + Окр(КореньА,5));
КонецПроцедуры
```

Листинг 3.4.14

Опять создадим новую команду «*ВычислитьКореньПоложительногоЧисла*» и в её обработчике напишем следующий код:

```
&НаКлиенте
Процедура ВычислитьКореньПоложительногоЧисла(Команда)

    Перем А;
    Если ВвестиЧисло(А, "Введите число А") Тогда
        ВычислитьКореньТолькоПоложительныхЧислел(А,"А");
    КонецЕсли;

КонецПроцедуры
```

Листинг 3.4.15

Разберем данный пример. В самом начале процедуры мы проверяем, меньше ли нуля параметр *ЧислоПодКорнем* или нет. Если меньше нуля, то выводим предупреждение и посредством оператора *Возврат* выходим из процедуры.

Сохраните и перезапустите Вашу обработку.

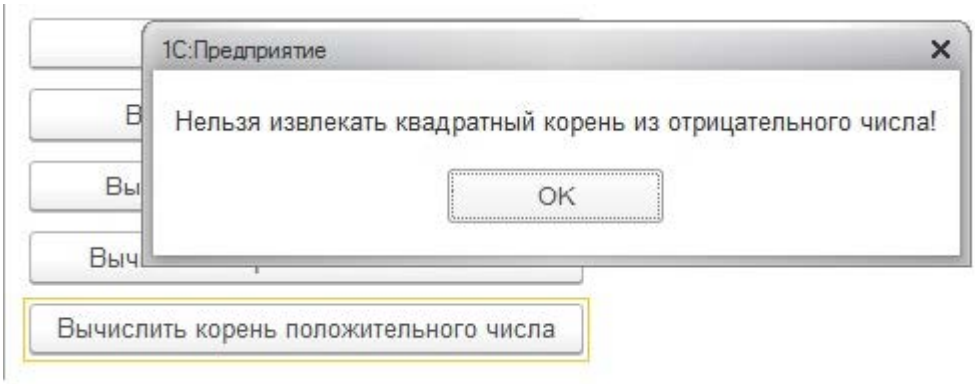

Рис. 3.4.4

Обращаю Ваше внимание: что бы Вы ни написали после слова *Возврат*, все это выполняться уже не будет. Теперь самостоятельно напишите предупреждение после ключевого слова *Возврат* и посмотрите, что получится.

Только что мы изучили с Вами работу с процедурами в языке программирования 1С. Теперь Вы самостоятельно сможете создавать любые процедуры, какие захотите.

Рассмотрим синтаксис написания функции. Как я уже рассказывал, все, что мы изучили касательно параметров на примере процедур, абсолютно справедливо и для функций.

Итак, синтаксис функций.

Функция ИмяФункции(*<Параметры>*)

//операторы

Возврат ВозвращаемоеЗначение;

КонецФункции

Рассмотрим данный синтаксис. Ключевое слово *Функция* идет всегда перед названием функции. Поскольку все параметры мы разобрали на примере процедуры, не будем подробно останавливаться на этом. После параметров идут операторы, которые описывают те или иные действия внутри функции (для удобного чтения кода я рекомендую начинать писать их со следующей строки и делать отступ с помощью клавиши *«Tab»*). Основное отличие функции от процедуры в том, что ключевое слово *Возврат* не только завершает выполнение функции, но и возвращает определенное значение в то выражение, где использовалась данная функция. Ключевое слово *КонецФункции* завершает код функции. Как и в случае процедуры, все, что Вы написали после данного слова, не будет иметь к вашей функции никакого отношения.

Рассмотрим пример. Напишем функцию *Сигнум*, параметром которой является некоторое число, и она возвращает *1*, если число больше нуля; ноль, если число равно нулю; и *-1*, если число меньше нуля.

Создайте в модуле формы следующую функцию:

```
&НаКлиенте
Функция Сигнум(ЧислоА)
    Перем Сигн;

