ЛАБОРАТОРНАЯ РАБОТА N1

Тема: "Стандартные типы данных и выражения"

Цель работы

Ознакомиться со стандартными типами данных и выражениями языка Паскаль.

Краткие сведения из теории

1.1. Стандартные типы данных

Тип данных определяет возможные значения констант, переменных, функций, выражений, принадлежащих к этому типу, форму представления в ЭВМ и операции, которые могут выполняться над ними.

Все типы данных делятся на простые и сложные.

Простые типы - это стандартные и переменные типы данных.

Переменные типы - это ограниченные, перечислимые, регулярные, строковые, структурные и ссылочные.

Сложные типы - это типы данных, которые задаются в разделе типов и строятся на базе простых.

В настоящей работе будут рассмотрены только стандартные типы.

Стандартными являются целый со знаком INTEGER, целые без знака BYTE и WORD, действительный REAL, логический BOOLEAN и символьный CHAR.

Данные представляются в программе в виде констант и переменных. При выполнении программы в каждый момент времени любая переменная имеет некоторое значение (константу). Это значение и переменная должны относиться к одному типу данных.

1.1.1. Данные целого типа со знаком

Константа целого типа (целая константа) - это последовательность цифр со знаком ('+', '-') или без него.

Примеры констант целого типа: -324, 16, 0, +9352, 1024.

Диапазон целых чисел зависит от конкретного типа ЭВМ. Для 16-разрядного процессора диапазон изменяется в пределах от -32768 до +32767.

Переменные, принимающие в качестве своих значений константы целого типа, относятся к целому типу (тип INTEGER).

Над данными целого типа можно выполнять следующие арифметические операции, которые дают целый результат:

+	сложение;
-	вычитание;
*	умножение;
DIV	деление с отбрасыванием дробной части;
MOD	получение целого остатка при делении целого данного на целое.

Пример: A, B, N - переменные целого типа, принимающие значения A = 25, B = 2, N = -17. В этом случае допустимы следующие операции:

операция	результат
A + 51	76
B - A	-23
B * N	-34
A DIV B	12
A MOD B	1

1.1.2. Данные целого типа без знака

Данные целого типа без знака предоставляет Турбо-Паскаль. К ним относятся числа без знака длиной в один или два байта. Им соответствуют типы BYTE и WORD. Диапазон представления данных типа BYTE изменяется от 0 до 255, а данных типа WORD - от 0 до 65535. Над данными указанных типов можно выполнять те же операции, что и над данными типа INTEGER.

Данные типа BYTE и WORD удобно использовать для индексации массивов, а также в тех случаях, когда переменные принимают только положительные значения.

1.1.3. Данные действительного типа

Константы действительного типа могут быть представлены в двух формах: с фиксированной точкой и плавающей точкой.

Константы с фиксированной точкой изображаются десятичным числом с дробной частью, которая может быть и нулевой. Дробная часть отделяется от целой с помощью точки. Примеры констант действительного типа с фиксированной точкой: 35.62; -12.005; 55.0.

Константами с плавающей точкой являются числа, представленные с десятичным порядком. Они имеют вид:

$\pm$ mE $\pm$ p, где

m - мантисса,

E - признак записи числа с десятичным порядком,

p - порядок числа.

Для записи мантиссы m могут использоваться целые числа и действительные числа с фиксированной точкой. В качестве p могут использоваться только целые числа.

Примеры записи чисел с плавающей точкой:

математическая запись		запись на языке Паскаль
$2 * 10^n$,	где n = -4	2E-4
$0{,}32 * 10^n$,	где n = 4	0.32E+4
$-12{,}75 * 10^n$,	где n = 11	-12.75E11

Примеры записи действительного числа 253 с плавающей точкой: 0.253E+3, 2.53E+2, 25.3E+1, 253.0E0, 2530E-1, 25300E-2 и т.д.

Переменные действительного типа REAL - это переменные, которые в качестве значений принимают числа с фиксированной или плавающей точкой.

Над данными действительного типа можно выполнять следующие операции, дающие действительный результат:

- \+ сложение;
- \- вычитание;
- * умножение;
- / деление.

Данные целого и действительного типа называют арифметическими данными.

Диапазон представления чисел действительного типа изменяется от 10^{-n} до 10^n (n = 38). Для различных процессоров отличие в представлении действительных чисел состоит в количестве значащих цифр мантиссы; при работе с 16-разрядным процессором - это 11 неполных шестнадцатеричных цифры ($\pm$ 7FFFFFFFFF).

1.1.4. Данные логического типа

В языке Паскаль имеются две логические константы: TRUE (истина), FALSE (ложь). Логическая переменная принимает одно из этих значений и имеет тип BOOLEAN.

Над данными логического типа выполняются следующие операции:

OR	логическое сложение (или);
AND	логическое умножение (и);
NOT	логическое отрицание (не).

Логические операции OR и AND являются бинарными и выполняются над двумя величинами, операция NOT - над одной величиной и является унарной операцией.

Логический тип определяется таким образом, что FALSE < TRUE.

Результатом любой логической операции является логическая переменная со значением FALSE или TRUE.

В таблице 1 приведены результаты операций над логическими данными.

Таблица 1

A	B	NOT A	A OR B	A AND B

TRUE	TRUE	FALSE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE	FALSE
FALSE	TRUE	TRUE	TRUE	FALSE
FALSE	FALSE	TRUE	FALSE	FALSE

Примечание: каждая логическая операция имеет свой ранг старшинства. Самой старшей является операция отрицания. Далее в порядке убывания старшинства следуют умножение и сложение.

1.1.5. Данные символьного типа

Символьная, или литерная, константа - это любой символ языка, заключенный в апострофы.
Примеры символьных констант: 'B' , ';' , '+' , '5'.
Символьная константа занимает один байт памяти.
Примечание. Чтобы представить апостроф как символьную константу его повторяют дважды: ''. Внешние апострофы (по одному слева и справа) не входят в константу, они являются признаком символьной константы.
Символьная переменная типа CHAR - это переменная, принимающая значение символьной константы. Все символы языка Паскаль упорядочены, т.е. каждый символ имеет свой порядковый номер. Это позволяет применять к символьным данным операции сравнения: <, >, =, <>, >=, <=. Например, результат операции сравнения 'A' < 'B' будет истинным, так как сравниваются их порядковые номера, а они равны 66 и 67 соответственно.

1.1.6. Данные строкового типа

В Турбо-Паскале допускается использование последовательности символов, заключенной в апострофы, длиной не более 256 символов. Например: 'Sigma', 'Alfa', 'Группа 623-1а' и т.д. Такие последовательности относятся к строковым типам данных. Более подробно строковые данные рассмотрены в лабораторной работе N 8.

1.2. Выражения

В любом языке программирования определены два типа выражений: арифметические и логические.
В состав выражения могут входить константы, переменные, стандартные функции, объединенные круглыми скобками и знаками операций. Константы, входящие в выражение, могут быть объявлены либо в разделе CONST, либо непосредственно в самом выражении. Переменные должны иметь уникальное имя, называемое идентификатором, и описываются в разделе VAR. Идентификаторы могут содержать до 16 символов, причем первый символ должен быть либо буквой, либо символом "_" (подчеркивание). В идентификаторах разрешается использовать как строчные, так и прописные буквы латинского алфавита, цифры и специальные символы.

1.2.1. Стандартные функции

При записи стандартных функций следует учитывать следующие правила:
1) имя функции записывается буквами латинского алфавита;
2) аргумент (параметр) функции записывается в круглых скобках после имени функции;
3) аргументом функции может быть константа, переменная или арифметическое выражение.
В таблице 2 приведены основные стандартные функции.

Таблица 2

Функция	Назначение	Тип аргумента	Тип функции
ABS(x)	Вычисление абсолютного значения x	REAL INTEGER	REAL INTEGER
SQR(x)	Вычисление квадрата x (x*x)	REAL INTEGER	REAL INTEGER

SIN(x)	Вычисление синуса x	REAL INTEGER	REAL REAL
COS(x)	Вычисление косинуса x	REAL INTEGER	REAL REAL
ARCTAN(x)	Вычисление арктангенса x	REAL INTEGER	REAL REAL
EXP(x)	Вычисление экспоненты	REAL INTEGER	REAL REAL
EXP10(x)	10^n, где n=x	REAL INTEGER	REAL REAL
LN(x)	ln x	REAL INTEGER	REAL REAL
LOG(x)	log x	REAL INTEGER	REAL REAL
SQRT(x)	Вычисление квадратного корня из x	REAL INTEGER	REAL REAL
TRUNC(x)	Определение целой части от x	REAL INTEGER	INTEGER INTEGER
ROUND(x)	Округление x в сторону ближайшего целого	REAL INTEGER	NTEGER INTEGER
ODD(x)	TRUE, если x - нечетное FALSE, если x - четное	INTEGER	BOOLEAN
SUCC(x)	а) x+1 б) следующий символ после x в упорядоченном множестве символов	INTEGER CHAR	INTEGER CHAR
PRED(x)	а) x-1 б) предыдущий по отношению к x символ в упорядоченном множестве символов	INTEGER CHAR	INTEGER CHAR
ORD(S	порядковый номер символа S в упорядоченном множестве символов	CHAR	INTEGER
CHR(I)	определяет символ, стоящий под номером I в упорядоченном множестве символов	INTEGER	CHAR

В данной таблице не приведены стандартные функции работы со строковыми данными. Они рассмотрены в лабораторной работе N 8.

Примеры использования стандартных функций:

1) X = 21.53 TRUNC(X) = 21 ROUND(X) = 22
2) X = -2.7 TRUNC(X) = -2 ROUND(X) = -3
3) ORD('5') = 53 ORD('5') = 58
4) CHR(66) = 'B', CHR(57) = '9'
5) PRED('B') = 'A' PRED(100)= 99
6) SUCC('C') = 'D' SUCC(12) = 13

Примечания:

1. В Паскале нет операции возведения в степень. При необходимости вычисления a^n, n = x, используют имеющиеся стандартные функции, например: a^n = Exp(n * Ln(A)).
2. Из таблицы стандартных функций следует справедливость равенств:

PRED(S) = CHAR(ORD(S) - 1) CHR(ORD(S)) = S
SUCC(S) = CHAR(ORD(S) + 1) ORD(CHR(I)) = I

3. Значения FALSE и TRUE можно рассматривать как упорядоченное множество, состоящее из двух элементов. При этом определены следующие значения стандартных функций:

ORD(FALSE) = 0 SUCC(FALSE) = TRUE

ORD(TRUE) = 1 PRED(TRUE) = FALSE.

1.2.2. Арифметические выражения

Арифметическое выражение представляет собой совокупность одного или нескольких арифметических констант, переменных, функций, соединенных знаками арифметических операций и круглыми скобками.

Следовательно, константа, переменная, функция являются частными случаями арифметического выражения.

При записи выражений необходимо выполнять следующие правила:

1. Все составные части выражения записываются в одну строку. Поднимать и опускать символы над строкой не разрешается.

2. Использовать в выражениях можно скобки только одного типа - круглые. Применение в выражениях фигурных и квадратных скобок запрещается, так как они имеют особое назначение. Число открывающихся скобок всегда должно равняться числу закрывающихся скобок.

3. Нельзя записывать последовательно два знака арифметических операций, их необходимо разделить круглой скобкой.

4. Вычисление выражений производится слева направо в соответствии со старшинством операций.

