# 5. ПРОЦЕДУРНОЕ ПРОГРАММИРОВАНИЕ

Научиться использовать локальные переменные, разрабатывать и использовать процедуры и функции с параметрами. Научиться разрабатывать и использовать программные модули.

Процедуры – наиболее важная часть CF Pascal (и многих других языков программирования) потому что они позволяют разбить программу на небольшие разделы для лучшего понимания. Каждый раздел в значительной степени самодостаточен, его можно изучать в отдельности. При использовании процедуры нет необходимости читать весь ее текст, обычно достаточно прочитать название процедуры или обобщающий комментарий.

В некотором смысле процедура реализует идею функции значения части программы.

Преимущество процедур над другими конструкциями CF Pascal в том, что они могут включать любой код и поэтому являются мощным средством для абстрагирования и обобщения. Более того, механизм параметров позволяет обобщить одну процедуру для решения целого класса задач, что позволяет сокращать трудоемкость разработки

**5.1. Процедуры с параметрами**

Процедуры могут быть объявлены с формальными параметрами, таким образом, что данные, сохраненные в состоянии выполнения, могут быть обработаны кодом процедуры, где параметры процедуры заменяются переменными программы.

Понятия: оператор процедуры, объявление процедуры, формальные параметры, фактические параметры, правило «все или ничего»

**5.1.1. Простые процедуры и параметры.**

Процедуры, описанные в главе 2 и их формальное значение, описанное в главе 6 позволяют объявить оператор BEGIN и потом вызывать его через процедурное выражение используя только идетификатор процедуры. Объявление добавляет идентификатор в состояние выполнения, значением которого является текст тела процедуры.

Например, объявление процедуры:

PROCEDURE SwitchOneTwo;

BEGIN

Hold := OneV;

OneV := TwoV;

TwoV := Hold;

END.

Может быть написано для обмена значений OneV и TwoV, разрушая значение Hold.

Идентификаторы параметров помещаются в заголовке процедуры и используются в тексте как «подставные лица», то есть они фактически только обозначают место, куда будут подставлены переменные, использованные при вызове процедуры в тексте программы.

Например, заголовок:

PROCEDURE SwitchThese2(VAR V1, V2, temp: CHAR)

имеет три *формальных* параметра, которые принимают символьные значения. Тело процедуры может использовать их, скажем, следующим образом.

PROCEDURE SwitchThese2(VAR V1, V2, temp: CHAR);

BEGIN

Temp := V1;

V1 := V2;

V2 := Temp;

END

Процедурные выражения могут использовать идентификатор процедуры, со следующими за ним *фактическими* параметрами, которыми могут быть любые переменные того же типа, что и формальные параметры, которые существуют в момент вызова процедурного выражения.

Например, если в программе объявлены переменные Hold, OneV и TwoV, то процедурное выражение

SwitchThese2(OneV, TwoV, Hold)

Выполнит обмен значений OneV и TwoV точно так же как это сделала бы процедура SwitchOneTwo. Но мы также с помощью

SwitchThese2(OneV, AnyV, Hold)

сможем обменять значения OneV и AnyV без всяких изменений в объявлении переменной Switchthese2. Как будто в тексте объявления процедуры формальные параметры были заменены на фактические.

Процедуры могут иметь параметры разных типов. Например:

PROCEDURE Copy (VAR F1, F2: TEXT; VAR Ch: CHAR);

Может быть использован как заголовок процедуры, чей оператор BEGIN копирует файл F1 в файл F2 с сохранением структуры строк.

PROCEDURE Copy (VAR F1, F2: TEXT; VAR Ch: CHAR);

BEGIN {Copy}

WHILE NOT EOF (F1)

DO

BEGIN

WHILE NOT EOLN (F1)

DO

BEGIN

READ(F1, Ch);

WRITE(F2, Ch)

END

READLN(F1);

WRITELN(F2)

END

END {Copy}

Эта процедура может в дальнейшем быть использована с любым существующими файлами, которые подставляются вместо F1 и F2 и любой существующей символьной переменной, которая подставляется вместо Ch. Например:

Copy(INPUT, OUTPUT, temp)

Будет отображать input в output.

**5.1.2. Синтаксические и контекстные правила для процедур с параметрами.**

Введение процедур с параметрами требует изменений в синтаксических правилах CFPascal.

