**6. Программирование с рекурсией.**

Рекурсия – подход к решению задач, когда задача сводится к аналогичной, но более простой. Полное решение задачи принимает форму рекурсивной процедуры, которая выполняет необходимые упрощения задачи, далее вызывает себя для решения более простой задачи.

**6.1. Рекурсивные процедуры в CF Pascal.**

Процедуры в CF Pascal могут включать процедурные операторы, которые вызывают их самих. Для некоторых задач рекурсия – естественный и очевидный подход, более предпочтительный, чем итеративные решения.

Новые идеи: рекурсия и ее реализация в CF Pascal.

**6.1.1. Решение задач с рекурсией.**

Метод семейства задач, рассмотренный в главе 4 подводит нас к рекурсивным решениям. Как показано в семействах программ IFSort и MinSort, в этом подходе для сортировки более длинных строк использовались ранее разработанные подходы для сортировки более коротких строк. Например, в семействе MinSort, MinSort4, которая сортирует 4-строки может быть описана в терминах MinSort3, MinSort5 – в теминах MinSort4 и т.д. Решение для сортировки m-строк – использовать сортировку (m-1)-строк.

Именно эта концепция используется в SelectSort, для m-строки, которую требуется отсортировать, находится минимальный символ и удаляется из строки. Остается (m-1)-строка, которую также обрабатывает SelectSort, получая для дальнейшей сортировки (m-2)-строку, …, 2-строку, 1-строку. Финальная задача, сортировка 1-строки, имеет тривиальное решение.

Стратегия решения задачи ищет способ циклически приводить задачу к более простой той же формы, пока не будет достигнута последняя задача, которая может быть решена напрямую. Предположим, что может быть написана процедура решения для каждого члена в таком семействе задач с данными для решения в параметре процедуры. Далее, процедура должна быть способна вызывать себя с параметром, содержащим данные меньшего члена семейства. Например, придумаем процедуру для сортировки строки из файла F1 в OUTPUT.

PROCEDURE RSort (VAR F1: TEXT);

VAR

Min: CHAR;

F2: TEXT;

BEGIN {RSort}

RESET(F1);

IF NOT(EOLN(F1))

THEN

BEGIN

REWRITE(F2);

{Положить минимальный из F1 в Min

и копировать другие символы в F2}

WRITE(Min);

{Копировать F2 обратно в F1}

{\*\*\*}

END

END; {RSort}

В строке отмеченной {\*\*\*} F1 содержит строку на один символ меньше, чем при вызове процедуры, первый символ отсортированной последовательности уже записан. Задача была сведена к сортировке более короткой строки в точно таком же формате, для которого была разработана RSort. Таким образом, {\*\*\*} может быть замещен на процедурное выражение

RSort(F1)

В случае пустого файла, нам нечего писать в OUTPUT, RSort именно это и делает, проверяя на EOLN.

Предположим, что RSort вызывается в процедурном идентификаторе.

PROGRAM DoRSort (INPUT, OUTPUT);

VAR

Raw: TEXT;

{Включить PROCEDURE RSort (VAR F1: TEXT);}

BEGIN {DoRSort}

REWRITE(Raw);

{Копировать строку из INPUT в Raw}

WRITELN(‘Сортированая строка:’);

RSort(Raw);

WRITELN;

END. {DoRSort}

Может показаться, что здесь есть что-то магическое. RSort обрабатывает только один символ, не выполняя сортировки. Но она будет перезапускаться снова и снова, каждый раз обрабатывая один символ, снова до тех пор пока файл передаваемый как параметр не станет пустым. Каждый символ будет выведен в OUTPUT в сортированном порядке.

**6.1.2. Выполнение рекурсивной процедуры**

Когда объявление процедуры включает процедурный оператор с собственным именем этой процедуры – рекурсивный вызов - ничего особенного не происходит. Вновь объявленные переменные и параметры скрывают одноименные объявленные ранее. Ничего особенного также не происходит при завершении выполнения процедуры. Экземпляры переменных и параметров, созданные при последнем запуске процедуры исчезают, становятся доступны одноименные копии из предыдущего запуска и выполняются операторы, следующие за процедурным оператором.