Установлен следующий в порядке убывания приоритет арифметических операций:

а) умножение, деление;

б) DIV, MOD;

в) сложение, вычитание.

Необходимый порядок вычислений в выражениях задается с помощью скобок.

Выражения в скобках вычисляются первыми. Если выражений в скобках несколько, и они вложены друг в друга, то вычисление начинается в самых внутренних скобках и далее последовательно переходит во внешние скобки.

Если аргумент функции задан в виде выражения, то сначала определяется значение этого выражения, а затем значение функции.

Если операции, следующие одна за другой, имеют одинаковое старшинство, то они выполняются последовательно в порядке их записи.

1.2.3. Логические выражения

Логические выражения строятся из логических данных, логических операций и операций отношения.

В операциях отношения могут участвовать арифметические и логические выражения, а также символьные данные.

Результатом логического выражения является значение TRUE или FALSE .

При вычислении логических выражений принят следующий приоритет операций (по старшинству):

1) арифметические операции;

2) операции отношений;

3) логические операции.

При наличии скобок сначала выполняются действия в скобках (в первую очередь самые внутренние), а затем вне скобок.

В круглые скобки обязательно заключаются части выражения, стоящие слева и справа от логических операций AND и OR.

Пример. Определить результат логического выражения

(A > 3) AND (C = 5) OR NOT (A + C < X) при A = 1, C = 3, X = 0.

С заданными значениями переменных, входящих в логическое выражение, оно принимает вид:

(1 > 3) AND (3 = 5) OR NOT (1 + 3 < 0)

Результат вычислений в скобках:

(FALSE) AND (FALSE) OR NOT (FALSE).

Логические операции выполняются в следующей последовательности NOT - AND - OR:

1) NOT (FALSE) = TRUE;
2) (FALSE) AND (FALSE) = FALSE;
3) (FALSE) OR (TRUE) = TRUE;

Результат рассмотренного логического выражения равен TRUE.

Контрольные вопросы

1. Перечислите стандартные типы данных в Паскале.
2. Укажите отличие данных действительного и целого типов.
3. Какие переменные называют логическими и какие значения они могут принимать?
4. Какие логические операции вы знаете?
5. Что представляет собой условие?
6. Что понимается под символьными данными?
7. Какие данные называют арифметическими?
8. Перечислите правила записи стандартных функций.
9. Перечислите стандартные функции, предназначенные для работы с арифметическими данными?
10. Перечислите стандартные функции, предназначенные для работы с символьными данными?
11. Что представляет собой арифметическое выражение?
12. Перечислите правила записи арифметических выражений.
13. Что представляет собой логическое выражение?
14. В чем отличие арифметического выражения от логического?
15. Каков порядок вычисления значения логического выражения?

Содержание отчета

Контрольная работа по рассмотренной теме.

Варианты индивидуальных заданий

1. Какие из приведенных ниже записей являются неправильными и почему?

а) 7.	б) -6.1	в) 0.0	г) 9
д) .0E-2	е) 0.1E-5	ж) -5.3E4	з) +2.3E+3.5
и) -71	к) 0.31	л) .456	м) 2,1

2. Чему равно значение следующих функций?

а) TRUNC(5.61)	г) TRUNC(-5.61)
б) ROUND(17.16)	д) ROUND(17.96)
в) ROUND(-17.16)	е) ROUND(-17.96)

3. Какие из приведенных ниже записей функций являются неправильными и почему?

а) ODD(17.1)	г) COS(32.1)
б) COS(5)	д) SIN(0.5)
в) SUCC(3.2)	е) PRED(7)

4. Какие из приведенных выражений являются правильными с точки зрения языка Паскаль?

а) 'A' < 'C'	г) '1' > '9'
б) FALSE < TRUE	д) 'A' < 'B' AND 1 < 2
в) 'O' OR '9'	е) ('D' > 'G') AND (1 < 2)

5. Записать на языке Паскаль следующие арифметические выражения:

а) $\frac{\ln|x-1|}{\sqrt{(x^2-y^2)}}+\frac{\cos x \cdot \sin x}{(a-b)^2}$ б) $\frac{\log(a+b)^3}{1-a\cdot b\cdot e^{\operatorname{arctg} 2x}}$

в) $x+\cfrac{y}{c+\cfrac{x}{d+\cfrac{x}{z+\cfrac{1}{1-x}}}}$

г) $\left(x+\frac{a}{b}\right)^{1/3}-\left|\frac{a^2-b^2}{\sqrt{|c|}}\right|\cdot\sqrt[6]{\left(1-\frac{a}{b}\right)}$

д) $\sqrt{\left(x^3-\sqrt[4]{(a-b)^3}\right)}\cdot\frac{0{,}0001}{\operatorname{arctg}2x\cdot e^n}$

е) $e^{|x^2-y^2|}\cdot\frac{1}{\ln|x^3-y^3|}+\frac{\cos(2x-1)^2}{\sin^2(2x-1)}$

6. Вычислить значения логических выражений при заданных значениях переменных:
а) x>0 AND y=3 OR x + y > 5 AND x-y < 0 при x = 5 и y = 15;
б) a<0 AND b<0 AND c<0 при a = 5, в = -3, c = -25.3;
в) x > 2 OR y < 3 OR z = y AND x + y + z > 0 при x = 12, y = -20, z = 0;
г) x > 0 AND y > 0 OR x < 0 AND y < 0 при x = -2, y = 9;
д) x * x + y * y < 49 AND x * x + y * y > 1 при x = 3, y = 4;
е) (A < B) AND (X + A < B) OR (C < M) OR E при A = 15.5, B = 8.2, C = -10.6, M = 0, X = 7, E = TRUE.

ЛАБОРАТОРНАЯ РАБОТА N2

Тема: "Алгоритмизация линейных вычислительных процессов "

Цель работы

Овладение навыками разработки алгоритмов линейных процессов, умением составлять математические модели алгоритмов и записывать программы на языке Паскаль.

Краткие сведения из теории

2.1. Понятие алгоритма

Определение 1. Алгоритм - это четкое описание последовательности действий, приводящих к решению задачи.

В теории алгоритмов принята следующая классификация алгоритмов: линейные, разветвляющиеся, циклические.

Линейный алгоритм состоит из последовательности действий, каждое из которых выполняется только один раз в порядке их следования.

Существует ряд способов записи алгоритмов; основными из них являются: текстуальный, графический, операторный.

Определение 2. Схема алгоритма - это графическое изображение алгоритма с помощью различных геометрических фигур или блоков. Каждому блоку соответствует определенный этап решения задачи. Блоки соединяются линиями потока информации, причем линии определяют направление вычислений сверху вниз и слева направо. Если необходимо отразить другое направление (снизу вверх или справа налево), то на линиях ставятся стрелки. Внутри блоков записываются выполняемые действия.

Для записи линейных алгоритмов применяются следующие блоки (рис.1):

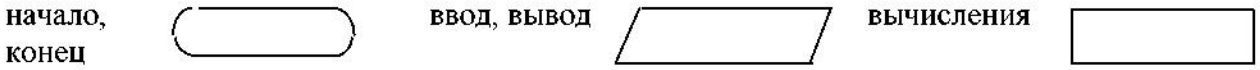


Рис.1. Основные элементы схем линейных алгоритмов.

2.2. Этапы решения задачи на ЭВМ

1. *Словесная постановка задачи.* Этот этап предполагает точную формулировку задачи и цели, которые необходимо достигнуть при ее решении. Определение целей сводится к нахождению исходных и промежуточных величин, а также результатов, т.е.
   дано:
   определить:
   промежуточные величины.
2. *Математическая постановка задачи* заключается в записи условия задачи в виде математических соотношений с использованием выше выбранных переменных. На этом же этапе осуществляется выбор математического метода решения задачи согласно составленной математической модели. Метод должен обеспечить решение задачи выполнением последовательности четырех арифметических операций и функций. Для простых задач обычно метод решения очевиден, и этот этап опускается.
3. *Разработка алгоритма и его графическое описание.* Алгоритмизация задачи представляет собой первый этап программирования. В процессе его выполнения устанавливается необходимая последовательность арифметических и логических действий, с помощью которых реализуется выбранный численный метод. Эту последовательность можно записать в виде схемы алгоритма.
4. *Запись алгоритма на языке программирования.* Алгоритм, записанный в графическом виде, далее записывается в виде последовательности операторов выбранного языка программирования, т.е. составляется программа решения задачи.
5. *Разработка контрольного теста и отладка программы.* Отладка программы необходима для выявления и устранения ошибок, допущенных на предыдущих этапах.

Правильность выполнения программы осуществляется сравнением результатов, полученных при расчете нескольких вариантов задачи вручную и на ЭВМ. Полученный вручную расчет является *контрольным тестом*. Такой метод называется *тестированием* программы.

6. *Работа на ЭВМ с получением результатов.* После решения задачи на ЭВМ необходим анализ результатов. Решение задачи на ЭВМ производится по отлаженной программе для всего множества исходных данных. Анализ результатов выполняется, как правило, лицом, в интересах которого решалась задача.

2.3. Структура программы на языке Паскаль

Язык программирования (алгоритмический язык) - это специальный язык общения человека с ЭВМ. Общение производится следующим образом:
программа записывается средствами языка программирования;
эта запись переводится на язык команд ЭВМ с помощью специальной программы, называемой транслятором или компилятором;
в результате трансляции (перевода) создается программа в машинных кодах (объектный модуль);
объектный модуль компонуется и в результате создается загрузочный модуль, при выполнении которого получается результат.
Программа на алгоритмическом языке состоит из предложений, каждое из которых выполняет определенное действие.
Для описания линейных алгоритмов достаточно применение операторов ввода, вывода и присваивания (назначения).
Программа на языке Паскаль состоит из двух частей: *декларативной и процедурной*.
В *декларативную* часть входят:

заголовок программы;
раздел меток;
раздел констант;
раздел типов;
раздел переменных;
раздел процедур и функций.

Процедурная часть состоит из раздела операторов.
Общая структура программы на Паскале имеет следующий вид:

```
PROGRAM имя (INPUT,OUTPUT);        - заголовок
LABEL                  - раздел меток
            . . .
            CONST              - раздел констант
             . . .
TYPE                   - раздел типов
            . . .
VAR                    - раздел переменных
            . . .
PROCEDURE,FUNCTION             - раздел процедур и функций
    . . .
BEGIN
            оператор 1;
            оператор 2;
            ...
            оператор n-1;
            оператор  n
END.
```

Заголовок содержит служебное слово PROGRAM, имя программы, задаваемое пользователем, и в круглых скобках имена стандартных процедур INPUT, OUTPUT для

связи с внешними стандартными устройствами ввода-вывода ЭВМ. Заголовок заканчивается символом "точка с запятой".

Декларативная часть программы предназначена для объявления всех встречающихся в программе данных. После каждого объявления данного или метки ставится точка с запятой. В стандартном Паскале разделы декларативной части должны располагаться в строго указанном выше порядке, в Турбо-Паскале строгий порядок не обязателен. Следует заметить, что не все перечисленные разделы должны обязательно присутствовать в программе. В простых программах, например, могут потребоваться только разделы объявления констант и переменных. Некоторые программы могут вообще не содержать разделы декларативной части.

Раздел операторов заключается в операторные скобки BEGIN (начало) и END (конец), при этом после END ставится точка. В этом разделе записывается последовательность исполняемых операторов, после каждого из которых ставится точка с запятой.

