

Глава 1. Описание языка СБиС++

Введение.....	2
Выполнение языка СБиС++.....	2
Структура программы.....	2
Зарезервированные ключевые слова	4
Выражения.....	4
Общие положения.....	4
Оператор присваивания.....	5
Базовые арифметические операции.....	5
Присваивание и арифметические операции.....	6
Операторы увеличения/уменьшения на 1.....	7
Отрицание.....	7
Приоритеты операций и круглые скобки.....	7
Операции с текстом.....	8
Операции с датой.....	8
Операции со временем.....	9
Типы данных	9
Простые типы данных	9
Объекты и контексты	11
Массивы	15
Переменные	16
Идентификаторы.....	16
Создание переменных.....	18
Глобальные и локальные переменные	19
Условные операторы	20
Общие положения.....	20
Операторы сравнения.....	20
Логические операторы.....	21
Оператор Если-иначе.....	22
Оператор ?.....	23
Оператор ВыборПо.....	23
Циклы	25
Цикл Пока.....	25
Цикл ДляВсех.....	26
Оператор Прервать.....	27
Оператор Вернуть.....	27
Объекты в СБиС++	27
Общий перечень объектов.....	27
Функции.....	28
Описание функции.....	28
Правила вызова функции.....	29
Оператор Вернуть.....	30
Транзакции в языке.....	30
Понятие транзакции.....	30
Роль транзакции при многопользовательском режиме работы СБиС++.....	31
Описание запуска транзакции в языке.....	32
Пример обработки транзакции в СБиС++.....	32

Введение

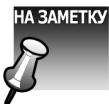
Выполнение языка СБиС++

Процесс выполнения программ на языке СБиС++ проходит в два этапа. На первом происходит разбор текста программы на составляющие (так называемые лексемы) и проверка корректности введённого текста. В результате первого этапа получается некоторый внутренний программный код, который уже и выполняется на втором этапе.



Поскольку язык СБиС++ - это всё-таки язык программирования, хоть и достаточно специфический, будем в дальнейшем текст на языке СБиС++ именовать «программа».

Язык СБиС++ является *интерпретируемым* языком. В частности, это означает, что код, полученный на этапе разбора, выполняется не непосредственно процессором, а некоторой виртуальной машиной.



Для особо любопытных: читайте про язык Java, там сделано похожее.

В связи с этим, чтобы уметь писать выражения на языке СБиС++, вам не нужно изучать особенности работы процессора и операционной системы, хотя и придётся понять логику работы этой виртуальной машины.

При написании простых выражений о машине выполнения можно вообще не задумываться – всё очень прозрачно. А вот если требуется выполнить что-то достаточно сложное, то тут понимание логики работы просто необходимо.

Соответственно, данная глава построена следующим образом: сначала идёт формальное описание конструкций языка СБиС++, потом описание машины выполнения.

Структура программы

Операторы

Программа на языке СБиС++ состоит из последовательности операторов и управляющих конструкций.

Оператор – это некоторое единичное действие, например, запись значения некоторого выражения в переменную или вызов функции.

Каждый оператор должен заканчиваться **точкой с запятой**. Таким образом, простейший текст программы на языке СБиС++ выглядит следующим образом:

```
Оператор1 ;  
Оператор2 ;  
Оператор3 ;  
...  
ОператорN ;
```

В языке СБиС++ конец строки *не является* признаком окончания оператора. То есть на одной строке можно располагать несколько операторов, каждый из которых должен заканчиваться точкой с запятой. И наоборот, один оператор может располагаться на нескольких строках.

Очерѐдность выполнения операторов и управляющие конструкции

В предыдущем примере операторы будут выполняться последовательно, то есть сначала будет выполняться «Оператор1», потом «Оператор2» и так далее. Но очень часто требуется выполнять в зависимости от тех или иных условий, ту или иную группу операторов. Часто требуется выполнить некоторую группу операторов несколько раз. Для реализации подобных действий в языке СБиС++ есть набор специальных управляющих конструкций, например, «если-то», «пока» и так далее. Все эти конструкции описаны ниже в этой главе.

Комментарии

Комментариями считается текст, начинающийся с символа «#» (решѐтка) и до конца строки. Этот текст просто игнорируется, так что здесь можно писать всё, что угодно.

```
Оператор1 ; # комментарий
```

При написании сложных программ комментарии – незаменимая вещь. Как показывает практика, без комментариев очень сложно бывает понять текст мало-мальски сложной программы, даже если она вами же и писалась.



Кроме того, при помощи комментариев можно временно отключать выполнение «кусков» программы, не удаляя их.

Чтобы поставить/снять комментарий во встроенном в СБиС++ редакторе, достаточно отметить нужный фрагмент текста и нажать <Ctrl+K>. Если же блок не отмечен, то по <Ctrl+K> знак комментария будет ставиться/сниматься в первой позиции текущей строки.

Зарезервированные ключевые слова

Следующий набор идентификаторов зарезервирован в языке СБиС++ и может использоваться *только* в конструкциях языка, описанных далее в этой главе.

Ключевое слово по-русски	Английский аналог
вернуть	return
выбор	case
выборпо	switch
длявсех	forall
если	if
и	&&
или	
иначе	else
пока	while
прервать	break
функция	func

Выражения

Общие положения

Выше были приведены несколько примеров простейших выражений языка СБиС++, которые объявляли переменные и присваивали им значения. Каждое выражение должно заканчиваться точкой с запятой «;». Выражение может содержать бесконечно много операторов и операндов, но не забывайте, что слишком длинные выражения трудны для понимания. Поэтому лучше разбивать их на более простые.

Оператор присваивания

Пожалуй, самый часто используемый оператор – это оператор присваивания. Его назначение – поместить указанное значение в указанное место (переменную или поле базы данных). Оператор присваивания записывается следующим образом:

КудаПомещать = ЧтоПомещать ;

То есть указывается, куда помещать значение (идентификатор переменной или поля базы данных), потом ставится знак «**=**» (знак равенства) и указывается, что помещать. Причём, если слева от знака равенства обязательно должен быть некоторый идентификатор, то справа может быть любое выражение. Например, такая запись некорректна:

2*2 = Ц; # так писать нельзя!

А вот такая строка вполне нормальна:

Б = 2*2;

Последняя строка читается не «Б равно два умножить на два», а «*присвоить* переменной Б значение выражения ”два умножить на два”».