**SR 27.** <процедура> ::= <заголовок процедуры>; <оператор BEGIN>

**SR 28.** <оператор процедуры> ::= <идентификатор>

| <идентификатор>(<список фактических параметров>)

**SR 30.** <список фактических параметров> ::= <список идентификаторов>

**SR 31.** <заголовок процедуры> ::= PROCEDURE <идентификатор>

| PROCEDURE <идентификатор>(<список формальных параметров>)

**SR 32.** <список формальных параметров> ::= VAR <список идентификаторов> : <тип>

| <список формальных параметров>; VAR <список идентификаторов> : <тип>

В контекстные правила также должны быть внесены изменения

**CR4.** Идентификатор процедуры, использованный в <операторе процедуры> должен появиться в <заголовке процедуры> при объявлении <процедуры>, <список фактических параметров> должен быть такой же длины что и <список формальных параметров> соответствующего <заголовка процедуры>. Типы идентификаторов в <списке формальных параметров> и <списке фактических параметров> должны соотвествовать.

**CR6.** Любой идентификатор, объявленный в<списке формальных параметров> процедуры может быть использован только в <операторе BEGIN> данной процедуры. Такие идентификаторы называются *локальными*.

**5.1.3. Разделение файлов.**

В разделе 4.1.5 была разработана программа для копирования данных из INPUT в OUTPUT с разделением на нечетные (odd), которые печатались первыми и четные (even).

Процедуры с параметрами могут упростить ее дизайн, дополнительно введем обработку маркеров конца строки вместо использования символа #.

Проект программы был такой:

PROGRAM Split(INPUT, OUTPUT);

{Копировать INPUT to OUTPUT, сначала нечетные, потом четные}

VAR

Odds, Evens: TEXT;

BEGIN {Split}

{Разделить INPUT на четные и нечетные}

{Копировать нечетные в OUTPUT}

{Копировать четные в OUTPUT}

END. {Split}

Две задачи

{Копировать нечетные в OUTPUT}

{Копировать четные в OUTPUT}

Могут быть выполнены одной процедурой CopyOut, если ей задать параметр, вместо которого будут поставлены либо Odds, либо Evens.

PROGRAM Split(INPUT, OUTPUT);

{Копировать INPUT to OUTPUT, сначала нечетные, потом четные}

VAR

Odds, Evens: TEXT;

PROCEDURE CopyOut(VAR F1: TEXT; VAR Ch1: CHAR);

{Копирует F1 в OUTPUT, игнорируя маркеры конца строк}

BEGIN {Split}

{Разделить INPUT на четные и нечетные}

CopyOut(Odds, Ch);

CopyOut(Evens, Ch);

END. {Split}

Вот определение CopyOut

PROCEDURE CopyOut(VAR F1 : TEXT; VAR Ch1 : CHAR);

{Копирует F1 в OUTPUT, игнорируя маркеры конца строк}

BEGIN {CopyOut}

RESET(F1);

WHILE NOT EOF(F1)

DO

BEGIN

WHILE NOT EOLN(F1)

DO

BEGIN

READ(F1, Ch);

WRITE(Ch)

END;

READLN(F1)

END

END {CopyOut}

Оператор BEGIN процедуры CopyOut использует только параметры F1 и Ch, объявленные в заголовке процедуры. Напротив, процедура без параметров может использовать переменные объявленные в теле программы, содержащей процедуру. Процедура с параметрами может использовать смешанный подход – одновременно пользоваться переменными окружающей программы и параметрами. Однако, процедуры, использующие только собственные параметры более легки для понимания и повторного использования. Эта идея использования одних только параметров в процедурах получила название правила «все или ничего».

**5.2. Процедуры с локальными переменными**

Дополнительно к переменным, которые являются параметрами процедуры, могут быть объявлены локальные переменные для использования в пределах процедуры. Эти переменные могут скрывать другие, объявленные вне процедуры. Существование переменных в различных местах поднимает вопрос области видимости переменных. Для иллюстрации использования локальных переменных предлагается библиотека операций по сравнению строк.

Новые идеи: объявление локальной переменной в процедуре, область видимости переменной, разрешение конфликтов имен переменных, сравнение строк, лексикографический порядок.

Параметры, как и другие переменные, используются в операторах программы. Однако, при выполнении формальные параметры заменяются соответствующими фактическими, переменными программы, которые существуют вне процедуры. Следовательно, выполнение процедуры проходит с использованием только переменных, объявленных в вызывающей программе.

Однако иногда переменные, используемые в процедуре должны используются для хранения промежуточных значений, значения которых не влияют на вызывающую программу. Например, процедура для обмена значений двух переменных с использованием третьей была написана как

PROCEDURE SwitchThese2(VAR V1, V2, temp: CHAR);

BEGIN

Temp := V1;

V1 := V2;

V2 := Temp

END

с использованием Temp как параметра. При использовании процедуры для обмена двух переменных, скажем, One и Two, вызывающая программа должна была предоставить третью, временную переменную, например

SwitchThese2(One, Two, Ch)

обменяет значения One и Two и также поместит в Ch старое значение One.