Примечание:

a) перед словом END точку с запятой можно опускать;
b) указание стандартных устройств в заголовке программы не обязательно в Турбо-Паскале, начиная с версии 4.0 и выше.

2.4. Объявление констант и переменных

Для записи линейных программ в разделе описаний может потребоваться объявление констант и переменных.

Объявление констант имеет вид: ключевое слово CONST, после которого записываются имена используемых констант и их значения.

Имена и значения разделяются знаком равенства. Каждое определение константы заканчивается точкой с запятой.

Примеры:
```
CONST
  NMIN = 1;  NMAX = 100;
  LIM = C;
```

Далее в разделе операторов используется только имя константы. Если значение константы меняется, то изменение производится только в разделе констант, что очень удобно. Если в программе не используются константы, то этот раздел отсутствует.

Описание переменных начинается с ключевого слова VAR, за которым идет перечисление имен всех переменных программы с указанием их типов, разделенных символом "двоеточие".

Общий вид описания переменных:

```
VAR
   V1,V2,V3,...,Vn: T;
```

где V1,V2,V3,...,Vn - имена переменных данного типа;

T - тип переменных.

В разделах описаний данное может быть описано только один раз.

Примеры описания переменных стандартного типа.

```
VAR
A, B, C      : real;      {переменные действительного типа}
N, K, L, M   : integer;   {переменные целого типа}
D, BETA      : boolean;   {переменные логического типа}
Stroka       : string[10]; {переменные строкового типа, в квадратных скобках указана максимальная длина переменной}
```

Для линейных программ достаточно применения операторов ввода, вывода и присваивания.

2.5. Операторы для записи линейных процессов

2.5.1. Оператор присваивания

Этот оператор относится к простым операторам, т.к. не содержит внутри себя других операторов.

Синтаксис оператора присваивания:

V : = A;

здесь V - имя переменной;

" : = " - знак присваивания;

A - выражение.

Данный оператор вычисляет значение выражения A и присваивает полученное значение переменной V. Выражение в правой части может быть арифметическим, логическим и символьным. Поэтому при использовании оператора присваивания нужно следить за тем, чтобы переменная в левой части и выражение в правой части были одного и того же типа. Из этого правила существует одно исключение: разрешается при целочисленном выражении использовать переменную действительного типа в левой части. В этом случае значение вычисленного выражения будет преобразовано в действительный тип.

Примеры:

```
T: =   527.475;
M: =   TEM;
Y: =   SQRT(T);
L: =   A and B;
ST: =  'PASCAL';
C5: =  2*K-SIN(PI/4-X);
```

Здесь T, Y, C5 имеют действительные значения и должны быть описаны в разделе переменных как данные типа REAL.

Переменная ST должна иметь строковый тип в разделе описаний VAR; переменные M и TEM должны иметь любые одинаковые типы, а переменные K и X могут быть либо действительного типа, либо целого.

Неверные записи:

а)
```
VAR
    D : REAL;
beqin
    D : = '7'
end.
```

б)
```
VAR
    A,B,C,D : REAL;
beqin
    A : = (B<C) and (D>C)
end.
```

В примере а) переменной D типа REAL присваивается константа типа CHAR. Это вызовет ошибку "несоответствие типа данных".

В примере б) переменной A действительного типа присваивается значение логического выражения, что также приведет к ошибке "несоответствие типов данных".

2.5.2. Оператор ввода числовых данных

Ввод числовых данных на Паскале выполняется операторами READ и READLN. Общий вид операторов следующий:

```
READ(a1, a2 ,..., an);
READLN;
READLN(a1, a2, ... , an); ,
```

где a1, a2,..., an - имена переменных, которым последовательно присваиваются вводимые с клавиатуры числовые значения.

При выполнении оператора READ вычислительная система приостанавливает работу и ждет ввода значений переменных, указанных в данном операторе. Если в операторе указано две и более переменных, то значения этих переменных при наборе на клавиатуре печатаются через пробел. Также допускается применение оператора ввода без параметров - READLN. Этот оператор выполняет переход на новую строку.

При выполнении оператора READLN(a1, a2 ,..., an) вводятся значения всех переменных, а затем осуществляется переход на новую строку.

Следует помнить, что значения переменных и их имена должны соответствовать одному и тому же типу.

2.5.3. Оператор вывода числовых данных

Вывод числовых данных выполняется операторами WRITE и WRITELN, которые имеют вид:
WRITE(b1, b2 ,..., bn);
WRITELN;
WRITELN (b1, b2 ,..., bn);.
Оператор WRITE выполняет вывод значений переменных b1, b2,..., bn в одной строке.
Оператор WRITELN без параметров осуществляет переход на начало новой строки.
Оператор WRITELN с параметрами после вывода последнего значения осуществляет переход на начало новой строки.
Примеры:
1) Write(a, b); Write(c); - вывод a, b, c в одну строку;
2) Writeln(a, b); Write(c); - вывод a и b на одной строке, c- на следующей строке.

2.5.4. Вывод числовых данных с форматом и без формата

1. Вывод данных без формата.
В случае бесформатного вывода значения целого типа выводятся в виде последовательности цифр и знака; значение действительного числа выводится в форме с плавающей точкой (или экспоненциальной форме) с указанием мантиссы и порядка.
2. Вывод данных с форматом.
Формат вывода указывается после переменной через двоеточие - :L:D. Для целых чисел формат состоит из одной величины L, равной количеству цифр в числе и символа под знак. Для вещественных чисел формат состоит из двух величин, указывающих соответственно общее поле выводимого значения L и количество цифр в дробной части D. L включает позиции под знак, целую часть, десятичную точку и дробную часть числа.
Синтаксис:
WRITE (V:L) - для целых переменных;
WRITE (V:L:D) - для вещественных переменных.
Здесь V - переменная, значение которой выводится на экран;
L - общая длина поля числа;
D - количество символов в дробной части числа.

Пример:
а) если X = 835, то результатом работы оператора WRITE(X:6) будет: _ _ _835;
б) если Y = 28.63, то результатом работы оператора WRITE(Y:6:2) будет _28.63.
Примечание:

a) Если количество указанных позиций недостаточно, то происходит автоматическое увеличение поля до нужных размеров; если же длина поля больше, чем позиций в числе, то лишние позиции заполняются пробелами, причем перед целой частью и после значащих цифр в дробной части числа.
b) Если указывается общее число позиций L, но не указывается число позиций после запятой D, то число выводится в экспоненциальной форме с шириной поля L.
c) С помощью оператора вывода можно выводить любую строку символов, заключенную в апострофы.

Пример:
WRITE('значение B=', B);
WRITE('введите значения X, Y:'); .
Эту возможность можно использовать для вывода в программе данных пояснительного характера (комментариев).

2.6. Разработка алгоритмов линейных процессов

Задание А.
Словесная постановка задачи.
Определить площадь и длину второго катета прямоугольного треугольника, если известны длина одного катета и длина гипотенузы.

Дано: a - длина катета,
c - длина гипотенузы.
Определить: b - длина искомого катета,
S - площадь треугольника.
Промежуточные величины: нет.
Математическая постановка.
По теореме Пифагора имеем:
$c^2 = a^2 + b^2$, откуда $b = \sqrt{(c^2 - a^2}$.

Тогда площадь $S = \frac{a * b}{2}$

Разработка схемы алгоритма.

В первую очередь необходимо вычислить $b = \sqrt{(c^2 - a^2}$, затем $S = \frac{a * b}{2}$.

Схема алгоритма будет иметь вид (рис.2):

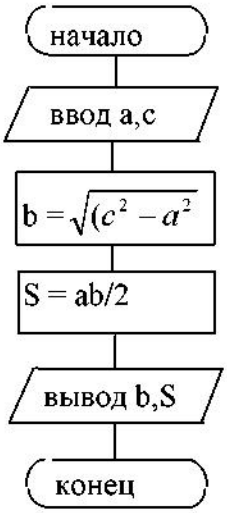


Рис.2 Схема алгоритма

Так как алгоритм состоит из последовательно выполняемых действий, то он описывает линейный процесс.

Задание Б.

Словесная постановка.

Определить длину ломаной линии, состоящей из трех звеньев, если известны координаты вершин (рис.3).

Дано: координаты вершин x1,y1,x2,y2,x3,y3,x4,y4.

Определить: L - длина ломаной.

Промежуточные величины: L1, L2, L3 - длины звеньев ломаной.

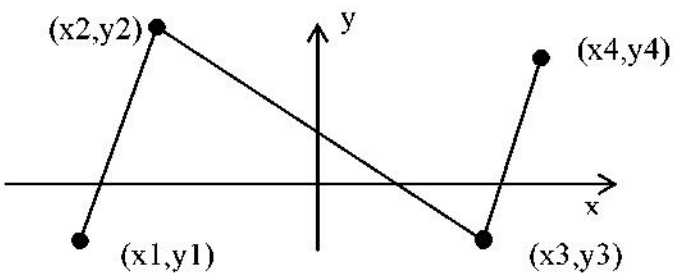


Рис.3

Математическая постановка.

Для определения длин составляющих звеньев воспользуемся формулой для вычисления расстояния между двумя точками плоскости.

$L1 = \sqrt{((x2 - x1)^2 + (y2 - y1)^2)}$, $L2 = \sqrt{((x3 - x2)^2 + (y3 - y2)^2)}$,

$L3 = \sqrt{((x4 - x3)^2 + (y4 - y3)^2)}$, откуда $L = L1 + L2 + L3$.

Разработка схемы алгоритма.

Схема алгоритма будет иметь вид:

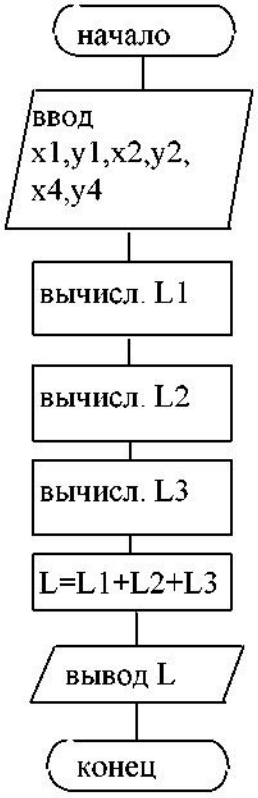


Рис.4. Схема алгоритма

Задание В.

Словесная постановка задачи.

Тело брошено вертикально вверх с некоторой скоростью V. Определить время, за которое тело пролетит заданное расстояние S.

Дано: V - скорость; S - расстояние; g = 9,8 - ускорение свободного падения.

Определить: t - время.

Промежуточные величины: D.

Математическая постановка задачи.

Расстояние рассчитывается по формуле, известной из физики:

$$S = Vt - \frac{gt^2}{2};$$

отсюда $2S = 2Vt - gt^2$.

Так как нам нужно определить t, запишем это выражение в виде квадратного уравнения:

$gt^2 - 2Vt + 2S = 0$.

Получена математическая модель задачи в виде квадратного уравнения.

Выбор численного метода решения задачи.

Этот этап в предыдущих задачах отсутствовал, т.к. метод решения был очевиден (с использованием полученных расчетных формул). В этой задаче необходимо применить формулы решения квадратного уравнения.

В общем виде уравнение имеет вид: $ax^2 + bx + c = 0$.

Определим дискриминант и корни: $D = b^2 - 4ac$; $x1 = \frac{-b + \sqrt{D}}{2a}$; $x2 = \frac{-b - \sqrt{D}}{2a}$.