Оператор присваивания в качестве результата своего выполнения возвращает получившееся значение переменной. Благодаря этому, а также тому факту, что несколько операторов присваивания выполняются справа налево, допустима такая запись:

А = Б = Ц = 4;

Выполняться эта строка будет так: значение 4 будет присвоено переменной Ц, получившееся значение переменной Ц будет присвоено переменной Б, а потом значение переменной Б – переменной А. В итоге во всех трех переменных будет лежать значение 4. Но здесь есть одно «но»: если переменные А, Б, Ц будут разных типов, то в А может попасть не совсем то, что попадёт в Ц, например:

число А, Ц;

деньги Б;

А = Б = Ц = 4.001;

Поскольку в этом примере Б – это деньги, то в А будет присвоено не 4.001, а 4.00. А в Ц будет лежать 4.001, то есть в А и Ц будут лежать *разные* значения. Отсюда правило: «**Не используйте последовательное присваивание для переменных разных типов!**».

Базовые арифметические операции



В языке СБиС++ поддерживаются следующие базовые операции:

Обозначение	Название	Пример
+	сложение	A+B
-	вычитание	A-B
*	умножение	A*B
/	деление	A/B
%	взятие остатка от деления	A%B

Сложение, вычитание, умножение и деление – это обычные арифметические операции, с ними всё ясно. Пояснений тут, пожалуй, требует только последняя операция – «взятие остатка от деления». Записывается она так: «**A%B**», а возвращает остаток от деления **нацело** A на B, например:

```
5 % 2 - результат 1
6 % 3 - результат 0
3.5 % 3 - результат 0.5
1.5 % 5 - результат 1.5
```

Операции «деление» и «взятие остатка от деления» не допускают, чтобы правый операнд был равен нулю. Если же такое случается, выдаётся сообщение об ошибке «**Произошло деление на ноль**» и выполнение программы на языке СБиС++ прерывается. Поэтому перед делением имеет смысл проверить правый операнд, например, так:

```
ц = B!=0 ? A/B : 0;
```

Присваивание и арифметические операции

Следующие операции допускается записывать в краткой форме:

Операция	Короткая форма
A = A+B;	A += B;
A = A-B;	A -= B;
A = A*B;	A *= B;
A = A/B;	A /= B;
A = A%B;	A %= B;

То есть в операциях вида «увеличить (уменьшить, умножить, разделить) переменную на некоторое значение» знак арифметической операции и знак присваивания можно совмещать. Кроме того, что такая запись короче, она ещё и быстрее выполняется, поскольку делается не две операции (сложение и присваивание), а только одна.

Операторы увеличения/уменьшения на 1

Кроме того, для операции «увеличить/уменьшить переменную на 1» есть специальные операторы «++» и «--» соответственно:

++A; # увеличить A на 1
--A; # уменьшить A на 1

Таким образом, следующие операции эквивалентны:

A = A+1;	A += 1;	++A;
A = A-1;	A -= 1;	--A;

Операторы «++» и «--» допускается указывать как до имени переменной, так и после. Разница тут в возвращаемом значении: если оператор указан *до* имени переменной, то возвращается значение уже увеличенное/уменьшенное на 1. Если же оператор указан *после* имени переменной, то возвращается исходное значение переменной. Например:

A = 1000;
Сообщить(++A); # будет сообщено 1001
Сообщить(A++); # тоже будет сообщено 1001

Отрицание

Есть ещё один оператор «отрицание», пишется так:

-A

В результате, положительное число становится отрицательным, а отрицательное - положительным.

Приоритеты операций и круглые скобки

В языке СБиС++ принят следующий порядок выполнения операций:

1. -
2. --, ++
3. +, -
4. /, %
5. =, +=, -=, *=, /=, %=.

Для изменения очерёдности выполнения операций используются круглые скобки:

пСумма = (100-10) * (20+5);



Операции с текстом

Для текстовых данных определена только операция сложения (конкатенации) строк:

```
пТекст1 = "СБиС++" ;  
пТекст2 = " for ever" ;  
Сообщить ( пТекст1 + пТекст2 ) ;
```

Допустим и оператор «+=»:

```
пТекст1 = "СБиС++" ;  
пТекст1 += " for ever" ;  
Сообщить ( пТекст1 ) ;
```

При сложении текстового значения и значения другого типа, например, числового, второе трактуется тоже как текст. То есть такое выражение:

```
Сообщить ( "Сумма = " + Сумма ) ;
```

Сообщит примерно следующее – «Сумма = 123.45».

Операции с датой

Для даты определены операции суммирования с числом, вычитания из даты числа и вычитание одной даты из другой.

При суммировании даты с числом и вычитании из даты числа число трактуется, как количество дней. То есть в итоге получаем дату, которая позже в случае суммирования или раньше в случае вычитания исходной даты на указанное число дней:

```
пДата1 = 08.03.2003 ;  
пДата2 = пДата1 + 10 ; # в пДата2 - 18.03.2003  
пДата2 = пДата1 - 10 ; # в пДата2 - 26.02.2003
```

В результате вычитания одной даты из другой получается число – разница в днях между этими двумя датами:

```
пДата1 = 08.03.2003 ;  
пДата2 = 26.02.2003 ;  
пДни = пДата1 - пДата2 ; # в пДни - 10
```

Аналогично работают и операторы «+=» и «-=».

Операции со временем

Для времени аналогично определены операции суммирования с числом, вычитания из времени числа и вычитание одного времени из другого.

При суммировании времени с числом и вычитании из времени числа число трактуется, как количество секунд. То есть в итоге получаем время, которое позже в случае суммирования или раньше в случае вычитания исходного времени на указанное число секунд:

```
пВремя1 = 14:04 + 60;      # в пВремя1 - 14:05:00
пВремя2 = пВремя1 - 600; # в пВремя2 - 13:55:00
```

В результате вычитания одного времени из другого получается число – разница в секундах между этими двумя временами:

```
пСекунды = 13:55 - 12:55; # в пСекунды - 3600
```

Аналогично работают и операторы «+=» и «-=».

Типы данных

Язык СБИС++ позволяет манипулировать данными определенных типов. Тип данных – это характеристика набора данных, например: текст, число, дата или что-то еще. Тип определяет диапазон возможных значений данных и допустимые операции над этими значениями.

Различаются следующие типы данных:

- **простые** – текст, двоичные данные, целое, число, деньги, дата, время;
- **составные** - массив, объект и др.

Каждый из этих типов будет подробно рассматриваться чуть ниже в соответствующем разделе этой главы.