    Если ЧислоА > 0 тогда
        Сигн = 1;
    ИначеЕсли ЧислоА = 0 тогда
        Сигн = 0
    ИначеЕсли ЧислоА < 0 тогда
        сигн = -1
    КонецЕсли;

    Возврат Сигн;
КонецФункции
```

Листинг 3.4.16

Опять создадим новую команду «*ВычислитьСигнумЧисла*» и в её обработчике напишем следующий код:

```
&НаКлиенте
Процедура ВычислитьСигнумЧисла(Команда)
    Перем А;
    Если ВвестиЧисло(А, "Введите число А") Тогда
        В = Сигнум(А);
        Сообщить("Signum (" + А + ") = " + В);
    КонецЕсли;
КонецПроцедуры
```

Листинг 3.4.17

Разберем данный код.

В модуле формы мы создали функцию, которая называется *Сигнум*, указываем, что у нее только один параметр, назовем его *ЧислоА*. После этого задаем переменную внутри функции, которую мы будем возвращать, назовем ее *Сиган*. Делаем сравнение, в результате которого *Сиган* становится равен либо единице, либо нулю, либо минус единице.

Смотрим основной код в обработчике команды *«ВычислитьСигнумЧисла»*. Первым делом интерактивно вводим некоторое число *А*. В следующей строке используем Вашу вновь созданную функцию: передаем в нее переменную *А*, а возвращать она будет переменную *В*, значение которой выводим в сообщении..

Сохраните обработку и посмотрите, как работает Ваша функция.

Обращаю ваше внимание, что необязательно писать один *Возврат* и только в конце функции, вполне может быть и такой вид функции:

```
&НаКлиенте
Функция л_Сигнум(ЧислоА)

    Если ЧислоА > 0 тогда
        Возврат 1;
    ИначеЕсли ЧислоА = 0 тогда
        Возврат 0;
    Иначе
        Возврат -1
    КонецЕсли;

КонецФункции
```

Листинг 3.4.18

Как видно из данного примера, после каждого результата сравнения мы возвращали нужное нам значение.

Еще одной особенностью функций является то, что необязательно какой-то переменной присваивать значение функции. Эту функцию можно использовать в качестве процедуры.

Сделаем пример, с помощью которого Вы поймете данный тезис. В этом примере мы будем вычислять квадратный корень из числа и возвращать значение в виде параметра, а сама функция будет возвращать *Истина*, если удалось вычислить, и *Ложь*, если нет.

Напишите в модуле формы следующую функцию:

```
Функция ВычеслитьКвадратныйКорень(НазваниеЧислаПодКорнем, ЧислоПодКорнем,
КореньЧисла)

    Если ЧислоПодКорнем < 0 тогда
        Сообщить("Нельзя вычислять корень из отрицательного числа");
        Возврат Ложь;
    иначе
        КореньЧисла = Окр(Sqrt(ЧислоПодКорнем),5);
        Сообщить("Корень из числа " + НазваниеЧислаПодКорнем + " = " +
КореньЧисла);
        Возврат Истина;
    КонецЕсли;

КонецФункции
```

Листинг 3.4.19

Опять создадим новую команду «*ВычислитьКореньЧислаФункцией*» и в её обработчике напишем следующий код:

```
&НаКлиенте
Процедура ВычислитьКореньЧислаФункцией(Команда)
    Перем А,Б;