Для данной задачи справедливы следующие соотношения:

$a = g$; $b = -2v$; $c = 2s$; $x = t$.

Тогда: $D = 4v^2 - 4g \cdot 2s = 4v^2 - 8gS$; $t1 = \frac{2v + \sqrt{D}}{2g}$; $t2 = \frac{2v - \sqrt{D}}{2g}$

Разработка схемы алгоритма.

Вычисляем последовательно D, t1, t2. Схема алгоритма приведена на рис.5.

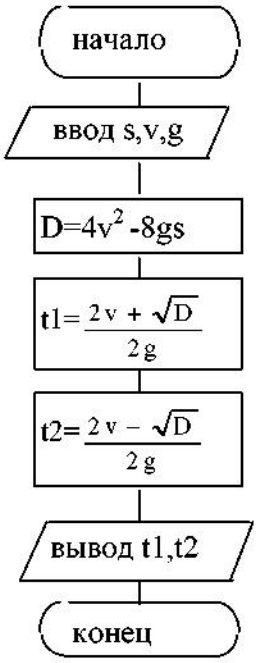


Рис.5. Схема алгоритма

2.7. Примеры программ линейных процессов

Запишем программы рассмотренных выше задач.

Задание А.

```
Program TREYG;
Var
    a, b, c, s : real;
Begin
   write('введите длину катета: ');
   readln(a);
   write('введите длину гипотенузы: ');
   readln(c);
   b : = SQRT(c * c - a * a);
   S : = a * b / 2;
   WRITELN('Длина второго катета=', b:6:1);
   WRITELN('Площадь треугольника=', s:8:2)
End.
```

Контрольный тест.

Допустим $a = 5$; $c = 6$.

Тогда $b = \sqrt{(36 - 25)} = \sqrt{11} \approx 3{,}3$;

S = 5 * 3.3 /2 ≈ 8,25.

Протокол работы на ЭВМ.

Введите длину катета : 5 <ENTER>
Введите длину гипотенузы : 6 <ENTER>
Длина второго катета = 3.3
Площадь треугольника = 8.25
Результаты работы программы и ручного счета совпали.

Задание Б.

```
Program DLINA;
Var
    x1, y1, x2, y2, x3, y3, x4, y4 : real;
    L, L1, L2, L3                  : real;
Begin
    Write('Введите координаты вершин x,y :');
     readln(x1, y1, x2, y2, x3, y3, x4, y4);
     L1 := SQRT(SQR(x2 - x1) + SQR(y2 - y1));
     L2 := SQRT(SQR(x3 - x2) + SQR(y3 - y2));
     L3 := SQRT(SQR(x4 - x3) + SQR(y4 - y3));
     L := L1 + L2 + L3;
    Writeln('L1 = ', L1:4:1, ' L2 = ', L2:4:1, ' L3 = ', L3:4:1);
    Writeln('Длина ломаной =', L:6:1)
End.
```

Контрольный тест.

Пусть A(1, 1); B(4, 5); C(7, 3); D(10, 7). Тогда

$L1 = \sqrt{((4-1)^2 + (5-1)^2)} = \sqrt{(9+16)} = 5;$

$L2 = \sqrt{((7-4)^2 + (3-5)^2)} = \sqrt{(9+4)} = \sqrt{13} \approx 3{,}6;$

$L3 = \sqrt{((10-7)^2 + (7-3)^2)} = \sqrt{(9+16)} = 5;$

$L = 5 + 3{,}6 + 5 \approx 13{,}6.$

Протокол работы на ЭВМ.

Введите координаты вершин x,y : 1 1 4 5 7 3 10 7
L1 = 5.0 L2 = 3.6 L3 = 5.0
Длина ломаной = 13.6

Задание В.

```
Program TELO;
Const
    g = 9.8;
Var
    s, v, t1, t2, D  : real;
Begin
    write('Введите расстояние:');
    readln(s);
    write('Введите скорость:');
    readln(v);
   D  : = 4 * v * v - 8 * g * s;
   t1 : = (2 * v + SQRT( D )) / (2 * g);
   t2 : = (2 * v - SQRT( D )) / (2 * g);
   Writeln('t1 = ', t1:4:2, 't2 = ', t2:4:2)
End.
```

В этой задаче при анализе решения отрицательные значения времени отбрасываются.

Контрольные вопросы

1. Что называется алгоритмом?
2. Перечислите типы алгоритмов.
3. Дайте определение линейного алгоритма.
4. Перечислите способы описания алгоритмов.
5. Что называется схемой алгоритма?
6. Какие блоки используются для описания линейных алгоритмов?
7. Опишите этапы решения задачи на ЭВМ.
8. Дайте понятие контрольного теста.
9. Дайте понятие алгоритмического языка.
10. Что называется транслятором?
11. Что такое объектный модуль?
12. Что представляет собой программа на алгоритмическом языке?
13. Дайте понятие оператора.
14. Опишите структуру программы на Паскале.
15. Поясните смысл объявления констант и переменных.
16. Опишите выполнение оператора присваивания.
17. Объясните выполнение оператора ввода.
18. Что такое бесформатный вывод данных?
19. Какие форматы вывода данных вам известны?
20. В следующих операторах присваивания укажите типы переменных:

```
a  : = 32;
x  : = sin(y + 2.3);
l  : = (k > 3) and (d < c);
m : = 'Сегодня' + t;
b  : = '1993'.
```

Задание к работе

Выполнить индивидуальное задание.

Методические указания

Задачу необходимо решать согласно рассмотренной технологии:

а) изучить словесную постановку задачи;
б) сформулировать математическую постановку задачи;
в) выбрать метод решения задачи, если это необходимо;
г) разработать схему алгоритма;
д) записать разработанный алгоритм на языке Паскаль;
е) разработать контрольный тест программы;
ж) отладить программу;
з) написать отчет.

Содержание отчета

1. Титульный лист.
2. Словесная постановка задачи.
3. Математическая формулировка задачи.
4. Схема алгоритма решения задачи.
5. Листинг программы.
6. Контрольный тест.
7. Результат тестирования программы.
8. Ответы на контрольные вопросы к лабораторной работе по согласованию с преподавателем.

Варианты индивидуальных заданий

1. Треугольник задан длинами сторон. Найти длины высот.
2. Треугольник задан длинами сторон. Найти радиусы вписанной и описанной окружностей.
3. Найти корни системы двух алгебраических уравнений с двумя неизвестными по формуле Крамера, считая, что система разрешима.
4. Хозяин хочет оклеить обоями квартиру. Определить количество необходимых рулонов и затрат на их приобретение.
5. В некотором треугольнике основание больше высоты на заданную величину d. Определить значение высоты и основания, если известна величина площади треугольника.
6. Известны координаты вершин треугольника. Определить периметр и площадь треугольника.
7. Имеется кольцо с известными радиусами внутреннего и внешнего кругов. Определить длины окружностей обоих кругов, а также площадь кольца.
8. Дано действительной число a. Не пользуясь никакими другими арифметическими операциями, кроме умножения, получить a^8 за три операции.
9. Дано действительной число a. Не пользуясь никакими другими арифметическими операциями, кроме умножения, получить a^5 и a^{13} за пять операций.
10. Определить площадь равнобедренной трапеции, если заданы основание b, высота h и угол при основании b, равный L.

ЛАБОРАТОРНАЯ РАБОТА N 3

Тема: "Разветвляющиеся вычислительные процессы"

Цель работы

Получение навыков разработки алгоритмов и программирования разветвляющихся вычислительных процессов.

Краткие сведения из теории

3.1. Типы разветвляющихся алгоритмов

Второй типовой структурой алгоритмов является разветвляющийся вычислительный процесс, в котором направление вычислений зависит от результата проверки некоторого условия. Направления, по которым может следовать вычислительный процесс, называются ветвями. В программах разветвляющихся процессов естественный порядок выполнения операторов нарушается и обеспечивается выполнение той последовательности операторов, которая соответствует выбранным условиям. В языке Паскаль это реализуется специальными управляющими операторами или операторными структурами, которые называются операторами перехода.

Схему алгоритма разветвленной структуры характеризует наличие блока "условие", который имеет два выхода, помеченные словами "да" и "нет". Еще этот блок называют логическим блоком. В этом блоке осуществляется проверка выполнения некоторого логического условия. Если условие "истинно", вычислительный процесс идет по выходу "да", в противном случае - по выходу "нет".

Различают три типа разветвляющихся алгоритмов, функциональные схемы которых приведены на рисунке 6.

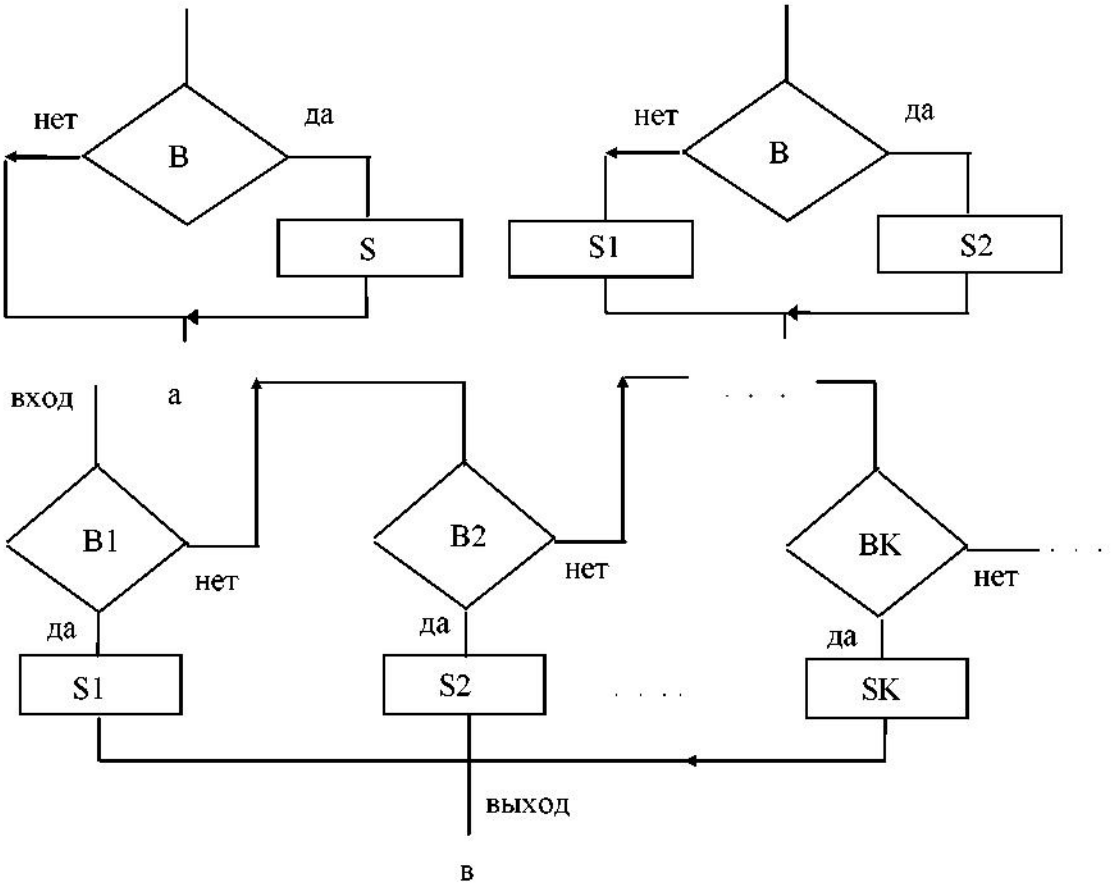


Рис.6. Функциональные схемы разветвляющихся процессов

Ветвление, представленное на рис.6(а), называется обходом, так как оператор S, записанный в арифметическом блоке, не выполняется, если условие B ложно. При реализации вычислительного процесса арифметический блок будет обойден, и направление вычислений пойдет по ветви "нет".