Возможность объявления локальных переменных в CF Pascal позволяет таким переменным как Temp быть скрытыми внутри процедуры. Например:

PROCEDURE Exchange(VAR Ch1, Ch2: CHAR);

{Обменивает Ch1 и Ch2}

VAR

Temp: CHAR;

BEGIN

Temp := Ch1;

Ch1 := Ch2;

Ch2 := Temp

END

Как и формальные параметры Ch1 и Ch2, локальная переменная Temp не известна и не может быть использована извне процедуры Exchange. Оператор процедуры

Exchange(One, Two)

обменяет значения One и Two. Exchange может быть проще описан для повторного использования, поскольку он использует и изменяет только две переменные вместо трех.

Существование локальных переменных скоротечно. Они входят в состояние выполнения в начале выполнения оператора процедуры, аналогично тому как переменные программы попадают в состояние выполнения в начале выполнения программы и они пропадают из состояния выполнения когда выполнение процедуры завершается.

Таблицы выполнения показывают поведение переменных в процедурах также, как они делали это для программ. Например, программа

PROGRAM Copy2(INPUT, OUTPUT);

{Копирует первые два символа из INPUT в OUTPUT}

PROCEDURE CopyChar(VAR F1, F2: TEXT);

{Копирует в F2 следующий символ их F1}

VAR

Ch: CHAR;

BEGIN {CopyChar}

IF NOT EOF(F1)

THEN

BEGIN

READ(F1, Ch);

WRITE(F2, Ch)

END

END; {CopyChar}

BEGIN {Copy2}

CopyChar(INPUT, OUTPUT);

CopyChar(INPUT, OUTPUT);

WRITELN

END. {Copy2}

имеет следующую таблицу выполнения.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | INPUT | OUTPUT | Ch | EOF |
| PROGRAM Copy2(INPUT, OUTPUT);  BEGIN {Copy2}  CopyChar (INPUT, OUTPUT)  VAR Ch: CHAR;  BEGIN {CopyChar}  IF NOT EOF(INPUT)  READ(INPUT, Ch);  WRITE(OUTPUT, Ch)  END; { CopyChar}  CopyChar (INPUT, OUTPUT)ж  VAR Ch: CHAR;  BEGIN {CopyChar}  IF NOT EOF(INPUT)  READ(INPUT, Ch);  WRITE(OUTPUT, Ch)  END; { CopyChar}  WRITELN  END. {Copy2} | ABC  ABC/  ABC/  ABC/  ABC | \_  A\_  AB\_  AB/\_  AB | ?  A  §  ?  B  § | FALSE  FALSE  FALSE |

Локальная переменная Ch перестает существовать в состоянии выполнения в местах отмеченных знаком параграфа. Хотя оба процедурных оператора вызывают CopyChar один за другим, значение Ch теряется, действительно переменная перестает существовать между вызовами CopyChar.

Внутри процедуры могут быть не только объявления локальных переменных, но и объявления локальных процедур. Локальные процедуры могут вызываться только в пределах процедуры, где они объявлены, и полностью невидимы за ее пределами. Значение, которое приходит в состояние выполнения для локальной процедуры – это ее текст. В целом дела с объявлениями внутри процедур обстоят примерно также, как и с объявлениями в программе за исключением того, что идентификаторы могут быть использованы только в пределах процедуры и входят и выходят из состояния выполнения при каждом вызове процедурного оператора.

**5.2.1. Переменные с одинаковыми именами**

В больших программах процедуры могут разрабатываться разными людьми. Если бы каждая локальная переменная должна бала иметь уникальное имя, параллельная разработка была бы усложнена. Кроме того, требование уникального имени создает сложности при повторном использовании процедуры, поскольку ее имена могут конфликтовать с именами приложения. К счастью, Паскаль позволяет использовать одинаковые имена в разных процедурах.

Это согласуется с тем, что локальные идентификаторы не существуют за пределами процедуры, где они объявлены. Сюрпризом может казаться то, что Паскаль позволяет локальному идентификатору перекрыть другой, который существует в окружающем контексте. Переменные, которые могут быть использованы в процедуре, но не объявлены в ней, называют глобальными.

Рассмотрим программу, где существует глобальная переменная Ch и локальная Ch, значения которых существуют одновременно, хотя в любой момент времени доступна только одна.