Поскольку каждый вызов рекурсивной процедуры использует те же имена локальных переменных (и обычно те же имена параметров), существует возможность коллизии значений связанных с этими именами. Отслеживание значений требует аккуратности, но не отличается сильно от рассмотренного ранее случая без рекурсии. Рассмотрим рекурсивную процедуру в следующей программе.

PROGRAM RunReverse(INPUT, OUTPUT);

PROCEDURE Reverse (VAR File1: TEXT);

VAR

Ch: CHAR;

BEGIN {Reverse}

IF NOT EOLN(File1)

THEN

BEGIN

READ(File1, Ch);

Reverse(File1);

WRITE(Ch)

END

END; {Reverse}

BEGIN {RunReverse}

Reverse(INPUT);

WRITELN;

END. {RunReverse}

Выполнение:

INPUT: AB

OUTPUT: BA

Для INPUT AB процедура была вызвана три раза. Первый раз в процедурном операторе программы, два последних раза – в процедурном операторе внутри процедуры. Каждый такой процедурный оператор выполняет блок процедуры, начинающийся с объявления VAR. Каждое такое объявление создает новую копию переменной Ch, и Паскаль-машина сохраняет и скрывает ранее объявленную переменную с таким же именем.

Первые два запуска процедуры снова вызывают процедуру через процедурный оператор, но третий запуск завершается без всякого эффекта, потому что условие в операторе IF не срабатывает. При завершении выполнения каждого процедурного оператора происходит две вещи:

1) переменная Ch для текущего запуска процедуры исчезает, открывая раннее скрытую Ch

2) выполняется следующий оператор

Для двух последних запусков следующим оператором будет WRITE(Ch), после выполнения первого оператора, выполняется WRITELN и программа завершается.

Таблица выполнения, приведенная ниже показывает в деталях, что происходит:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | INPUT | OUTPUT | EOLN (File1) | Ch в запуске | | |
| 1 | 2 | 3 |
| PROGRAM RunReverse(INPUT, OUTPUT)  BEGIN {RunReverse}  Reverse(INPUT)  VAR Ch: CHAR  BEGIN {Reverse}  IF NOT EOLN(File1)  THEN  BEGIN  READ(File1, Ch);  Reverse(File1);  VAR Ch: CHAR  BEGIN {Reverse}  IF NOT EOLN(File1)  THEN  BEGIN  READ(File1, Ch);  Reverse(File1);  VAR Ch: CHAR  BEGIN {Reverse}  IF NOT EOLN(File1)  END {Reverse}  WRITE(Ch)  END  END {Reverse}  WRITE(Ch)  END  END {Reverse}  WRITELN;  END; {RunReverse} | AB  AB/  AB/  AB/  AB | \_  B\_  BA\_  BA/\_  BA | F  F  T | ?  A | ?  B | ? |

В следующем примере операторы внутри рекурсивной процедуры меняются, сначала выполняется запись, затем рекурсивный вызов. В результате выполняется копирование о обычном, не реверсивном порядке.

PROGRAM RunCopy(INPUT, OUTPUT);

PROCEDURE Copy (VAR File1: TEXT);

VAR

Ch: CHAR;

BEGIN {Copy}

IF NOT EOLN(File1)

THEN

BEGIN

READ(File1, Ch);

WRITE(Ch);

Copy (File1)

END

END; {Copy}

BEGIN {RunCopy}

Copy(INPUT);

WRITELN;

END. {RunCopy}

Выполнение:

INPUT: AB

OUTPUT: AB

В итеративной версии копирования каждый символ перемещается из входного файла через переменную программы в выходной файл. Рекурсивная версия совсем по иному. Каждый символ сохраняется в отдельной локальной переменной и все они существуют в Паскаль-машине одновременно. Эти сохраненные в локальных переменных символы могут быть потом использованы, как в следующем примере, программе, которая копирует входные символы, потом добавляет их в обратном порядке.