Выбор из двух возможностей, или альтернатива, представлен на рис.6(б). Если проверяемое условие B будет истинным, выполнится оператор S1, в противном случае выполнится оператор S2. Отметим, что алгоритм обхода является частным случаем альтернативы.

Выбор из многих возможностей представлен на рис.6(в). Здесь Bi, i=1,K представляют собой условия выбора, в зависимости от значений которых выполняется один из соответствующих им операторов S1, S2, . . , SK.

3.2. Операторы перехода

Назначение операторов перехода состоит в организации безусловных и условных переходов в программе в зависимости от результата проверки логического условия B.

3.2.1. Операторная запись обхода

Разветвление такого типа может быть записано с помощью условного оператора, который имеет вид:

IF < выражение > THEN < оператор > ,

где IF ("если") и THEN ("то") - ключевые слова;

< выражение > - логическое условие;

< оператор > - оператор.

При решении практических задач бывает необходимо в зависимости от результата проверки, выполнить (или обойти) группу операторов. Язык Паскаль предоставляет возможность сделать из группы операторов один составной оператор, заключив группу операторов в операторные скобки Begin и End.

Тогда условный оператор для записи обхода группы операторов имеет вид:

IF < выражение > THEN Begin < S1, S2, . . , SN > End;.

Пример. Вычислить и вывести на экран корень квадратный из положительного числа X, значение которого не превышает 1000. В случае отрицательного числа никаких вычислений в программе не производится.

Дано: X - исходное число.

Математическая запись задачи:

$$Y = \begin{cases} \sqrt{X} & \text{, если } X >= 0 \\ \text{нет вычислений, если } X < 0. \end{cases}$$

Схема алгоритма решения задачи приведена на рисунке 7.

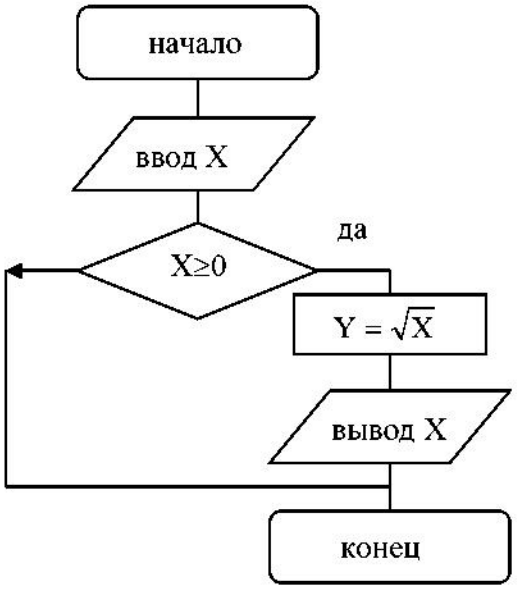


Рис.7

Программа на языке Паскаль (операторная запись алгоритма).

```
Program KOR(input, output);
 VAR  {описание действительных переменных X и Y }
```

```
  X, Y   : real;
Begin
  Readln( X );  { ввод переменной X }
  If  X >= 0 Then
          Begin         { составной оператор }
            Y :=SQRT ( X );
            WriteLn( ' Корень из X равен ', Y : 6:2)
          End
End.
```

3.2.2. Операторная запись альтернативы

Выбор из двух возможностей реализуется условным оператором, имеющим следующий синтаксис:

IF < выражение > THEN < оператор 1 > ELSE < оператор 2 >;

где ELSE ("иначе") - ключевое слово.

Если < выражение > истинно, то выполняется < оператор 1 >, в противном случае - < оператор 2 >.

Оператор, стоящий после ключевого слова ELSE, может быть любым оператором, в том числе и условным. Кроме того, < оператор 1> и <оператор 2> могут быть составными операторами.

Ниже приведена одна из возможных конструкций условного оператора:

IF B1 THEN Begin <S1, S2> End ELSE If B2 Then <S3> Else <S4>;

Здесь B1 и B2 - логические условия. < Оператор 1 > представляет собой составной оператор, а < оператор 2 > - условный оператор If - Then - Else. Если логическое условие B1 истинно, выполнятся S1 и S2, и управление будет передано следующему в программе оператору. Если логическое условие B1 ложно, выбирается < оператор 2 >. При его выполнении проверяется условие B2: в случае истинности выражения B2 выполняется оператор S3, в противном случае - S4.

Пример.

Словесная постановка задачи. Вычислить корни квадратного уравнения. Квадратное уравнение имеет вид:

$aX^2 + bX + c = 0.$

Словесное описание задачи:

1. Ввести значения коэффициентов квадратного уравнения a, b и c.
2. Вычислить значение $D = b^2 - 4ac$.
3. Сравнить D с нулем. Если D < 0, перейти к п. 6.
4. Вычислить $X1 = (-b + \sqrt{D})/(2a)$; $X2 = (-b - \sqrt{D})/(2a)$.
5. Вывести на печать значения X1 и X2. Перейти к п. 7.
6. Вывести сообщение об отсутствии действительных корней уравнения.
7. Прекратить вычисления.

Схема алгоритма решения задачи приведена на рисунке 8.

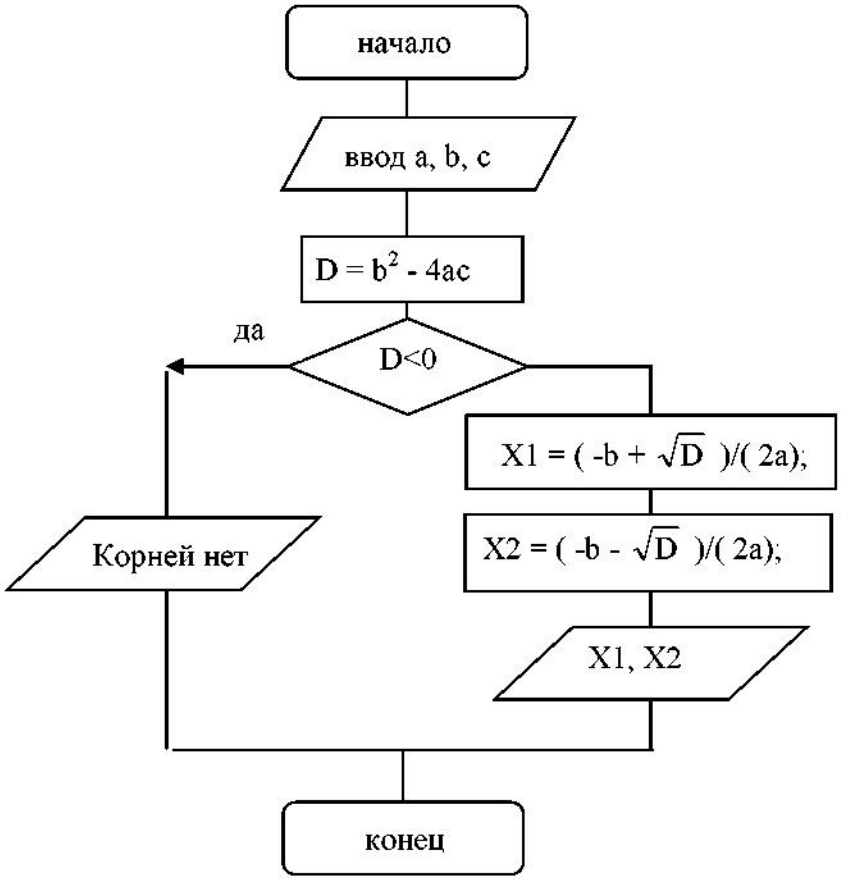


Рис.8. Схема алгоритма

Операторная запись задачи.

```
PROGRAM KORNY(input, output);
VAR   {описание переменных }
      a, b, c      : real;
      X1, X2, D    : real;
BEGIN
   { ввод коэффициентов квадратного уравнения }
   ReadLn( ' Введите коэффициенты a, b, c: ', a, b, c );
   { вычисление дискриминанта D }
   D = b * b - 4 * a * c;
   WriteLn;
   { условный оператор }
   IF D < 0 THEN
        WriteLn( ' Уравнение не имеет действительных корней.')
        ELSE
          Begin
            X1 := ( -b + SQRT(D) ) / 2* a;
            X2 := ( -b - SQRT(D) ) / 2 *a;
            WriteLn( ' X1= ', X1 : 7:2, ' X2= ', X2 : 7:2 )
          End
END.
```

3.2.3. Операторная запись алгоритма выбора

Обычно при написании программ не рекомендуется использовать многократно вложенные друг в друга условные операторы, так как программа становится громоздкой и ее трудно понимать. Считается, что число уровней вложения не должно превышать двух - трех. Для тех случаев, когда необходимо проверять много условий и в зависимости от них выполнять различные действия, в языке Паскаль существует специальный оператор выбора.

Синтаксис оператора выбора:

CASE < индекс выбора > OF < элементы списка выбора > [ELSE <оператор>] END;

Индекс выбора состоит из выражения. Элементы списка выбора включают список операторов, каждый из которых помечен одной или несколькими константами выбора. Все константы выбора должны быть различными, а их тип должен быть идентичен типу выражения (индекса выбора). В качестве типа выражения, следовательно, и констант выбора, можно использовать любой простой тип, за исключением типа REAL.

Выполнение оператора выбора начинается с вычисления значения индекса выбора. Затем выбирается для выполнения тот оператор из списка элементов выбора, который помечен константой выбора, имеющей вычисленное значение индекса выбора. Если константы с таким значением нет, то управление передается оператору, стоящему за зарезервированным словом ELSE. Конструкция ELSE <оператор> может отсутствовать. В этом случае при отсутствии в списке выбора нужной константы ничего не произойдет и оператор выбора просто завершит работу.

Рассмотрим фрагменты программ, содержащие оператор выбора.

Пример 1.

```
PROGRAM EXAMPL1( input, output );
    VAR
     I    : INTEGER;
     X    : REAL;
      . . .
  BEGIN
      . . .
     CASE I OF
       1 : X := Sin( X );
       2 : X := Cos( X );
       3 : X := Exp( X );
       4 : X := Ln( X )
     End;
      . . .
  END.
```

Пример 2.

```
PROGRAM EXAMPL2;
VAR
    I    : INTEGER;
    X    : REAL;
     . . .
BEGIN
     . . .
    CASE  I  OF
          1 : X := Sin( X );
          2 : X := Cos( X );
          3 : X := Exp( X );
          4 : X := Ln( X )
    ELSE
```

```
        WriteLn ( ' Значение индекса выбора не соответствует заданным константам .' );
    END
END.
```

Пример 3.

В элементе списка выбора можно использовать несколько констант выбора, а также диапазоны:

```
CASE XCH OF
  'A', 'C', 'E', 'G' : WriteLn( 'Указано  несколько констант');
  'K' . . 'R'        : WriteLn( 'Указан интервал')
END;
```

В этом операторе переменная XCH символьного типа.