Простые типы данных

Название типа	Описание типа
------------------	---------------



Текст	<p>Набор любых символов, кроме символа с кодом ноль. Количество символов может быть не более 255. Данные типа «текст» указываются в кавычках:</p> <p>”Это текстовая строка.”</p> <p>Текст, не содержащий ни одного символа, называется <i>пустой строкой</i> и указывается двумя кавычками, идущими подряд: “”</p> <p>Чтобы указать в самом тексте кавычки, перед ними ставится обратная косая черта:</p> <p>”Скажи – \”СБиС++\”.”</p> <p>Соответственно, чтобы указать в тексте обратную косую черту (например, при записи полного имени файла), их указывается две подряд:</p> <p>”C:\SBIS\TEST.TXT”</p>
Двоичные-Данные	Это текст неограниченной длины.
Целое	Целые числа, не содержащие дробную часть.
Число	<p>Этот тип включает числа, содержащие целую и дробную часть. В качестве разделителя целой и дробной части используется <i>точка</i>, а не запятая:</p> <p>-123.4567</p> <p>Числовые величины – это вещественные числа в диапазоне от $-1.7 \cdot 10^{308}$ до $1.7 \cdot 10^{308}$. Точность чисел не может быть более восьми символов после запятой.</p>
Да/Нет	Логический тип, допускающий одно из двух возможных значений: «Да» или «Нет».
Деньги	Поскольку СБиС++ – это прежде всего бухгалтерская программа, для удобства работы с денежными величинами введён специальный тип данных – «деньги». Этот тип мало, чем отличается от типа «число». Единственное отличие – величины типа «деньги» всегда округляются до двух знаков после запятой.

Дата	<p>Величины типа «дата» указываются либо в виде «дд.мм.гг» (то есть два числа – день, точка, два числа – месяц, точка, два числа – год), либо в виде «дд.мм.гггг» (тоже самое, но год указывается четырьмя числами). Например: 01.01.06; 01.01.2006</p> <p>Обратите внимание, что <i>не допускается</i> указывать одну цифру вместо двух, использовать вместо точки другой разделитель и так далее. Существует специальное значение даты – нулевая дата. В комплексе СБиС++ в полях ввода и в таблицах такая дата отображается пробелами, а в языке так: 00.00.00</p>
Время	<p>Величины типа «время» указываются либо в виде «чч:мм:сс» (то есть два числа – часы, двоеточие, два числа – минуты, двоеточие, два числа – секунды), либо в виде «чч:мм» (тоже самое, но без секунд). Например: 12:00:10; 12:00</p> <p>Обратите внимание, что <i>не допускается</i> указывать одну цифру вместо двух, использовать вместо двоеточия другой разделитель и так далее.</p>

Объекты и контексты

При работе в программе мы сталкиваемся с такими понятиями как организация, документ, товар, основное средство и т.д. Каждое из таких понятий характеризуется определенным набором данных и свойств. Например, у документа есть дата, номер и сумма. Такой набор данных, связанных одним понятием, мы будем называть **объектом**. В большинстве своем все наши операции и выражения будут обрабатывать либо данные объектов, либо сами объекты.

Приведем пример простейшей операции выдачи денег в подотчет:

Проводка (Д71, Лицо1, Сумма, К50_1);

Под переменной *Сумма* в данном случае мы подразумеваем сумму документа, для которого выполняется проводка. Иными словами можно сказать, что *Сумма* - это поле данных или переменная объекта *Документ*. Очевидно, что у каждого документа есть еще много других полей (переменных). В этот набор входят, по меньшей мере, все те данные, которые мы заполняли, создавая или редактируя документ.

Как обращаться к переменным объекта? Для этого используется следующая форма записи:

Документ . Сумма



Здесь *Документ* - это имя объекта, *Сумма* - имя переменной объекта, «.» (точка) между ними - символ уточнения - указывает на обращение к переменной объекта. Наш пример с функцией *Проводка* можно записать в следующем виде:

```
Проводка (Д71, Документ.Лицо1, Документ.Сумма,  
к50_1);
```

И эта, и ранее приведенная запись являются правильными. Заметим, что в первом случае можно не уточнять, что *Сумма* относится к документу, и использовать, таким образом, короткую форму записи. Функция *Проводка* выполняется в операции, закрепленной за документом, и когда мы обращаемся к переменной *Сумма*, то достаточно очевидно, что мы обращаемся именно к сумме документа (к чему же еще). В данном случае, мы можем говорить о том, что объект *Документ* является **контекстным** или умалчиваемым (принимаемый по умолчанию), а обращение к любой переменной подразумевает обращение к переменной этого объекта.

В ходе выполнения операции возможны случаи, когда вместо документа будет установлен другой контекстный объект. Например, при переборе строк накладной текущим контекстом будет объект типа строка документа - мы называем его *Наим* (от слова наименование). Кроме того, есть специальная функция - *Установить* - позволяющая Вам самостоятельно установить объект, который будет считаться текущим контекстом.

Вернемся к нашему примеру. Указывая аналитический признак по дебету 71-го счета, мы написали - *Лицо1*. Что такое *Лицо1*? С точки зрения оформления документа - это сотрудник, которому выдаются деньги. С точки же зрения правила операции это объект, точнее даже объект аналитического учета. Помня о том, что в документе у нас могут быть два лица - *Лицо1* и *Лицо2* - можно сказать, что в объекте типа Документ имеются два других объекта *Лицо1* и *Лицо2*, т.е. имеются два подобъекта.

Предположим теперь, что 71 счет разбит у нас на субсчета, в соответствии с отделами в нашей организации. Выдавая деньги в подотчет, мы должны будем сделать проводку не просто на 71 счет, а на субсчет того отдела, к которому относится данный сотрудник. Как решить эту задачу?

При занесении данных в справочник сотрудников будем указывать в поле данных 'Субсчет' тот субсчет 71-го счета, к которому относится данный сотрудник. Тем самым мы фактически определим переменную *Субсчет* в объекте *Лицо1*, и в дальнейшем сможем использовать ее при составлении правила операции.

Проводка (Лицо1.Субсчет, Лицо1, Сумма, К50);

Запись *‘Лицо1.Субсчет’* означает, что мы обращаемся к полю *Субсчет* объекта *Лицо1*. Уточнение в данном случае является обязательным, так как объект *Лицо1* в отличие от объекта *Документ* не является контекстным. Написав просто переменную *Субсчет*, мы можем получить сообщение об ошибке, так как контекстный объект *Документ* в общем случае не содержит такой переменной.