    Если ВвестиЧисло(А, "Введите число А") Тогда
        ВычеслитьКвадратныйКорень("А",А,Б);
    КонецЕсли;
КонецПроцедуры
```

Листинг 3.4.20

Разберем данный пример. Наша функция имеет три параметра: название числа под корнем, само число, которое будет под корнем, и корень из этого числа. Если число под корнем меньше нуля, то мы возвращаем *Ложь*, соответствующее сообщение и ничего не считаем, а если нет - то возвращаем *Истина*, вычисляем число и выводим соответствующее сообщение.

В основной процедуре мы задаем данную функцию в виде процедуры. Поскольку нам в этом случае неинтересно, что она вернет, нам важно знать, какое будет число *Б*.

К примеру, мы можем использовать то, что возвращает эта функция для дальнейшей работы с числом *Б*, по аналогии как мы используем функцию *ВвестиЧисло*. Теперь сами измените данный пример и возведите в квадрат полученное число *Б*, если оно было вычислено.

На этом мы закончим нашу интересную и огромную часть про функции и процедуры. Теперь Вы самостоятельно сможете создавать любые процедуры и функции, которые Вам необходимы. В этой части мы изучили почти все, что необходимо для дальнейшей работы.

Остался только момент про экспорт процедур и функции. Этот вопрос мы будем проходить в четвертой главе.

Часть 5. Попытка…Исключение

В заключительной части третьей главы мы рассмотрим очень нужный и полезный оператор ***Попытка… Исключение***. В чем суть этого оператора? Как Вы уже заметили из предыдущих примеров, в ходе выполнения программы могут возникать ошибки, такие как деление на ноль, корень из отрицательного числа и прочие. Мы эти ошибки обходили конструкцией *Если...Тогда*, но не всегда возможно применить данную конструкцию, чтобы обойти исключительную ситуацию. Поэтому в языке программирования 1С существует оператор *Попытка…Исключение.*

Рассмотрим его синтаксис.

```
Попытка
//операторы попытки
Исключение
//операторы исключения.
КонецПопытки
```

Разберем данный синтаксис.

Ключевое слово *Попытка* открывает список операторов, выполнение которых может привести к исключительной ситуации, все операторы между ключевыми словами *Попытка* и *Исключение* это *Операторы попытки*.

Ключевое слово *Исключение* открывает список операторов, которые будут выполняться в том случае, если вызвана исключительная ситуация операторами попытки. Т.е. если во время выполнения программы один из операторов попытки вызвал ошибку выполнения программы (исключительную ситуацию), то выполнение данного оператора прерывается и управление передается на первый *Оператор исключения*. Причем обращаю Ваше внимание, что управление будет передано и в том случае, если исключительную ситуацию вызвали функции и процедуры, разработанные программистом и применяемые в качестве операторов попытки. Естественно, что если ошибка произошла в процедуре или функции, то ее выполнение будет прервано.

Создайте обработку, форму, команду на форме и в обработчике команды сделайте следующий пример:

```
&НаКлиенте
Процедура ВыполнитьКоманду(Команда)
    Перем А;

    Если ВвестиЧисло(А,"Введите число А") тогда
        Попытка
            КореньЧислаяА = Sqrt(А);
            Сообщить("Квадратный корень числа А = " + Окр(КореньЧислаяА,5));
        Исключение
            Сообщить("Попытка извлечь корень из отрицательного числа");
        КонецПопытки;
    КонецЕсли;

КонецПроцедуры
```

Листинг 3.5.1

Разберем данный пример.

Как обычно, вводим интерактивно число, после чего пытаемся вычислить квадратный корень данного числа. Используем ключевые слова *Попытка, Исключение, КонецПопытки*. Эти ключевые слова всегда должны быть в наличии и представлены в такой очередности, как мы видим. Таким образом, в Вашем примере вычисление квадратного корня и вывод сообщения являются операторами попытки. Между словом *Исключение* и *КонецПопытки* располагаются операторы исключения. На эти операторы перейдет программа после вызова исключения. Операторы попытки, следующие после строки, когда, где была инициализирована ошибка, исполняться не будут.

Запустите вновь созданную обработку и посмотрите, как она работает.

Если Вы все сделали правильно, то увидите, что никакой ошибки не возникает, просто вышло сообщение о корне из отрицательного числа. А что если нам нужно, чтобы все-таки ошибка вышла? Для этого есть оператор ***ВызватьИсключение***.

Переделайте Ваш пример.