PROGRAM CopyR(INPUT, OUTPUT);

{Копирует в OUTPUT первые два символа из INPUT в обратном порядке}

PROCEDURE CopyChar(VAR F1, F2: TEXT);

{Копирует в F2 следующий символ их F1}

VAR

Ch: CHAR;

BEGIN {CopyChar}

IF NOT EOF(F1)

THEN

BEGIN

READ(F1, Ch);

WRITE(F2, Ch)

END

END; { CopyChar}

BEGIN {CopyR}

READ(Ch);

CopyChar(INPUT, OUTPUT);

WRITELN(Ch)

END. {CopyR}

Оператор READ CopyR использует глобальную Ch, а во время выполнения процедурного оператора

CopyChar(INPUT, OUTPUT)

Локальная Ch (объявленная в CopyChar) перекрывает глобальную и используется операторами READ, WRITE внутри процедуры.

READ(F1, Ch);

WRITE(F2, Ch)

Во время когда локальная Ch перекрывает глобальную, значение глобальной не теряется, к ней невозможен доступ, но ее значение сохраняется Паскаль-машиной. Далее, когда выполнение CopyChar завершается локальная Ch пропадает и глобальная снова становится доступной. Таким образом, финальный оператор

WRITE(OUTPUT, Ch)

Ссылается на глобальную переменную.

Таблица выполнения ниже показывает значения обоих переменных. Символ параграфа используется, чтобы явно обозначить, что переменная не существует. Участок между горизонтальными линиями обозначает недоступность глобальной переменной Ch.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | INPUT | OUTPUT | global  Ch | Local  Ch |
| PROGRAM CopyR(INPUT, OUTPUT);  VAR Ch: CHAR;  BEGIN {CopyR}  READ(Ch);  CopyChar(INPUT, OUTPUT)  VAR Ch: CHAR;  BEGIN {CopyChar}  READ(INPUT, Ch);  WRITE(OUTPUT, Ch)  END  WRITELN(Ch)  END. {CopyR} | ABC  ABC/  ABC/  ABC/  ABC | \_  B\_  BA/\_  BA | §  ?  A  \_\_\_\_\_  \_\_\_\_\_ | §  ?  B  § |

Таким образом, при выполнении A считывается первой, а записывается последней, после того как B считывается и записывается CopyChar.

**5.3. Область видимости локальных переменных**

Объявление внутри процедуры других процедур, которые в свою очередь могут содержат объявления, поднимает мудреные вопросы о том, какой из идентификаторов может быть использован в каждой точке программы.

*Область видимости (scope)* идентификатора – множество операторов, в которых этот идентификатор может быть использован.

В Паскале идентификатор может быть использован в пределах <блока>, в котором он был объявлен. Максимальная область видимости идентификатора – это оператор BEGIN, размещенный внутри PROCEDURE или PROGRAM, содержащих объявление идентификатора. Например, для переменных объявленных в разделе объявлений программы это ее главный оператор BEGIN и операторы BEGIN всех процедур программы.

Однако область видимости может быть меньше, если существует вложенный блок, внутри которого объявлен идентификатор с тем же именем.

Многие сложности с областью видимости идентификаторов могут стать понятными, если мы рассмотрим пример.

На рисунке ниже область видимости каждого идентификатора показана вертикальной линией, которая выделяет те операторы, где эта переменная доступна. Для того, чтобы различать идентификаторы с одним именем, объявленные в разных местах, мы используем имя процедуры или программы.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| идентификатор  объявлен в | A  S | A  P | B  S | B  Q | B  R | C  P | D  Q | D  R | P  S | Q  P | R  S |
| PROGRAM S(INPUT, OUTPUT);  VAR A, B: CHAR;  PROCEDURE P(VAR C: CHAR);  VAR A: CHAR;  PROCEDURE Q(VAR D: CHAR);  VAR B: CHAR;  BEGIN {Q}  ...  END; {Q}  BEGIN {P}  ...  END; {P}  PROCEDURE R(VAR B: CHAR);  VAR D: CHAR;  BEGIN {R}  ...  END; {R}  BEGIN {S}  ...  END. {S} | |  |  |  |  |  |  |  |  |  |  | | |  |  |  |  |  |  |  | | |  |  |  |  |  |  |  |  |  |  | | |  |  |  | | |  |  |  |  | | |  |  |  |  |  |  |  |  |  | | |  |  |  |  | | |  |  |  | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | |  |  |  |  | | |  |  |  |  |  |  |  | |

Отметим, что P может быть использован в P, Q, R и S, потому что он объявлен в S, все эти процедуры объявлены в S и ни одна из них не содержит нового объявления P. R не имеет такой же области видимости, как P только потому, что объявлен позднее. Идентификаторы в Паскале могут быть использованы только после того, как были объявлены. A и B, объявленные в S, скрываются в соответственно в P и Q. Идентификатор B, объявленный в S, виден за пределами объявлений в Q и R.