Если записать предыдущий наш пример без использования контекстного объекта, то правило будет выглядеть следующим образом:

Проводка (Документ.Лицо1.Субсчет, Документ.Лицо1, Документ.Сумма, К50);

Остановимся более подробно на записи *Документ.Лицо1.Субсчет*. Что мы делаем в данном случае - берем объект *Документ*, у него берем подобъект *Лицо1*, а у этого подобъекта берем поле *Субсчет*. Если бы у объекта *Лицо1* были бы подобъекты, то эту цепочку можно было бы продолжить. Здесь очень важно понять, что везде, где у нас определен объект типа документ, всегда можно обратиться к его переменным и подобъектам. И у этих подобъектов можно обратиться к их переменным и подобъектам и т.д.

Приведем еще один пример обращения к полям объекта. В правиле по закрытию 44 счета можно написать следующую операцию.

```
ДляВсех (Лиц ("44"))
{
  С = СКД44 / .ОК46;
  Проводка (Д46_10, С * .ОК46_10, К44, Лицо);
  Проводка (Д46_20, С * .ОК46_20, К44, Лицо);
}
```

В данном случае мы перебираем всех лиц, имеющих сальдо или обороты по 44-му счету. При переборе в операцию вносится контекстный объект *Лицо*. Псевдопеременная *СКД44* (сальдо конечное дебетовое по 44-му счету) в случае, когда объект типа *Лицо* является контекстным, трактуется как сальдо по 44-му счету в разрезе данного лица. Аналогичная ситуация и по 46-му счету. Как же в таком случае получить общее сальдо или в данном случае обороты по 46 счету? Для этого используется запись вида *‘.ОК46’*. Точка в начале записи говорит о том, что мы обращаемся к базовому значению переменной, вне текущего контекста. Запись с точкой перед именем переменной может использоваться и в других случаях. Не вдаваясь в сложные теоретические обоснования, попробуем пояснить это на примерах:

Вариант 1
ДляВсех (Наименований)

Вариант 2
ДляВсех (Наименований)



```

{ A += СуммаСебест; }           { .A +=
СуммаСебест; }
Сообщить (A);                   Сообщить (A);

```

В варианте 1 при выполнении функции '*Сообщить*' будет выдано сообщение об ошибке, в связи с отсутствием переменной '*A*', в варианте 2 все пройдет нормально. В чем разница? В точке перед именем переменной '*A*' при суммировании в цикле. В первом случае переменная создается внутри цикла и при его завершении будет уничтожена, и следовательно в момент выполнения функции '*Сообщить*' ее уже не будет. Во втором случае переменная '*A*' создается в базовом контексте, т.е. как бы вне цикла, и соответственно не уничтожается при его завершении.

Набор объектов, с которыми Вам придется оперировать в программе, строго фиксирован и достаточно невелик. Наиболее типичными среди них являются объекты типа *Лицо*, *Документ* и *Наим*. В дальнейшем мы приведем более подробное описание всех объектов, используемых в системе.

Алгоритм сравнения объектов

Как сравниваются два объекта в СБИС++, например, в выражении `Объект1 == объект2`.

```

Если оба объекта нулевые (равны «Нет», пустые),
    то они равны.
Иначе Если один из них нулевой, а другой - нет,
    то они не равны
Иначе (оба не нулевые)
    Если оба объекта не имеют записей, то они равны
    Иначе Если один объект имеет запись, а другой -
нет, то они не равны
    Иначе (оба объекта имеют записи)
        Если записи имеют ОДНОГО И ТОГО ЖЕ хозяина и
ОДИНАКОВЫЙ адрес, то они равны
        Иначе
            они не равны

```

Поясним на примере действие этого алгоритма. В выражении:

```

перем объект = Выборка ("Организации");
объект.Загрузить (0x12345);
объект.МоёДополнительноеПоле = "1234";

```

записью будет запись выборки (а *МоёДополнительноеПоле* в эту запись НЕ войдет). Хозяином записи будет выборка "Организации". При сравне-

нии важно, чтобы записи принадлежали не просто к одинаковой выборке, а к **ОДНОМУ** экземпляру выборки. Например:

```
перем оТаблица1 = Выборка ("Организации" );
перем оОбъект1 = оТаблица1.Запись ( );
оОбъект1.Загрузить ( 0x26В );
перем оОбъект2 = оТаблица1.Запись ( );
оОбъект2.Загрузить ( 0x26В );
перем оТаблица2 = Выборка ("Организации" );
перем оОбъект3 = оТаблица2.Запись ( );
оОбъект3.Загрузить ( 0x26В );
Сообщить ("Равны ли оОбъект1 и оОбъект2? " +
(оОбъект1 == оОбъект2) );
Сообщить ("Равны ли оОбъект1 и оОбъект3? " +
(оОбъект1 == оОбъект3) );
```

В результате получим:

- Равны ли оОбъект1 и оОбъект2? – **Да**;
- Равны ли оОбъект1 и оОбъект3? - **Нет**.

Важно отметить, что в СБИС++ два экземпляра одной таблицы всегда равны, т.е. замена в этом примере функции **Выборка** на **Таблица** вызвала бы ответ «Да» и во втором случае.

Массивы

Иногда возникает необходимость оперировать не отдельными переменными, а целым набором переменных. Такие наборы переменных принято называть **массивами** данных. В свою очередь значения, хранящиеся в массиве, называются **элементами** массива.

Фактически массивы – это переменные, хранящие в себе не одно конкретное значение, а набор других переменных. Так же как и переменные, каждый массив именуется.