В заключение отметим, что:

- в качестве элемента списка выбора можно использовать составной оператор;
- все константы выбора внутри одного оператора выбора обязательно должны быть различными, так как в противном случае возникает неоднозначность в выборе исполняемого оператора;
- в разных операторах выбора разрешается использовать одинаковые константы выбора.

3.2.4. Метка. Оператор перехода. Пустой оператор.

Каждый оператор в программе может быть помечен *меткой* – произвольным идентификатором. Метка позволяет именовать некоторый оператор программы и таким образом ссылаться на него. Метка располагается перед оператором и отделяется от него двоеточием.

Например:

```
10:  ReadLn( ' Введи значение переменной A:', A);
124: Y := X * X + S * Z;
```

Метки не влияют на выполнение оператора. Они должны быть описаны в разделе описания меток. Описание меток состоит из ключевого слова LABEL и следующего за ним списка меток.

Например:

```
LABEL 10, 124, 540, L1, L2;
```

Описания меток располагаются до совокупности всех описаний переменных. Описанной меткой должен быть помечен только один оператор программы.

Оператор перехода прерывает естественный порядок выполнения операторов. Он состоит из ключевого слова GOTO, за которым следует метка. Дальнейшее выполнение программы должно продолжаться, начиная с оператора, помеченного указанной меткой.

В языке Паскаль существует довольно строгая дисциплина использования операторов перехода. Сформулируем эти ограничения:

1) с помощью оператора перехода нельзя войти внутрь составного оператора, но внутри составного оператора разрешены любые передачи управления;
2) с помощью оператора перехода нельзя войти внутрь тела цикла, минуя его заголовок. Внутри тела цикла разрешены любые передачи управления;
3) с помощью оператора перехода нельзя войти ни в одну из ветвей условного оператора, а также передать управление из одной ветви в другую;
4) с помощью оператора перехода нельзя войти внутрь оператора выбора или передать управление на другую константу выбора;
5) с помощью оператора перехода нельзя войти в тело процедуры;
6) с помощью оператора перехода можно выйти из любой конструкции языка программирования за единственным исключением: если в программе содержатся многократно вложенные друг в друга процедуры, то из любой внутренней процедуры можно выйти с помощью оператора перехода, ведущего к метке, которой помечен оператор

на самом внешнем уровне вложенности блоков (то есть выйти можно только в главную программу).

Пустой оператор в программе обозначается точкой с запятой. Чаще всего он встречается с меткой и ставится в конце составного оператора или программы.

Контрольные вопросы

1. Дайте классификацию разветвляющихся алгоритмов.
2. Какие блоки используются для описания разветвляющихся алгоритмов?
3. Какой оператор используется для записи обхода?
4. Какой оператор используется для записи альтернативы?
5. Каково назначение оператора выбора и в каких случаях он используется?
6. Каково назначение оператора перехода?
7. Сформулируйте ограничения использования оператора перехода.
8. Что такое метка и для чего она предназначена?
9. Можно ли алгоритм выбора из многих возможностей записать с помощью оператора условного перехода?

Задание к работе

Разработать программу в соответствии с индивидуальным вариантом задания.

Методические указания

При выполнении индивидуального задания необходимо соблюдать рассмотренную технологию решения задач на ЭВМ:

1) изучить словесную постановку задачи, выделив при этом все виды данных;
2) сформулировать математическую постановку задачи;
3) выбрать метод решения задачи, если это необходимо;
4) разработать графическую схему алгоритма;
5) записать разработанный алгоритм на языке Паскаль;
6) разработать контрольный тест к программе;
7) отладить программу;
8) представить отчет по работе к защите.

Содержание отчета

1. Титульный лист.
2. Словесная постановка задачи.
3. Математическая постановка задачи.
4. Листинг программы.
5. Контрольный тест.
6. Ответы на контрольные вопросы по согласованию с преподавателем.

Варианты индивидуальных заданий

1. Дано натуральное число n ($n \le 9999$). Является ли это число палиндромом (перевертышем) с учетом четырех цифр, как, например, числа 2222, 6116, 0440 и т.д.?
2. Дано натуральное число n ($n \le 100$), определяющее возраст человека (в годах). Дать для этого числа наименования «год», «года», «лет». Например, 1 год, 23 года, 46 лет и т.д.
3. Поле шахматной доски определяется парой натуральных чисел, каждое из которых не превосходит восьми: первое число – номер вертикали (при счете слева направо), второе – номер горизонтали (при чете снизу вверх) Даны натуральные числа k, l, m, n, каждое из которых не превосходит восьми. Требуется выяснить, являются ли поля (k, l) и (m, n) полями одного цвета.
4. Поле шахматной доски определяется парой натуральных чисел, каждое из которых не превосходит восьми: первое число – номер вертикали (при счете слева направо), второе – номер горизонтали (при чете снизу вверх) Даны натуральные числа k, l, m, n, каждое из которых не превосходит восьми. На поле (k, l) расположен ферзь. Требуется выяснить, угрожает ли он полю (m, n).

5. Даны действительные положительные числа a, b, c, x, y. Выяснить, пройдет ли кирпич с ребрами a, b, c в прямоугольное отверстие со сторонами x, y. Просовывать кирпич в отверстие разрешается только так, чтобы каждое из его ребер было параллельно или перпендикулярно каждой из сторон отверстия.
6. Можно ли из круглой заготовки радиусом r вырезать две прямоугольные пластинки размером a×b, c×d?
7. Среди целых чисел k, l, m найти пары кратных.
8. Число делится на 3, если сумма его цифр делится на 3. Проверить этот признак на примере заданного трехзначного числа.
9. Можно ли на прямоугольном участке застройки размером a×b метров разместить два дома размером в плане p×q, r×s метров? Дома можно располагать только параллельно сторонам участка.
10. Можно ли коробку размером a×b×c упаковать в посылку размером r×s×t. «Углом» укладывать нельзя.

ЛАБОРАТОРНАЯ РАБОТА N4

Тема: " Перечислимые и ограниченные типы данных "

Цель работы:

Ознакомиться с перечислимыми типами, получить навыки в организации ввода-вывода значений переменных перечислимого типа данных.

Краткие сведения из теории

В лабораторной работе N 1 рассмотрены базовые типы данных: целые, вещественные, символьные и булевские. В языке Турбо-Паскаль с помощью базовых типов можно определить такие типы, как перечислимые и ограниченные.

4.1. Перечислимые типы

Перечислимый тип данных задается в виде перечисления в строго определенном порядке и в строго ограниченном количестве всех значений, которые могут принимать данные рассматриваемого типа.

Перечислимый тип состоит из списка констант. Переменные этого типа могут принимать значения любой из этих констант. Описание перечислимого типа имеет вид:

```
Type
    < имя типа > = <( список констант)>;
Var
    < имя переменной >: < имя типа >;
```

Под константой понимается особый вид констант, задаваемый пользователем. Под списком понимается перечень констант, разделенных запятыми. Сам список заключается в круглые скобки. Например:

```
Type
    year = (winter, spring, summer, autumn);
Var
    avar : year;
```

Здесь year - имя перечислимого типа, winter ,... - константы; avar - переменная, которая может принимать значение любой из констант. Допускается указывать константы перечислимого типа непосредственно в разделе описания переменных без использования раздела TYPE, например:

```
Var
     avar : (winter, spring, summer, autumn);
```

Каждая из констант имеет порядковый номер, который начинается нуля. Так, winter имеет порядковый номер 0, а autumn - 3. Упорядоченность констант позволяет применять к ним операции отношений <, <=, =, <>, >=, >, а также стандартные функции:

ORD(avar) - для определения порядкового номера указанного элемента.

PRED(avar) - для определения элемента, являющегося предыдущим для указанного элемента.

SUCC(avar) - для определения элемента, являющегося следующим для указанного элемента.

При работе с данными перечислимого типа необходимо помнить следующие правила:

1. Нельзя считывать или печатать значения перечислимого типа, например:

```
    Read(avar);     { ошибка }
    Writeln(avar);  { ошибка }
```

2. С данными перечислимого типа нельзя выполнять арифметические действия, например:

```
     avar := winter + summer;  { ошибка }
```

3. Тип boolean - перечислимый тип, определение которого автоматически присутствует в программе, например:

```
   Type
    boolean = (false,true);
```

причем ORD(false) = 0, ORD(true) = 1, т.е. false < true.

4.2. Ограниченные типы

Если переменная принимает не все значения своего типа, а только значения, содержащиеся в некотором ограниченном диапазоне, то ее можно объявить как переменную ограниченного типа, например:

```
Type
    mounth = (jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec);
    summer = jun..aug;  { ограниченный тип }
```

В данном примере тип mounth, из которого выделен тип summer, является базовым относительно типа summer.

При определении ограниченного типа указывается начальное и конечное значения, которые может принимать константа базового типа. Граничные константы разделяются двумя точками. Описание ограниченного типа имеет вид:

```
Type
    < имя типа > = < левая константа .. правая константа >;
```

При использовании ограниченных типов данных необходимо помнить следующие правила:

Обе граничные константы должны быть одинакового типа.

В качестве базового типа можно использовать любой простой тип, кроме действительного типа. Например, возможны отрезки:

целого типа - index = 0..63;

символьного типа - letter = 'A'..'Z';

перечислимого типа - summer = jun..aug.

Левая граничная константа должна быть меньше правой.

Переменные ограниченного типа должны быть описаны в разделе переменных с помощью имен этих типов. Например:

```
Var
   simv : letter;
   pole : index;
```

Переменные ограниченного типа, как и перечислимого, можно описывать, не обращаясь к разделу описания типов. Например:

```
Var
   simv : 'A'..'Z';
   pole :  0 .. 63;
```

Интервалы перечислений подчинены тем же самым ограничениям, что и сам перечислимый тип. Так элементы интервала jun..aug не могут вводиться или выводиться, к ним нельзя применять арифметические действия, их нельзя присваивать переменным никаких других типов, кроме summer, так как summer - базовый тип, по отношению к которому jun..aug является интервалом.

Константы любого ограниченного типа имеют порядковые номера, совпадающие с их порядковыми номерами в базовом типе. Например:

ORD(jun) - имеет порядковый номер 5;

ORD(aug) - имеет порядковый номер 7.

Если базовый тип переменных ограниченного типа INTEGER, то значения переменных могут обрабатываться как целые числа. Такая обработка включает чтение, запись и целую арифметику. Например:

```
Read(pole);
score := sqr(pole);
Write(score);
```

Если базовый тип CHAR, то значения из интервала могут обрабатываться как литеры, поэтому их можно вводить, выводить и использовать в логических выражениях, например:

```
Read(simv);
If simv <= 'D' then  Write (' очень хорошо! ')
                 else  Write (' надо поработать ');
```

Пример.

Словесная постановка задачи:

Определить следующую дату дня недели, если заданы текущая дата и день недели. Текущая дата включает число и номер месяца. При выводе дата отображается в виде числа и названия месяца.