```
&НаКлиенте
Процедура ВыполнитьКоманду(Команда)
    Перем А;

    Если ВвестиЧисло(А,"Введите число А") тогда
        Попытка
            КореньЧислаяА = Sqrt(А);
            Сообщить("Квадратный корень числа А = " + Окр(КореньЧислаяА,5));
        Исключение
            Сообщить("Попытка извлечь корень из отрицательного числа");
            ВызватьИсключение;
        КонецПопытки;
    КонецЕсли;

КонецПроцедуры
```

Листинг 3.5.2

В данном примере после операторов исключения пишем оператор *ВызватьИсключение*.

Сохраните, запустите и введите отрицательное число.

Должно выйти следующее окно:

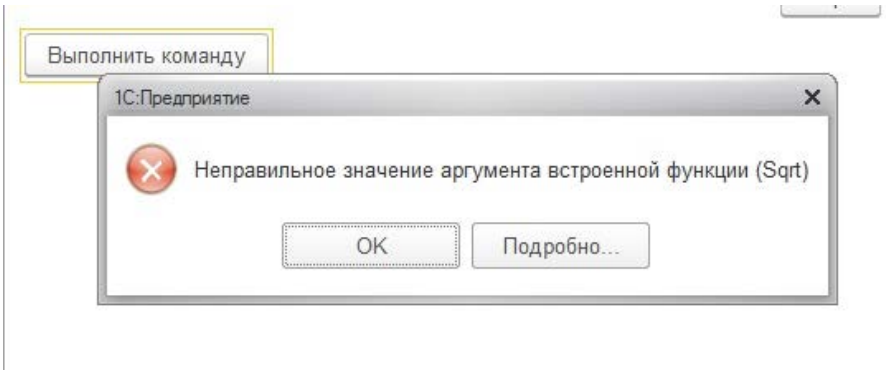

Рис. 3.5.1

Как видите, оператор исключения отработал, и сразу вышла ошибка о неправильном параметре.

Оператор *ВызватьИсключение* особенно удобно использовать на этапе отладки, когда исключительная ситуация может сложиться в различных случаях и необходимо точно знать, где она произошла и в результате чего она произошла.

Теперь сделаем пример, когда исключительная ситуация будет происходить внутри функции, которая используется в качестве оператора попытки.

Данный пример будет следующим: вычислим квадратный корень некоторого числа с округлением, потом возведем в квадрат и посчитаем погрешность: вычтем из заданного числа получившийся квадрат.

Напишите в модуле формы следующую функцию:

```
&НаКлиенте
Функция ВычеслитьИОкруглитьКорень(ЧислоПодКорнем,Точность = 5)

    КореньЧисла = Окр(Sqrt(ЧислоПодКорнем),Точность);

    Возврат КореньЧисла;

КонецФункции
```

Листинг 3.5.3

Создадим команду *«ПолучитьПогрешностьОкругления»* и в обработчике команды напишем следующий код:

```
&НаКлиенте
Процедура ПолучитьПогрешностьОкругления(Команда)
    Перем А;

    Если ВвестиЧисло(А,"Введите число А") тогда
        Попытка
            А2 = Pow(ВычеслитьИОкруглитьКорень(А,10),2);
            ПогрешностьОкругления = А - А2;
```

```
            Сообщить("Погрешность округления = " + ПогрешностьОкругления);
        Исключение
            Сообщить("Ошибка при вычислении");
        КонецПопытки;
    КонецЕсли;

КонецПроцедуры
```

Листинг 3.5.4

Сохраните и запустите Вашу обработку.

Если сделано все правильно, то Вы увидите, что при введении положительного числа выходит результат, а в случае отрицательного числа – сообщение об ошибке.

Разберите данный пример самостоятельно.

Сейчас мы рассмотрели оператор *Попытка…Исключение*. Теперь Вы сможете избегать неприятных ситуаций с возникновением ошибок. Особенно желательно использовать данный оператор в критических моментах выполнения кода, например, при записи чего-либо в базу.

Резюме

На этом третья глава закончилась. В ней Вы научились работать с условиями, а также освоили два из трех операторов цикла. После данного урока Вы сможете создавать собственные процедуры и функции, а также использовать оператор *Попытка…Исключение* в своей практике программирования. Удачи в изучении языка!