Создание массива в программе выглядит следующим образом:

```
перем мМассив [КоличествоЭлементов] ;
```

Для доступа к значениям, хранящимся в массиве используется оператор [] (квадратные скобки), при этом номер элемента массива именуется **индексом**, а начинается нумерация с **1**. Количество элементов – общее число элементов массива или **размер** массива. Если на этапе описания массива не возможно предугадать его размер, то в качестве начального количества

элементов можно указать 0, а при дальнейшей работе увеличить это значение просто обращаясь к элементам массива с нужным индексом.

```
переменная mMac[0]; # создали массив нулевого размера
ЧислоЭлем = 5;
Номер = 1;
Пока (Номер <= ЧислоЭлем)
{
    mMac[Номер] = Номер; # присваиваем элементу
    значения
    Номер++;
}
```

В результате выполнения данного примера мы получаем массив mMac размера 5, значение каждого элемента которого равно его индексу. В последующем, чтобы получить доступ к любому элементу, указываем его индекс в квадратных скобках после имени массива.

```
A = mMac[3];
```

Любой элемент массива сам может быть массивом. В общем случае такие массивы называются *многомерными*. Для того, чтобы получить доступ к элементу внутреннего массива, нужно указать его индекс в квадратных скобках через запятую после индекса основного элемента.

```
A = mМассив[3, 1]; # переменная A содержит значение
первого элемента внутреннего массива, расположенного
под индексом 3.
```

Создать массив можно так же с помощью функции «Массив», например:

```
mMac = Массив("а", "б", "в"); # mMac[1] = "а",
mMac[2] = "б", mMac[3] = "в".
```

Для более эффективной работы с массивами есть функции «Вставить», «Заполнить» и т.д. Подробнее об этих функциях рассказывается в последующих главах данного справочника.

Переменные

Идентификаторы

Переменная – это участок памяти для хранения данных, который имеет собственное имя. Фактически идентификатор – это имя, обозначающее переменную, функцию или некоторую конструкцию языка.

В языке СБиС++ существует два вида идентификаторов, назовём их условно «правильные» и «неправильные».

Термины «правильный» и «неправильный» введены исключительно для облегчения изложения. С точки зрения программы разницы между двумя типами идентификаторов нет. Просто «правильный» идентификатор больше похож на идентификаторы в традиционных языках программирования, а аналога «неправильного» в большинстве языков программирования просто нет.

«Правильные» идентификаторы

«Правильный» идентификатор – это непустой набор **алфавитно-цифровых** символов и символов «_» (подчёркивание) и «@». В идентификаторах могут использоваться как английские, так и русские буквы. Примеры «правильных» идентификаторов:

```
имя_переменной
Сумма2
ндс20
20ф
```

Обратите внимание на последний пример. В отличие от традиционных языков программирования в языке СБиС++ допускаются идентификаторы, начинающиеся с цифры. Нужно только учитывать, что сначала идёт попытка разобрать такую строку, как число, дату или время. И только если разобрать не удалось, строка трактуется, как идентификатор. То есть «20» - это число, а «20Ф» - это уже идентификатор.

«Неправильные» идентификаторы

«Неправильный» идентификатор – это набор *любых* символов, заключённый в апострофы, например:

```
' Полное название '  
' Сумма '  
' Иванов Сидор Петрович '  
' 123.44 '
```

Обратите внимание, на последний пример. В нём записан именно идентификатор, а не число (виной всему апострофы, не будь их, было бы число).

Рассмотрим ещё один пример:

```
' Полное название '  
"Полное название"
```

В первой строке указан **идентификатор**, а во второй – **текст**. То есть первая строка будет рассматриваться, как имя некоторой конструкции языка (например, если это поле базы данных, то будет извлекаться значение этого поля). А вторая строка – это именно текст «Полное название» и больше ничего.

«Неправильные» идентификаторы были введены в основном для адресации к полям таблиц, поскольку имена многих полей содержат пробелы и другие «неправильные» символы. Вводить дополнительные «неправильные» идентификаторы (например, именовать так свои переменные), в общем-то, не рекомендуется.

Для большинства идентификаторов в языке СБиС++ не важно, в каком регистре они записаны, то есть имена «моясумма», «МояСумма» и «МОЯСУММА» будут идентичны. Исключение составляют лишь имена полей таблиц и выборки из базы данных, их имена нужно указывать *точно* так, как они записаны в базе данных и никак иначе. Имена же переменных, функций, ключевых слов и так далее можно указывать буквами в любом регистре.

Создание переменных

Переменную любого типа принято объявлять следующим образом:

перем А;

В этом случае будет создана временная с именем «А» неопределённого типа. Данный способ определения переменной можно совмещать с присвоением начального значения:

перем А=11290;

Существует и более простой способ определения переменной, например:

А=100;

Но такой способ определения переменной нежелателен. Потому как может порождать конфликтные ситуации в программе.

Объясним на примерах:



Пример 1:

А=100;

ДляВсех (Записей (НазваниеВыборки))

{

...

А++;

```

...
}
...
Сообщить (А) ;# в результате А будет равно
(100+количество записей выборки) .

```



```

Пример 2:
A=100;
ДляВсех (Записей (НазваниеВыборки) )
{
  ... перемА;
    A++;
  ...
}
...
Сообщить (А) ;# в результате А будет равно 100.

```

Как видим, результаты выполнения программного кода будут отличаться. В первом случае, и внутри цикла **ДляВсех(Записей)** и перед вызовом функции определяется одна и та же переменная А. Во втором случае, это две разные переменные: переменная А, определённая внутри цикла, будет уничтожена при выходе из него.

Глобальные и локальные переменные

Переменные, описанные в функции, являются **локальными**. Они могут использоваться только в теле функции. При попытке использования этой переменной вне функции, она будет определена заново. Локальные переменные создаются при вызове функции и уничтожаются по ее завершении.

Сама программа, по сути, тоже является функцией. Так что ее переменные тоже являются локальными. Они не видны из внутренних функций. Для того, чтобы создать **глобальные** переменные, их нужно описать определенным образом:

```
общПерем ГлобПеременная ;
```

Глобальные переменные будут видны из любого участка программы, в том числе и из написанных функций.

Еще одно отличие глобальной переменной от локальной: глобальная переменная, определенная один раз, не инициализируется на протяжении всего выполнения программы:

```
ОбщПерем м = 1 ;
```



```
Сообщить ("ОбщПерем м =", м);
...
ОбщПерем м = 2;
Сообщить ("ОбщПерем м =", м);
```

Чтобы её инициализировать, необходимо написать следующий код:

```
ОбщПерем м = 1;
Сообщить ("ОбщПерем м =", м);
...
ОбщПерем м;
м = 2;
Сообщить ("ОбщПерем м =", м);
```

Условные операторы

Общие положения

Условные операторы отвечают за то, будет ли выполнен некий участок программы в зависимости от определенного условия. Это условие записывается с помощью логического выражения, содержащего операторы сравнения, и результатом которого является одно из двух значений: «**истинно**» или «**ложно**».

В языке СБисС++ нет специального выделенного типа данных для значений типа «истинно-ложно», просто принято следующее правило:

- Любое **ненулевое** значение трактуется, как «**истинно**». Соответственно **нулевое** значение трактуется, как «**ложно**».
- Для текста нулевым значением является пустая строка. Для дат – нулевая дата.