Программа на языке Паскаль:

```
Program PRIMER;
Type
    mounth = (jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec);
    day = 1..31;
    year = 1900..2000;
Var
    d : day;
    m : mounth;
    g : year;
    num : 1..12;
Begin
    Write('==>');
    Readln(d, num, g);   { ввод даты - число, номер месяца, год }
    Case num of
        1 : m:= jan;
        2 : m:= feb;
        3 : m:= mar;
        4 : m:= apr;
        5 : m:= may;
        6 : m:= jun;
        7 : m:= jul;
        8 : m:= aug;
        9 : m:= sep;
        10 : m:= oct;
        11 : m:= nov;
        12: m:=  dec;
    End;
    Case  m of     { контроль на корректность количества дней в месяце}
    jan, mar, may, jul, aug, oct :  If d = 31 then
                                    begin
                                        d := 1;  m := succ(m)
                                    end
                                    else d := d + 1;
    apr, jun, sep, nov       :   If d = 30 then
                                 begin
                                     d := 1; m := succ(m)
                                 end
                                 else d := d + 1;
    dec                      :   If d = 31 then
                                 begin
                                     d := 1; m := jan; g := g + 1
                                 end
                                 else d := d + 1;
    feb                      :   If (( d = 28) and ((g mod 4 <> 0) or (g mod 100 = 0))
                                 and (g mod 400 <> 0)) or (d=29) then
                                 begin
```

```
                    d := 1; m := succ(m);
                end
                else d := d + 1
    end;  { case }
    Write('Следующая дата: ', d : 2); { вывод результата }
    Case m of  { выбор названия месяца по номеру месяца}
        jan:  write(' января ');
        feb:  write(' февраля ');
        mar:  write(' марта ');
        apr:  write(' апреля ');
        may:  write(' мая ');
        jun:  write(' июня ');
        jul:  write(' июля ');
        aug:  write(' августа ');
        sep:  write(' сентября ');
        oct:  write(' октября ');
        nov:  write(' ноября ');
        dec:  write(' декабря ')
    end;  { case }
    Writeln(g : 5,' года ')
End.
```

Пояснения к программе:

В современном календаре каждый год, номер которого делится на 4, является високосным, за исключением тех номеров, которые делятся на 100 и не делятся на 400.

Контрольные вопросы

1. Что такое перечислимый тип?
2. Какие операции допускаются с данными перечислимого типа?
3. Какие стандартные функции можно использовать для данных перечислимого типа?
4. Что такое ограниченный тип?
5. Может ли быть последующее значение больше предыдущего при определении ограниченного типа?
6. Какие операции допускаются над переменными ограниченного типа?

Задание к работе

Выполнить индивидуальное задание.

Методические указания

Перед выполнением индивидуального задания по вариантам 1-7, 10 ознакомьтесь с примером, приведенным выше.

При выполнении индивидуального задания придерживаться следующей технологии:

1) изучить словесную постановку задачи, выделив при этом все виды данных;
2) выбрать метод решения задачи, если это необходимо;
3) разработать программу на языке Паскаль, используя перечислимые и ограниченные типы данных;
4) разработать контрольный тест к программе;
5) отладить программу;
6) представить отчет по работе к защите.

Содержание отчета

1. Титульный лист.
2. Словесная постановка задачи.
3. Графический или текстуальный алгоритм решения задачи.

4. Листинг программы.
5. Контрольный тест и результаты тестирования программы.
6. Ответы на контрольные вопросы по согласованию с преподавателем.

Варианты индивидуальных заданий

1) Дан номер года. Указать число дней в этом году.
2) Даны натуральные числа n,m(n<=m). Определить, сколько из чисел n, n+1,..,m являются номерами високосных годов.
3) Даны натуральные числа a_1,b_1,c_1,a_2,b_2,c_2, которые указывают две даты (число, месяц, год). Вычислить количество дней прошедших между двумя этими датами.
4) Даны натуральные числа a_1,b_1,c_1,a_2,b_2,c_2, которые указывают две даты (число, месяц, год). Вычислить количество полных лет прошедших между двумя этими датами.
5) Даны натуральные числа a,b,c которые обозначают число, месяц и год. Проверить корректность этой даты.
6) Даны натуральные числа a,b,c которые обозначают число, месяц и год. Найти номер этого дня с начала года.
7) Даны натуральные числа a,b,c которые обозначают число, месяц и год. Определить, сколько полных дней осталось до конца года.
8) Известно, что астрологи делят год на 12 периодов и каждый из них ставит в соответствии один из знаков зодиака:

20.1-18.2-Водолей	19.2-20.3-Рыбы	21.3-19.4-Овен
20.4-20.5-Телец	21.5-21.6-Близнецы	22.6-22.7-Рак
23.7-22.8-Лев	23.8-22.9-Дева	23.9-22.10-Весы
23.10-22.11-Скорпион	23.11-21.12-Стрелец	22.9-19.1-Козерог

   Дата дана в виде dd. mm. Определить какой знак зодиака соответствует этой дате.
9) "Угадай число". Один из играющих задумывает число от 1 до 100, другой пытается угадать его за 5 вопросов вида: Верно ли, что задуманное число больше такого то числа? Написать программу, загадывающую число.
10) В некоторой библиотеке последний четверг каждого месяца - санитарный день. По заданному номеру месяца выводить на экран дату санитарного дня.

ЛАБОРАТОРНАЯ РАБОТА N 5

Тема: "Организация итерационных циклических процессов"

Цель работы

Овладение теоретическими основами и практическими навыками программирования итерационных вычислительных процессов циклической структуры, получение дальнейших навыков по отладке и тестированию программы.

Краткие сведения из теории

Большинство вычислительных процессов обладает следующей особенностью: отдельные участки вычислений повторяются многократно, при этом всякий раз используются новые значения исходных данных.

Такие вычислительные процессы называют циклическими, а их повторяющиеся участки - циклами.

Применение циклов в программе позволяет эффективно использовать вычислительную машину, приводит к уменьшению длины программы и сокращению времени на ее составление и отладку.

Циклы бывают арифметическими и структурными. Арифметические циклы связаны, в основном, с изменением одной переменной. Они делятся на два типа: циклы на достижение заданной точности (итерационные) и циклы с известным числом повторений.

Циклический процесс называется итерационным, если заранее неизвестно количество повторений цикла, а конец вычисления определяется при достижении некоторой величиной заранее заданной точности вычисления.

Примерами итерационных циклов являются вычисления:

- корней алгебраических и трансцендентных уравнений численными методами;
- корней линейных алгебраических и нелинейных уравнений;
- вычисление квадратного корня по итерационной формуле и т.д.

При программировании итерационных процессов принято их разделять на циклы с "предусловием" и с "постусловием". Их отличие заключается в том, что проверка достижения некоторой величиной заданной точности вычисления осуществляется соответственно либо в начале цикла, либо в конце цикла. Особенность цикла с "постусловием" заключается в том, что повторяющийся участок алгоритма выполнится хотя бы один раз, в то время как в цикле с "предусловием" этот участок может не выполниться ни разу.

Для графического изображения итерационных процессов используется блок "условие". Функциональная схема итерационных процессов с "постусловием" приведена на рис.1,а) и с "предусловием" - на рис.1,б).

Инициализация представляет собой задание начальных значений переменным, которые используются в теле цикла.

Тело цикла - это последовательность действий, которые выполняется многократно.

Для записи итерационных процессов на языке ПАСКАЛЬ имеется два вида операторов цикла:

1) оператор с предварительным условием (предусловием);
2) оператор цикла с последующим условием (постусловием).

Операторы для записи циклов являются сложными, так как в их состав входят другие операторы.

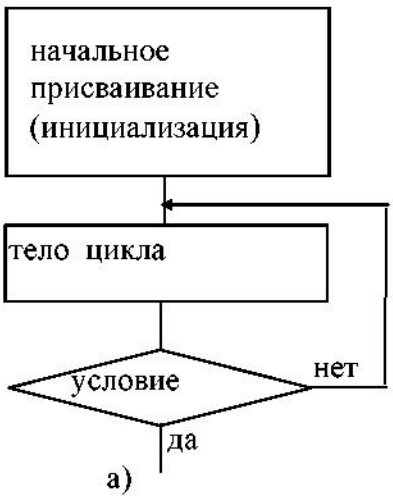
Для всех операторов цикла характерны следующие особенности:

повторяющиеся вычисления записываются лишь один раз;

вход в цикл возможен только через его начало;

переменные оператора цикла должны быть определены до входа в циклическую часть;

необходимо предусмотреть выход из цикла: или по естественному его окончанию, или по оператору перехода. Если этого не предусмотреть, то циклические вычисления будут повторяться бесконечно и это приведет к "зацикливанию" программы.



а)

б)

Рис.1. Функциональная схема итерационных циклов.

5.1. Оператор цикла с предварительным условием

Синтаксис оператора цикла с "предусловием":

```
WHILE <логическое выражение> DO
BEGIN
   <предложения тела цикла>
END
```

Здесь WHILE (пока) и DO (выполнить) - служебные слова.

Оператор цикла действует следующим образом. Предварительно проверяется значение логического выражения. Пока оно истинно, выполняются предложения циклической части. Как только оно становится ложным, происходит выход из цикла. Если с самого начала значение логического выражения ложно, то предложения циклической части не выполнятся ни разу.

Обратите внимание на то, что предложения циклической части, заключеннные в операторные скобки BEGIN - END, представляют собой составной оператор.

Возможен случай, когда в циклической части стоит оператор перехода, передающий управление за пределы цикла. В такой ситуации цикл может завершиться до его естественного окончания (т.е. при истинном значении логического выражения).

Если в циклической части стоит всего одно предложение, то операторные скобки BEGIN - END можно не использовать и оператор цикла принимает вид:

```
WHILE <логическое выражение> DO <предложение>; .
```

Пример.

Словесная постановка задачи.

Вычислить приближенное значение суммы S=1/ i с точностью до Eps. При этом i может принимать целочисленные значения с заданным шагом.

Дано: Eps - точность приближения;
Step - шаг изменения переменной i;
iStart - начальное значение переменной i.

Результат: S - приближенное значение суммы.

Промежуточные данные: a - текущее значение слагаемого 1/ i.

Текстуальное описание алгоритма.

1) ввод значений переменных iStart, Step, Eps;
2) S = 0;

3) i = iStart;
4) a = 1/iStart;
5) если a < Eps, перейти на п.10;
6) S = S + a;
7) i = i + Step;
8) a = 1/ i;
9) перейти на п.5;
10) вывод результата.

Схема алгоритма.

Графическая схема алгоритма приведена на рисунке 2.