PROGRAM RunPalindrome(INPUT, OUTPUT);

PROCEDURE Palindrome(VAR File1: TEXT);

VAR

Ch: CHAR;

BEGIN {Palindrome}

IF NOT EOLN(File1)

THEN

BEGIN

READ(File1, Ch);

WRITE(Ch);

Palindrome (File1)

WRITE(Ch)

END

END; {Palindrome}

BEGIN {RunPalindrome}

Palindrome (INPUT);

WRITELN

END. {RunPalindrome}

Выполнение:

INPUT: AB

OUTPUT: ABBA

INPUT: 123456789

OUTPUT: 123456789987654321

**6.2. Сортировка и реверсирование с помощью рекурсии.**

Рекурсия часто предлагает программы, которые эффективны и элегантны. Идея быстрой сортировки с использованием разделения и слияния может быть применена рекурсивно.

Новые идеи: рекурсивная сортировка, рекурсивное реверсирование.

RSort сортирует используя идею повторяющегося выбора, реализованную итеративно в SelectSort. Однако MergeSort – значительно более быстрая итеративная сортировка чем SelectSort. Существует ли рекурсивный вариант MergeSort?

Чтобы применять рекурсию эффективно, сортируемая строка должна быть разделена, скажем, наполовину. Если половинки по отдельности отсортированы, они могут быть слиты, как показано в примере:

Исходная строка:

†character†

Разбиваем на две примерно равные строки:

†chara† †cter†

Сортируем полустроки

†aachr† †cert†

Сливаем полустроки в сортированную строку

†aaccehrrt†

Рекурсия происходит на втором шаге: та же процедура может быть использована для сортировки полустрок, какая применялась ранее для сортировки целой.

Остается только найти способ разделить файл на половинки. Один из известных нам способов разделения файла на две половинки – это деление на четные и нечетные. В данном случае нам не важно, какие именно символы оказались в той или иной части, важно, чтобы половинки были примерно равной длины.

Для примера, рассмотренного выше, половинки, полученные разбиением на четные и нечетные, будут:

†caatr† †hrce†

Сортируем полустроки

†aacrt† †cehr†

Сливаем полустроки в сортированную строку

†aaccehrrt†

В предыдущем и данном случае полустроки разные, но результат одинаковый.

Каждая рекурсия будет сортировать файл, который будет не больше половины длины, того, что обрабатывался на предыдущем шаге. Наконец, это деление произведет файл размером в один символ, который уже отсортирован. Таким образом, рекурсия будет завершаться проверкой длины файла.

Эти идеи приводят нас к следующему проекту быстрой рекурсивной сортировки

Раздел проекта 1 [DP 1]

PROCEDURE RecursiveSort(VAR F1: TEXT);

VAR

F2, F3: TEXT;

Ch: CHAR;

{PROCEDURE Split(VAR F1, F2, F3: TEXT)

Разбивает F1 на F2 и F3}

{PROCEDURE Merge(VAR F1, F2, F3: TEXT)

Сливает F2 и F3 в F1}

BEGIN {RecursiveSort}

RESET(F1);

IF NOT (EOLN(F1))

THEN

BEGIN

IF NOT (EOLN(F1))

THEN {Файл имеет как минимум 2 символа}

BEGIN

Split(F1, F2, F3);

RecursiveSort(F2);

RecursiveSort(F3);

Merge(F1, F2, F3);

END

END

END {RecursiveSort}

Раздел проекта 1.1 [DP 1.1]

PROCEDURE Split(VAR F1, F2, F3: TEXT);

{Разбивает F1 на F2, F3}

VAR

Ch, Switch: CHAR;

BEGIN {Split}

RESET(F1);

REWRITE(F2);

REWRITE(F3);

{Копировать F1 попеременно в F2 и F3}

WRITELN(F2);

WRITELN(F3);