Операторы сравнения

Допустимы следующие операторы сравнения:

Обозначение	Название	Пример
==	равно	A==B
!=	не равно	A!=B
>	больше	A>B
<	меньше	A<B



\geq	больше или равно	$A \geq B$
\leq	меньше или равно	$A \leq B$

Результатом работы оператора сравнения будет число **1**, если сравнение верно (например, для оператора «равно» указанные значения равны, а для оператора «не равно» указанные значение не равны). В противном случае результатом будет число **0**.

Приоритет операторов сравнения меньше, чем у арифметических операций. То есть сначала будут выполняться арифметические операторы, а потом уже операторы сравнения.

Сравнение текстовых значений осуществляется посимвольно, то есть:

```
"АБГ" == "АБ" # результат - 0
```

```
"АБГ" > "АБ" # результат - 1
```

При сравнении дат большей считается более поздняя дата:

```
08.03.03 > 01.01.01; # результат - 1
```

Если одна из сравниваемых дат – нулевая дата, то и сравнение «больше», и сравнение «меньше» верно:

```
08.03.03 > 00.00.00; # результат - 1
```

```
08.03.03 < 00.00.00; # результат - 1
```

```
08.03.03 == 00.00.00; # результат - 0
```

```
08.03.03 != 00.00.00; # результат - 1
```

Если одно из сравниваемых значений типа «текст», то второе значение тоже преобразуется к тексту, и происходит сравнение двух текстовых величин посимвольно. Поэтому результатом следующего выражения будет **1**, то есть сравнение верно.

```
"10" < 2; # результат - 1
```

В случае сравнения числа с датой или временем, дата/время преобразуется в число (дата – в количество дней, время – в количество секунд). Далее выполняется сравнение двух чисел.

Логические операторы

Обозначение

Описание и примеры



!	<p>Логическое отрицание инвертирует логическое значение. То есть если было «истинно», станет – «ложно» и наоборот. Указывается символом «!» (знак восклицания) перед отрицаемым значением: !1;# результат – 0 !0;# результат – 1 !100;# результат - 0</p>
ИЛИ (либо)	<p>Этот бинарный оператор возвращает «истинно» (число 1), если хотя бы один из двух параметров имеет значение «истинно» (то есть ненулевое значение). Если же оба оператора имеют значение «ложно» (нулевое значение), то оператор «или» вернёт «ложно» (число 0). Например: 1 или 2 ;# результат – 1 10 или 0;# результат – 1 0 или "";# результат – 0 Допускается короткая запись оператора «или»: А В</p>
И (либо &)	<p>Этот бинарный оператор возвращает «истинно» (число 1) только, если оба параметра имеют значение «истинно» (то есть ненулевое значение). В остальных случаях оператор вернёт «ложно» (число 0). Например: 1 и 2 ;# результат – 1 10 и 0;# результат – 0 0 и "";# результат – 0 Допускается короткая запись оператора «и»: А & В</p>

Логические операторы выполняются после арифметических операторов и операторов сравнения. Благодаря этому, в выражениях подобных следующему скобки ставить, как правило, не требуется:

A+B<C и C<X+Y

Оператор Если-иначе

Данный оператор позволяет в зависимости от некоторого условия выполнить тот или иной набор действий.

Оператор имеет следующий вид:

```
Если (условное выражение)
{
    первый блок
}
[ Иначе
```

```
{
    второй блок
}
```

Эта запись отличается от предыдущих примеров наличием фигурных скобок. Они нужны только тогда, когда блоки после операторов **Если** и **Иначе** состоят из нескольких выражений.

Условное выражение оператора может содержать операторы И, ИЛИ, ! и др. и быть весьма сложным.

Оператор ?

Фактически этот оператор аналогичен оператору «Если/Иначе». Записывается он в следующей форме:

УсловноеВыражение?Выражение1 : Выражение2

Работает оператор так. Вычисляется значение условного выражения, если его значение отлично от нуля, то возвращается значение первого выражения, в противном случае возвращается значение второго выражения.

Применение этого оператора не так очевидно, как других элементов языка. Его удобно использовать в тех случаях, когда нужно получить либо то, либо другое значение в зависимости от какого-то условия. При умелом использовании этого оператора можно значительно сократить запись некоторых операций. Например, в инсталляционной базе данных встречаются такие строки:

```
Если (Оплата > Отгрузка)  
    Расчет = Отгрузка ;  
иначе  
    Расчет = Оплата ;
```

Если же использовать новый оператор все выглядит значительно проще и понятнее:

```
Расчет = Оплата>Отгрузка?Отгрузка:Оплата ;
```

Сразу видно, что цель этой строки рассчитать значение переменной «Расчет».

Оператор ВыборПо

Допустим, требуется выполнить, в зависимости от значения некоторой переменной или выражения, те или иные действия. Причем возможных ва-



риантов может быть больше двух. В этом случае удобнее использовать не оператор «Если», а оператор «ВыборПо».

В общем случае рассматриваемая конструкция языка выглядит следующим образом:

```
ВыборПо (ВыражениеВыбора)  
{  
    выбор Выражение1 , Выражение2 , . . . :  
        Оператор ;  
        . . .  
        Оператор ;  
    . . .  
    выбор ВыражениеN1 , ВыражениеN2 , . . . :  
        Оператор ;  
        . . .  
        Оператор ;  
    [ иначе  
        Оператор ;  
        . . .  
        Оператор ; ]  
}
```

Давайте рассмотрим каждый из приведенных элементов более подробно.

После ключевого слова *«ВыборПо»* в круглых скобках идет некоторое выражение. В простейшем случае это может быть просто имя переменной.

Результат выполнения выражения выбора последовательно сравнивается со значениями выражений, идущих после ключевых слов *«выбор»*. Если значение какого-то выражения совпало, то будут выполняться операторы после соответствующего двоеточия до следующего ключевого слова *«выбор»*, либо до слова *«иначе»*, либо до закрывающейся фигурной скобки. Обратите внимание, что после слова *«выбор»* может быть указано несколько выражений для сравнения, разделенных запятой.

Если же ни одно из значений не равно, будут выполнены операторы после ключевого слова *«иначе»* до закрывающейся фигурной скобки.

При написании оператора *«ВыборПо»* учтите, что, по крайней мере, должен быть хотя бы один блок операторов, начинающийся со слова *«выбор»*. Таких блоков может быть сколько угодно много. Блок *«иначе»* можно опускать.