Поскольку правила области видимости Паскаля зависят от структуры текста программы, а не от ее поведения в процессе выполнения, они называются *статическими* правилами области видимости.

Область видимости идентификатора раз и навсегда определяется синтаксисом программы, в которой он объявлен, и все вопросы о его области видимости могут быть разрешены исходя из текста программы.

**5.3.1. Синтаксические и контекстные правила.**

Поскольку формат синтаксиса процедуры совершенно совпадает с форматом синтаксиса программы, синтаксическая единица <блок> описывает структуру объявлений и операторов, поэтому для того, чтобы описать вновь введенные понятия, необходимо всего лишь поменять правило 27.

**SR27.** <процедура> ::= <заголовок процедуры>;<блок>

Обсуждение области видимости дает возможность сформулировать следующее простое контекстное правило.

**CR7.** Идентификатор не может быть использован за пределами его области видимости.

**15.4. Параметры–переменные и параметры–значения.**

Параметры, передаваемые по значению, являются механизмом изоляции между точкой вызова и операторами вызываемой подпрограммы. Никакие действия в подпрограмме не могут передать информацию обратно вызывающему коду через параметр, переданный по значению. Таким образом, эти параметры являются безопасным способом для предоставления неизменяемых входных данных.

Существует две разновидности формальных параметров в D Pascal: параметры-переменные и параметры-значения. В объявлении процедуры параметры-переменные выглядят как объявления переменных. Они начинаются ключевым словом VAR, за которым следует список идентификаторов, разделенных запятыми, которые являются именами формальных параметров, двоеточие и идентификатор, определяющий тип параметра. Параметры-значения определяются как параметры-переменные, за исключением того, что ключевое слово VAR пропускается. В следующем примере VarParm – формальный параметр-переменная типа T, а ValParm – формальный параметр-значение типа T.

PROCEDURE P (VAR VarParm: T; ValPrm: T);

Синтаксические правила, описывающие параметры:

<формальный параметр> ::= <параметр-переменная> | <параметр-значение>

<параметр-переменная> ::= VAR <список идентификаторов> : <идентификатор типа>

<параметр-значение> ::= <список идентификаторов> : <идентификатор типа>

Когда фактический параметр связывается с формальным параметром-переменной, устанавливается ссылка между именем фактического параметра (который должен быть переменной, а не константой или выражением) и именем формального параметра. Ссылка используется потому, что когда процедурный оператор заменяется телом объявленной процедуры, формальные параметры-переменные заменяются соответствующими фактическими параметрами. Таким образом, каждый раз, когда формальному параметру-переменной присваивается значение, изменяется значение связанного фактического параметра. Тип фактического параметра должен быть одинаковым с типом формального параметра.

Часто параметры используются только для передачи значений. Значение в вызывающий код через эти параметры не возвращается. Такие исключительно входные параметры, должны быть объявлены как формальные параметры-значения для различения их с формальными параметрами-переменными, которые используются как выходные или входные-выходные параметры. Когда фактический параметр связан с формальным параметром-значением, значение фактического параметра присваивается формальному параметру, но ссылка не устанавливается. Таким образом, любые присваивания значений формальному параметру в блоке процедуры изменяют его значение, но не значение фактического параметра.

Поскольку в параметре-переменной сохраняется значение фактического параметра, а не устанавливается ссылка, между фактическим и формальным параметрами, любое выражение (включая константы, переменные и более сложные выражения) может быть использовано как фактический параметр, если тип значения фактического параметра совместимо для присваивания с типом формального.

Разница между параметрами-переменными и значениями видна в программе VarVsValue.

PROGRAM VarVsValue(OUTPUT);

VAR

A, B: INTEGER;

PROCEDURE Bump (VAR VarF: INTEGER; ValF: INTEGER);

BEGIN {Bump}

VarF := VarF + 1;

ValF := ValF + 1;

WRITELN(VarF, ValF)

END; {Bump}

BEGIN {VarVsValue}

A := 0;

B := 0;

Bump(A, B);

WRITELN(A, B)

END. {VarVsValue}

Выполнение:

OUTPUT: 1 1

1. 0

Bump добавляет 1 к каждому формальному параметру, так что оба ValF и VarF имеют значение 1 в конце процедуры. Однако, изменения сделанные с формальными параметрами в Bumpпоявляются только в фактическом параметре, который связан с формальным параметром-переменной и не происходят в фактическом параметре, который был связан со значением формального параметра-значения. А имеет значение 1 в конце блока программы, а B сохранило свое значение, которое она имела до выполнения процедуры.