END {Split}

Раздел проекта 1.1.1 [DP 1.1.1]

BEGIN {Копировать F1 попеременно в F2 и F3}

Switch := '2';

WHILE NOT (EOLN(F1))

DO

BEGIN

READ(F1, Ch);

IF (Switch = '2')

THEN

BEGIN

WRITE(F2, Ch);

Switch := '3';

END

ELSE

BEGIN

WRITE(F3, Ch);

Switch := '2';

END

END

END

Раздел проекта 1.2 [DP 1.2]

PROCEDURE Merge(VAR F1, F2, F3: TEXT);

{Сливает F2, F3 в F1 в сортированном порядке}

VAR

Ch2, Ch3: CHAR;

BEGIN {Merge}

RESET(F2);

RESET(F3);

REWRITE(F1);

READ(F2, Ch2);

READ(F3, Ch3);

WHILE (NOT(EOF(F2))) AND (NOT(EOF(F3))))

DO

BEGIN

IF Ch2 < CH3

THEN

BEGIN

WRITE(F1, Ch2);

READ(F2, Ch2);

END

ELSE

BEGIN

WRITE(F1, Ch3);

READ(F3, Ch3);

END

END

{Копировать остаток F2 в F1}

{Копировать остаток F3 в F1}

WRITELN(F1);

END {Merge}

Раздел проекта 1.2.1 [DP 1.2.1]

BEGIN {Копировать остаток F2 в F1}

WHILE NOT (EOF(F2))

DO

BEGIN

WRITE(F1, Ch2);

READ(F2, Ch2);

END

END

Раздел проекта 1.2.2 [DP 1.2.2]

BEGIN {Копировать остаток F3 в F1}

WHILE NOT (EOF(F3))

DO

BEGIN

WRITE(F1, Ch3);

READ(F3, Ch3);

END

END

Рекурсивное решение более ясное, чем итеративное, потому что здесь не требуется идентифицировать или отслеживать границы выполнения.

**6.2.2. Рекурсивное реверсирование**

Рекурсия может быть использована для реверсирования строк в подобным образом, как показано в примере

Исходная строка:

†character†

Разбиваем на первую половину и вторую:

†chara† †cter†

Реверсируем полустроки

†arahc† †retc†

Обмениваем реверсированные полустроки

†retc† †arahc†

Объединяем реверсированные полустроки в реверсированную строку

†retcarahc†

И снова неуклюжее разбиение может быть исключено. Предположим, файл был разбит на

противоположные символы как для сортировки. Если каждая часть была реверсирована, то будучи смешаны обратно, один символ из нечетной полустроки, один из четной и т.д., они образуют реверсированную строку, как на примере:

Исходная строка:

†character†

Разбиваем на полустроки из четных-нечетных

†caatr† †hrce†

Реверсируем полустроки

†rtaac† †ecrh†

Перемешиваем реверсированные полустроки в реверсированную строку

†retcarahc†

Пример показывает, что здесь необходимо обратить внимание, чтобы первый символ был из полустроки, в которую попал последний символ исходной строки.

Эти идеи позволяют нам подготовить следующий проект быстрого рекурсивного реверсирования.

Раздел проекта 2 [DP 2]

PROCEDURE RecursiveReverse(VAR F1: TEXT);

VAR

F2, F3: TEXT;

Ch, Last: CHAR;

BEGIN {RecursiveReverse}

RESET(F1);

IF NOT (EOLN(F1))

THEN

BEGIN

READ(F1, Ch);

IF NOT (EOLN(F1))

THEN {Файл имеет как минимум 2 символа}

BEGIN

{Разбить F1 на F2 и F3 и запоминаем в Last

в какой файл попал последний символ}

RecursiveReverse(F2);

RecursiveReverse(F3);

{Перемешиваем F2 и F3 в F1 начиная с файла

согласно тому, что запомнено в Last}

END

END

END {RecursiveReverse}

Оставшиеся разделы проекта аналогичны тем, что мы подготовили для RecursiveSort.