На самом деле, чтобы реализовать подобную конструкцию, можно воспользоваться комбинацией операторов «Если/Иначе». Выглядело бы это примерно так:

```
Если (ВыражениеВыбора==Выражение1)
{
    Оператор; ... Оператор;
}
иначе если (ВыражениеВыбора==Выражение2)
{
    Оператор; ... Оператор;
}
...
иначе
{
    Оператор; ... Оператор;
}
```

Данная конструкция *практически* аналогична оператору «ВыборПо» за исключением маленького «но» - выражение выбора будет рассчитываться заново при каждом сравнении.

Циклы

Цикл Пока

Цикл Пока используется для того, чтобы выполнять некий набор действий, пока условие цикла остается истинным. Нужно помнить, что условие проверяется перед выполнением тела цикла. Так что, если оно изначально ложно, цикл не выполнится ни разу. Напротив, если условие всегда остается истинным, вы получите бесконечный цикл, но необходимость в них возникает довольно редко.

Цикл имеет следующий вид:

```
Пока (условие цикла)
{
    тело цикла
}
```

В фигурных скобках находится *тело цикла*, т.е. некоторый набор операций, который будет выполняться при каждом повторе цикла. Фигурные скобки необходимы в случае, если тело цикла состоит из нескольких вы-



ражений. Если для перебираемых элементов выполняется только одно выражение, то фигурные скобки можно опустить.

Цикл ДляВсех

Большинство используемых в программе операций обрабатывает объекты (речь о них пойдет ниже). Данный цикл перебирает объекты, входящие в состав более сложного объекта. Так строка документа имеет свой набор данных, отличающийся от набора данных самого документа, и можно сказать, что эта строка также является объектом, и тогда перебор наименований документа можно свести к перебору объектов типа строка документа (*Наим*) внутри объекта *Документ*. Из этих соображений можно сказать, что оператор ДляВсех всегда вносит в операцию новый объект данных, который будет считаться при выполнении тела цикла текущим контекстом.. Как только цикл завершается, внесенный объект удаляется и возвращается старый контекст.

В общем случае конструкция типа:

```
ДляВсех (Наименований ( ) ) # цикл по каждому  
наименованию  
{  
    Тело цикла  
}
```

называется циклом по наименованиям.

Разберем ее более подробно.

Условно все это выражение можно разбить на две части:

ДляВсех () – оператор перебора
Наименований () – функция, перебирающая строки

Оператор перебора ДляВсех() указывает на то, что далее следует перебор отдельных позиций. В круглых скобках, следующих за оператором, указывается функция, обеспечивающая выбор очередной строки.

Функция Наименований обеспечивает выбор очередной строки документа. Вообще функций такого типа в программе несколько. Они позволяют перебирать не только наименования накладной или счета, но и связанные документы (функции *Оснований()*, *Связей()*), объекты аналитического учета (функция *Лиц()*) и т.д.)

Оператор Прервать

Иногда бывает нужно прервать выполнение цикла. Именно для этого необходим оператор *Прервать*. Он позволяет закончить повторения и перейти к выражениям, следующим за телом цикла.

Синтаксис оператора очень прост:

Прервать ;

Оператор Вернуть

Синтаксис оператора очень прост:

Вернуть ;

Объекты в СБИС++

Общий перечень объектов

Объект	Где встречается и используется	Что содержит
Документ	Во всех операциях, связанных с документами	Данные о самом документе и два подобъекта Лицо1, Лицо2 и Лицо3
Лицо	Циклы типа ДляВсех(Лиц) Типовые журналы по лицам В справочниках лиц	Текущее обрабатываемое лицо
Лицо0	В любом месте	Вашу организацию, установленной в качестве основной в СБИС++
Лицо1	Во всех объектах типа <i>Документ</i>	Лицо, установленное в документе качестве Лица 1
Лицо2	Во всех объектах типа <i>Документ</i>	Лицо, установленное в документе качестве Лица 2
Лицо3	Во всех объектах типа <i>Документ</i>	Лицо, установленное в документе качестве Лица 3
ЛицоАвтор	Во всех объектах типа <i>Документ</i>	Лицо, изначально создавшее документ
ЛицоИзменил	Во всех объектах типа <i>Документ</i>	Лицо, изменившее документ

ЛицоП	Во всех объектах типа <i>Документ</i>	Лицо папки документов, в которой находится текущий документ
Наим	Циклы по наименованиям и списаниям	Данные о строке документа, подобъекты <i>Товар</i> или <i>Лицо</i> , в расходных накладных подобъект <i>Приход</i>
Пользователь	В любом месте программы	Текущий пользователь системы
Приход	Циклы по наименованиям и списаниям в расходных накладных	Данные о партии прихода, подобъект <i>Лицо</i> с данными о поставщике
Расход		
Связь	Циклы по связанным документам	Данные о связи и подобъект <i>Документ</i>
ОснСредство	Ведомости амортизации и переоценки ОС	Данные об амортизируемом или переоцениваемом ОС
Склад	Все складские документы, кроме счетов	Информацию о складе, установленном в документе
Товар	Циклы по наименованиям и списаниям В складской карточке	Данные о текущей карточке складского учета
Проводка	Циклы по проводкам	Атрибуты проводки
Сальдо	Циклы по субсчетам	Атрибуты субсчетов

Функции

Описание функции

Чтобы полностью овладеть всеми возможностями программирования в СБиС++, нужно изучить функции, которые можно вызывать из языка СБиС++. А для начала надо понять, что такое функция и как её вызывать в программе?

В качестве примера рассмотрим удержание кредита из зарплаты. При выполнении этой операции необходимо сформировать всего одну проводку с показателями:

Дебет 70 Кредит 76_7 по лицу полученная сумма

Текст программы на языке СБиС++ автоматического формирования такой проводки будет выглядеть следующим образом:

Проводка (Д70 , К76_7 , Лицо , Сумма) ;

При выполнении программы начинают просматриваться выражения. Когда встретится ключевое слово *Проводка*, в журнал операций необходимо будет поместить проводку, атрибуты которой (счета и сумма) указаны в круглых скобках после ключевого слова. Такие ключевые слова (а их в программе довольно большое количество) называются **именами** функций. За каждым именем скрывается целый набор действий, необходимых для достижения некоторого результата (в данном случае формирование проводки и занесение ее в журнал операций). Совокупность этих действий называется **функцией**. В круглых скобках, сразу после имени функции, располагаются исходные данные, необходимые для ее выполнения. Это - **параметры** функции. Они *отделены друг от друга* запятыми.