**15.5. Функции**

Как и процедуры, функции инкапсулируют вычисления, так что они могут быть легко использованы и могли быть модифицированы передачей различных аргументов при использовании. Однако, функции возвращают значения и вызываются по имени внутри выражений.

Объявления процедур инкапсулируют последовательности операций, часто для вычисления и возврата значения через параметр-переменную. Объявления и вызовы функций в Паскале обеспечивают другой способ возвращения вычисленного значения. Объявления функций выглядят в основном как объявления процедур, но заголовок заканчивается двоеточием и идентификатором типа, обозначая тип результата. Процедура передает результаты, присваивая значения ее параметрам. Внутри блока объявления функции, результат обозначается присваиванием идентификатору, который является именем функции. Если внутри блока не сделано присваивания этому идентификатору, результат вызова функции не определен.

Механизм возвращения результата показан ниже:

FUNCTION Max2(Ch1, Ch2: CHAR): CHAR;

{Ch1 > Ch2 --> Max2 := Ch1, Ch1 <= Ch2 --> Max2 := Ch2}

BEGIN {Max2}

IF Ch1 > Ch2

THEN

Max2 := Ch1 {возвращаем Ch1 как значение Max2}

ELSE

Max2 := Ch2 {возвращаем Ch2 как значение Max2}

END; {Max2}

Синтаксические правила для объявления функции представлены ниже:

<раздел объявления функций/процедур> ::= <объявления функций/процедур> |

<объявления функций/процедур> ::= <объявление функции/процедуры>

| <объявления функций/процедур> <объявление функции/процедуры>

<объявление функции/процедуры> ::= <заголовок>; <блок>;

<заголовок>::= PROCEDURE <идентификатор> <список формальных параметров>

| FUNCTION <идентификатор> <список формальных параметров> : <тип результата>

Вызовы функций выглядят как вызовы процедур. За именем функции следует список фактических параметров, который представляет собой выражения, разделенные запятыми. Однако есть важное различие между вызовами функций и процедур. Поскольку функции возвращают значения, вызовы функций *являются выражениями*, которые могут появляться внутри других выражений или просто появляться там, где в программе может появляться выражение. Процедурные вызовы не возвращают значений и *являются операторами*. Для нахождения максимума из трех значений Max3 может быть объявлена, используя два вызова Max2 в правой части операторов присваивания.

FUNCTION Max3(Ch1, Ch2, Ch3: CHAR): CHAR;

{Max3 присваивается наибольшее значение из Ch1, Ch2, Ch3}

VAR

T: CHAR;

BEGIN {Max3}

T := Max2(Ch1, Ch2);

Max3 := Max2(T, Ch3);

END; {Max3}

Когда Max2 вызывается первый раз , она возвращает наибольшее из Ch1, Ch2, которое присваивается T. Далее T используется как фактический параметр во втором вызове Max2. Поскольку вызовы функций могут появляться везде, где появляются выражения, а параметры Max2 являются параметрами-значениями, Max3 может быть переписана так, что первый вызов Max2 будет параметром в ее втором вызове.

FUNCTION Max3(Ch1, Ch2, Ch3: CHAR): CHAR;

{Max3 присваивается наибольшее значение из Ch1, Ch2, Ch3}

BEGIN {Max3}

Max3 := Max2(Ch1, Max2(Ch2, Ch3));

END; {Max3}

Выражение в операторе присваивания будет вычислено сначала вызовом Max2 для получения максимума P2, P3 и далее вызовом Max2 с P1 и результатом предыдущего вызова.

Заметим, что <список фактических параметров> может быть пустым – функция может не иметь параметров.

**5.6. Программные модули**

Программный модуль – это механизм *инкапсуляции* идей программы именно таким образом. Несколько программных модулей, один для управления счетчиком, другие – для управления очередью будут разработаны в этой главе. Модуль отличается от процедуры тем, что он может сохранять информацию между успешными операциями, которые он выполняет. В этом смысле он ведет себя как формальный объект, называемый конечным автоматом.

Стратегия пошаговой детализации предназначения для разработки любой программы как иерархии шагов проектирования, которые определяют задачи, затем расширяют задачи в разделы проекта в соответствии с требованиями задач. Каждая детализация задачи – часть программы, для которой задается спецификация в терминах функции частного значения, которой она должна удовлетворять. Эта функция описывает преобразования данных, которые должна выполнять часть программы.

Основная идея этой главы – расширить концепцию части программы до новой, более мощной проектной конструкции, программного модуля. Программный модуль, или более просто, *модуль*, определяется как группа объявлений данных и процедур, которые задаются в одном блоке программы со следующими ограничениями:

* Данные, описанные в модуле, будут доступны только операторам процедур модуля и не доступны операторам вне модуля.
* Операторы внутри процедур модуля работают только с параметрами процедур и данным модуля и не работают с данными вне модуля.

Программный модуль обеспечивает механизм для хранения, обработки и выборки данных для программы. Иллюстрацией такого подхода является объявление файлов типа TEXT и операторы READ и WRITE, которые управляют такими файлами. Эти операторы предоставляют единственный способ для сохранения и выборки данных. Взаимодействие между модулем и остальной частью программы осуществляется через символьные параметры в операторах READ и WRITE.

Определение модуля не является явной частью синтаксиса стандартного языка Pascal, так же как и раздел проекта в методе пошаговой детализации. Скорее, модуль предоставляет другой способ проектирования и организации программ в режиме ориентированном на пошаговую детализацию.

Ограничения модуля позволяют защищать данные модуля от других частей программы, то есть, будучи корректно обработанными процедурами модуля, его данные не могут быть по невнимательности изменены или разрушены. Такой метод защищенного программирования называется скрытием информации, и горят, что данные *инкапсулированы* в модуле.

Такой метод принуждает выполнять все взаимодействие с модулем посредством его процедур и их параметров. Следовательно, не существует возможности для данных в модуле быть измененными иначе, чем с помощью процедурного оператора, который вызывает одну из процедур модуля. Также для такой процедуры невозможно изменить данные за пределами модуля иначе, чем через передаваемые ей параметры.

Модуль с исключительно сохранением и выборкой данных – важный особый случай. Данные попадают в модуль через параметры процедур и в дальнейшем могут быть возвращены без изменений через другие параметры. Файлы CF Pascal иллюстрируют это свойство: данные помещенные в файл, всегда снова появляются в той же самой форме при выборке.

Другой важный специальный случай модуля, когда он не имеет объявлений данных, в нем выполняются чистые вычисления. Процедура, не использующая глобальных данных, удовлетворяет всем ограничениям. Группа процедур, которые, например, образуют библиотеку, например,. для обработки целых чисел, предложенную в главе 9, образуют чисто-вычислительный модуль.

Модули предоставляют новые возможности для проектирования программ. Они позволяют разработку операций над данными на любом уровне необходимом для решения задачи. возможности для проектирования программ. Они позволяют разработку операций над данными на любом уровне необходимом для решения задачи. Например, если задача связана со словами текста, могут быть разработаны модули для обработки слов как отдельных блоков. Операции с данными для кроссворда, головоломки, шахмат или игры в бейсбол могут быть рассмотрены в терминах модулей с данными и процедурами для работы с особенными свойствами соответствующих игр.

**5.6.1. Реализация модулей в современных компиляторах Pascal**

В современных реализациях языка Pascal, как правило, присутствует поддержка модулей и дисциплины работы с модулями на уровне компилятора. Модуль размещается в отдельном файле, структура которого может быть описана с помощью следующего правила BNF.

<модуль> ::= UNIT <имя модуля>;

INTERFACE

<объявления модуля>

IMPLEMENTATION

<тексты процедур модуля>

BEGIN

<список операторов>

END.

Как и Паскаль-программа, модуль имеет заголовок, UNIT <имя модуля>; раздел объявлений (интерфейс модуля), начинающийся с ключевого слова INTERFACE и блок реализации, начинающийся с ключевого слова IMPLEMENTATION (реализация). Также блок содержит оператор BEGIN, выполняющий инициализацию данных модуля. Как и программа, модуль точкой.

**5.6.2. Модуль счетчика**

Как иллюстрация простого модуля который обеспечивает как сохранение/выборку так и вычисления рассмотрим задачу, которая включает счет до 999, например, подсчет пробелов в строке. В некоторых случаях требуется увеличивать счетчик по единице, или определить текущее значение, сохраняемое счетчиком, или обнулить счетчик. Для этого может быть разработан модуль, в котором счет организован в трех символьных переменных как в программе CountChars в разделе 1.6.3.

VAR

Ones, Tens, Hundreds :CHAR;

и процедуры:

Start {обнуляет счетчик}

Bump {Увеличивает счетчик на единицу}

Value(VAR V100, V10, V1: CHAR) {Возвращает значение счетчика}

Модуль разрабатывается таким образом, что после выполнения Start модуль содержит натуральное число из трех цифр. Если счетчик содержит 999, Bump не меняет значение счетчика. В Pascal-программе использующей этот модуль, переменные модуля должны быть помещены в раздел VAR.

UNIT Count3;

{Модуль счетчика}

INTERFACE

PROCEDURE Start;

PROCEDURE Bump;

PROCEDURE Value (VAR V100, V10, V1: CHAR);

IMPLEMENTATION

VAR

Ones, Tens, Hundreds: CHAR;

{Тексты процедур Start, Bump и Value модуля}

...