Правила вызова функции

При описании вызова функции необходимо учесть, что её имя должно быть уникальным в пределах программы. Для формирования имени функции действуют те же правила, что и для имен переменных. Каждая функция начинается с заголовка и перечисления параметров в круглых скобках через запятую, количество которых зависит от вызываемой функции, после следует тело функции:

```
функция МояФункция (Параметр1|Параметр2 [ ,  
Параметр3] ...)  
{  
    тело функции  
}
```

При описании синтаксиса функций некоторые параметры могут указываться в квадратных скобках – это *необязательные* параметры. При использовании функции такие параметры могут не указываться. Если параметры указаны через символ «|», то может быть указан либо один, либо другой параметр, но не оба вместе.

Списком параметров функции является набор переменных, значения которым будут переданы при вызове функции. Эти переменные могут иметь свои ни с чем не связанные имена, так как они будут использоваться только в теле функции. Вызов функции в программе будет выглядеть следующим образом:

МояФункция (А, Б, ...) ; # в качестве параметров будут переданы значения переменных А, Б и др.



Нередко будут встречаться и функции, у которых вообще нет параметров:

ВерсияПрограммы () ;

В этом случае указывать скобки после имени функции нет необходимости. Но всё же для сохранения стиля описания вызова функций в программе мы рекомендуем их указывать, пусть даже и пустые.

Функция может возвращать результат, который допускается использовать далее в программе.

Пользователь сам может оформлять кусок кода программы в виде функции. Вместо того, чтобы копировать часто используемый фрагмент в разные места программы, лучше написать функцию и вызывать ее, когда нужно и с разными параметрами.

Оператор Вернуть

Как уже говорилось, функция может возвращать результат выполнения тела функции. Отвечает за это оператор Вернуть. Он имеет следующий синтаксис:

Вернуть Результат ;

Здесь «Результатом» может быть переменная или выражение.

Чтобы вызвать функцию, возвращающую результат, используется следующий формат:

А = МояФункция (А, Б, ...) ;

Транзакции в языке

Понятие транзакции

Транзакция – это последовательность операций над базой данных, рассматриваемых программой как единое целое. Примерами таких операций над элементами БД является их удаление, добавление или сохранение.

При этом либо транзакция выполняется, и в программе фиксируются изменения базы данных, либо ни одно из этих изменений никак не отражается в состоянии БД. Графически этот процесс можно изобразить так:



Обработка транзакций гарантирует целостность информации в базе данных. Таким образом, транзакция переводит базу данных из одного целостного состояния в другое.

Роль транзакции при многопользовательском режиме работы СБиС++

Поддержание механизма транзакций – обязательное условие даже при однопользовательском режиме работы в СБиС++. Но понятие транзакции гораздо существеннее при многопользовательском режиме работы. То свойство, что каждая транзакция начинается при целостном состоянии БД и оставляет это состояние целостным после своего завершения, делает очень удобным использование понятия транзакции как единицы активности пользователя по отношению к БД. При соответствующем управлении параллельно выполняющимися транзакциями со стороны СУБД каждый пользователь может в принципе ощущать себя единственным пользователем системы СБиС++.



Но есть одна особенность, которой не следует пренебрегать: длительные операции в одну транзакцию заключать не следует. В этом случае, данные, к которым попытаются получить доступ другие пользователи, будут заблокированы, и им придётся ждать длительное время, пока транзакция выполнится и они получат доступ. Лучшим решением в этом случае будет сначала разделить операции, а потом последовательно запускать транзакции. Если удаётся добиться последовательного выполнения серии транзакций, то для каждого пользователя, по инициативе которого образована транзакция, присутствие других транзакций будет незаметно (если не считать некоторого замедления работы для каж-

дого пользователя по сравнению с однопользовательским режимом).

Описание запуска транзакции в языке

Для запуска транзакции в языке присутствует метод:

```
Транзакция (Запись)
{
    Последовательность операций над элементами БД на
    внутреннем языке СБиС++
}
```

Операциями над элементами БД, как мы уже упоминали, могут быть их добавление, удаление или сохранение.

Пример обработки транзакции в СБиС++

Рассмотрим на простейшем примере, как будут происходить изменения в базе данных при выполнении определённых действий над элементами. И увидим, что произойдёт, если эти же действия выполнить с использованием транзакции.

Возьмём справочник «**Организации и ЧП**». Заведомо известно, что в этом справочнике есть папка «**Банки**». В этой папке есть записи. Нужно, к примеру, отметить записи этой папки «плюсом», причём после отметки четвёртой записи отметка оставшихся записей должна прерваться. В программе это реализуется следующим образом:

```
oOrg = Выборка ("Организации") ;
пНазваниеРаздела = "БАНКИ" ;
oOrg . ПерейтиВРаздел (пНазваниеРаздела) ;
пСч=0 ;
Пока (Следующий (oOrg) )
{
    oOrg . Отметить ("+" ) ;
    пСч++;
    Если (пСч==4)
        Ошибка ("Прервать отметку" ) ;
}
```

В результате выполнения предложенной последовательности действий в базе данных произойдут изменения. А именно, в папке «**Банки**» справоч-

ника организаций будет отмечено только четыре записи, в то время как остальные останутся нетронутыми.

А теперь попробуем выполнить эти же действия с записями, запустив при этом сеанс транзакции. В программе это реализуется следующим образом:

```
oOrg = Выборка ("Организации" );
пНазваниеРаздела = "БАНКИ" ;
oOrg . ПерейтиВРаздел (пНазваниеРаздела) ;
пСч=0 ;
Транзакция (oOrg)
{
  Пока (Следующий (oOrg) )
  {
    oOrg . Отметить ("+" ) ;
    пСч++ ;
    Если (пСч==4)
      Ошибка ("Прервать отметку" ) ;
  }
}
```

В результате происходит следующее: перебираются записи в папке «Банки» и выполняется отметка записей до тех пор, пока не доберёмся до пятой записи. Срабатывает ошибка, и процесс отметки оставшихся записей прекращается. Произошло прерывание транзакции. В этом случае, происходит откат базы данных к исходному состоянию, т.е. изменения в базе данных, которые были сделаны до прерывания, будут отменены и база данных вернётся к состоянию на момент начала транзакции. В нашем случае, отмеченных записей в папке мы не увидим. При этом логическая целостность БД была сохранена.