BEGIN

{Инициализация данных модуля}

END.

Объявление трех переменных и трех процедур задет модуль счетчика, который может быть вставлен в любую программу, в которой есть необходимость в таком счетчике. Для соблюдения ограничений модуля достаточно, чтобы оставшаяся часть программы не использовала глобальные переменные Ones, Tens, Hundreds. Тогда счетчик полностью скрыт за тремя процедурами Start, Bump и Value.

UNIT Count3;

{Модуль счетчика}

INTERFACE

PROCEDURE Start;

PROCEDURE Bump;

PROCEDURE Value (VAR V100, V10, V1: CHAR);

IMPLEMENTATION

VAR

Ones, Tens, Hundreds: CHAR;

PROCEDURE Start;

{Сбрасывает счетчик в ноль}

BEGIN{Start}

Ones := '0';

Tens := '0';

Hundreds := '0'

END;{Start}

PROCEDURE Bump;

{Увеличивает 3-значный счетчик определенный Ones, Tens, Hundreds

на единицу ,если он находится в диапaзоне от 0 до 999 }

PROCEDURE NextDigit(VAR Digit: CHAR);

BEGIN {NextDigit}

IF Digit = '0' THEN Digit :='1' ELSE

IF Digit = '1' THEN Digit :='2' ELSE

IF Digit = '2' THEN Digit :='3' ELSE

IF Digit = '3' THEN Digit :='4' ELSE

IF Digit = '4' THEN Digit :='5' ELSE

IF Digit = '5' THEN Digit :='6' ELSE

IF Digit = '6' THEN Digit :='7' ELSE

IF Digit = '7' THEN Digit :='8' ELSE

IF Digit = '8' THEN Digit :='9' ELSE

IF Digit = '9' THEN Digit :='0'

END;{NextDigit}

BEGIN {Bump}

NextDigit( Ones );

IF Ones = '0'

THEN

BEGIN

NextDigit(Tens);

IF Tens= '0'

THEN

BEGIN

NextDigit(Hundreds);

IF Hundreds= '0'

THEN

BEGIN

Ones := '9';

Tens := '9';

Hundreds := '9'

END

END

END

END; {Bump}

PROCEDURE Value (VAR V100, V10, V1: CHAR);

{Возвращает содержимое счетчика}

BEGIN {Value}

V100 := Hundreds;

V10 := Tens;

V1 := Ones

END; {Value}

BEGIN

{Инициализация данных модуля}

Start;

END.

Этот модуль счетчика может быть использован для решения задач подсчета количества пробелов в строке.

PROGRAM CountingBlanksInText(INPUT, OUTPUT);

VAR

Ch, X100, X10, X1: CHAR;

{Включить модуль счетчика}

BEGIN {CountingBlanksInText}

Start; {обнулить счетчик}

WHILE NOT EOF

DO

BEGIN

WHILE NOT EOLN

DO

BEGIN

READ(Ch);

IF Ch = ‘ ‘

THEN

BEGIN

Bump; {Увеличиваем счетчик на едеинцу}

Ch := ‘#’;

END;

WRITE(Ch);

END;

READLN;

WRITELN

END;

WRITELN;

Value(X100, X10, X1); {получаем значение счетчика}

IF (X100 = ‘9’) AND (X10 = ‘9’) AND (X1 = ‘9’)

THEN

WRITELN(‘Количество пробелов как минимум 999’)

ELSE

WRITELN(‘Количество пробелов ’, X100, X10, X1)

END. {CountingBlanksInText}

INPUT:

Now is

the time for

all good men.

OUTPUT:

##Now#is#

the##time#for

all#good##men.

Количество пробелов 010

Основная стоимость использования модуля в собранной программе – это дублирование переменных Hundreds, Tens и Ones в модуле переменными X100, X10, X1 в блоке вне модуля. Если бы вызов Value был удален, могли бы быть исключены и переменные X100, X10, X1 и значения Hundreds, Tens и Ones распечатаны напрямую. Но это нарушает ограничение, что данные описанные в модуле доступны только операторам внутри модуля. Несколько лишних переменных – небольшая цена за возможность повторного использования модуля, который тщательно проверен и протестирован раз и навсегда и использован многократно в других программах.

Для использования модулей в программе необходимо указать эти модули явно с помощью ключевого слова USES <список модулей>, которое помещается перед разделом объявлений.

Например:

PROGRAM CountingBlanksInText(INPUT, OUTPUT);

USES

Count3;

VAR

Ch, X100, X10, X1: CHAR;

BEGIN {CountingBlanksInText}

...

END. {CountingBlanksInText}