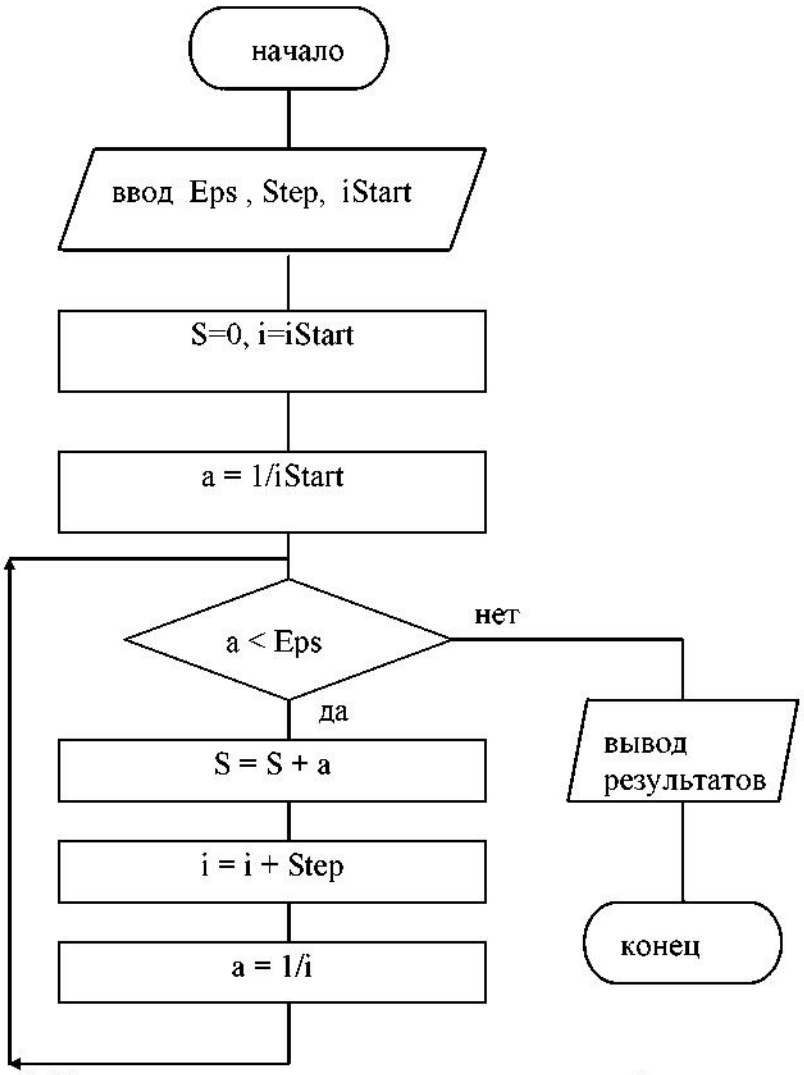


Рис.2. Принципиальная схема циклического алгоритма с "предусловием".

Запись алгоритма на языке Паскаль (операторная запись алгоритма).

```
Program ExampCircle1;
  Var
   i, iStart, Step : integer;
   S, a, Eps       : real;
  Begin
   { Ввод исходных данных }
   Writeln('Введите значения переменных iStart, Step, Eps:');
   Readln(iStart, Step, Eps);
   { инициализация }
```

```
S:= 0;
i:= iStart;
a:= 1/iStart;
{ цикл }
While a >= Eps do
begin
  S:= S + a;
  i:= i + Step;
  a := 1/i
end;
Writeln('Значение суммы ряда 1/i = ', S:10:4) { вывод результатов }
End.
```

Примечание: так как циклическая часть программы выполняется по условию a >= Eps (см. текстуальное описание алгоритма), то в операторе цикла с "предусловием" используется именно это условие.

Составление контрольного теста.

При начальных значениях переменных:

iStart = 2, Step = 2, Eps = 0.01

получаем следующий ряд

S = 0 + 0.5 + 0.25 + 0.1999 + 0.125 + 0.01 = 1.0849

Пояснение к работе программы:

Пока условие a >= Eps является истинным (т.е. значение a оказывается больше или равно Eps), выполняются следующие предложения циклической части:

- значение S увеличивается на a и результат вновь присваивается переменной S;
- значение i увеличивается на величину Step и результат снова присваивается переменной i;
- вычисляется новое значение переменной a.

Начальные значения переменных заданы до начала оператора цикла.

Переменные, a также логическое выражение принимают следующие значения в процессе выполнения этой части программы при заданном контрольном тесте: iStart = 2, Step = 2, Eps = 0.01

	1-й шаг	2-й шаг	3-й шаг	4-й шаг	5-й шаг	6-й шаг
i	2	4	6	8	10	12
a	0.5	0.25	0.1999	0.125	0.01	0.0083
S	0.5	0.75	0.9499	1.0749	1.0849	-
a >= Eps	0.5>=0.01	0.25>=0.01	0.1999>=0.01	0.125>=0.01	0.01>=0.01	0.0083>=0.01
Значение условия	Истинно	Истинно	Истинно	Истинно	Истинно	Ложно

Таким образом, предложения, составляющие тело цикла, выполнились пять раз. На шестом шаге вычислений, при i=12 и a=0.0083, логическое выражение становится ложным, и управление передается за пределы цикла, т.е. предложению Writeln, следующему за первой операторной скобкой (end).

5.2. Оператор цикла с последующим условием

Синтаксис оператора цикла с "постусловием" :

REPEAT < предложения тела цикла >

UNTIL < логическое выражение > ; .

Здесь REPEAT (повторить) и UNTIL (до тех пор) - служебные слова.

Оператор с "постусловием" работает следующим образом. Предложения циклической части выполняются повторно (по крайней мере один раз) до тех пор, пока значение логического выражения ложно. Условием прекращения циклических вычислений является истинное значение логического выражения. Итак, сначала выполняется циклическая часть, а затем проверяется условие. Обратите внимание, что эти действия прямо противоположны действиям оператора цикла с "предусловием", где сначала проверяется условие, а затем выполняются предложения циклической части.

Следует подчеркнуть, что нижняя граница предложений циклической части четко обозначена словом UNTIL, поэтому нет необходимости заключать предложение циклической части в скобки вида BEGIN-END. В то же время наличие операторных скобок не является ошибкой.

Если в циклической части встречается оператор перехода, указывающий на метку за пределами цикла, то цикл может завершиться до его естественного окончания.

Рассмотрим использование данного оператора на предыдущем примере. Основные изменения претерпевают только схема алгоритма и программа, поэтому рассмотрим только их.

Схема алгоритма

Графическая схема алгоритма приведена на рисунке 3.

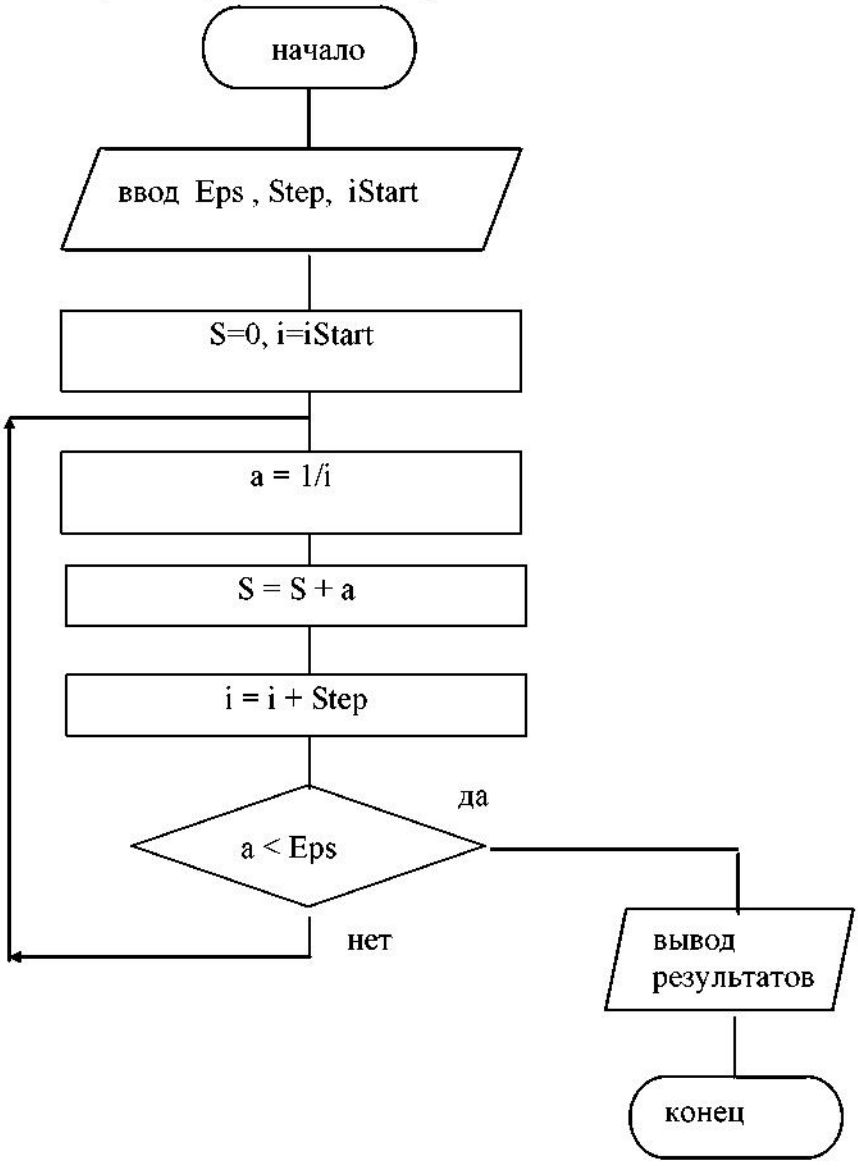


Рис.3. Принципиальная схема циклического алгоритма с "постусловием".

Запись алгоритма на языке Паскаль (операторная запись алгоритма).

```
Program ExampCircle2;
  var
    i, iStart, Step : integer;
    S, a, Eps       : real;
  Begin
    { Ввод исходных данных }
    Writeln('Введите значения переменных iStart, Step, Eps:');
    Readln(iStart, Step, Eps);
    { инициализация }
    S:= 0;
    i:= iStart;
    { цикл }
    repeat
      a := 1/i;
      S:= S + a;
      i:= i + Step;
    until a < Eps;
    { вывод результатов }
    Writeln('Значение суммы ряда 1/i = ', S:10:4);
  End.
```

Составление контрольного теста

При начальных значениях переменных:

iStart = 2, Step = 2, Eps = 0.01,

получаем следующий ряд

S = 0 + 0.5 + 0.25 + 0.1999 + 0.125 + 0.01 = 1.0849

Пояснение к работе программы

Циклическая часть программы повторяется до тех пор, пока выражение a<Eps не станет истинным. В предыдущем варианте программы, при использовании оператора с "предусловием", повторение выполнялось до тех пор, пока условие a >= Eps было истинным.

Переменные, а также логическое выражение принимают следующие значения в процессе выполнения этой части программы при том же контрольном тесте: iStart = 2, Step = 2, Eps = 0.01

	1-й шаг	2-й шаг	3-й шаг	4-й шаг	5-й шаг	6-й шаг
i	2	4	6	8	10	12
a	0.5	0.25	0.1999	0.125	0.01	0.0083
S	0.5	0.75	0.9499	1.0749	1.0849	-
a < Eps	0.5<0.01	0.25<0.01	0.1999<0.01	0.125<0.01	0.01<0.01	0.0083<0.01
Значение условия	Ложно	Ложно	Ложно	Ложно	Ложно	Истинно

Предложения, составляющие тело цикла, выполнились снова пять раз. На шестом шаге вычислений, при i=12 и a=0.0083, логическое выражение становится истинным, и управление передается за пределы цикла, т.е. предложению WriteLn, следующему за первой операторной скобкой (end).

Примечание: Как видно из примеров, использование оператора с "постусловием" сократило программу на одно предложение и пару операторных скобок.

Действительно, предложение REPEAT достаточно удобно использовать в программах, но следует помнить, что при использовании предложение WHILE перед выполнением

операторов тела цикла обязательно проверяется заданное условие, во втором случае предложения тела цикла обязательно выполнятся хотя бы один раз.

Контрольные вопросы

1. Дайте определение арифметических итерационных циклов.
2. Перечислите особенности циклических процессов с предварительной проверкой условия.
3. Перечислите особенности циклических процессов с последующей проверкой условия.
4. В чем состоит отличие операторов цикла с "предусловием" и с "постусловием"?
5. Будет ли выполняться циклическая часть программы, если логическое выражение является ложным с самого начала в предложении WHILE?
6. Будет ли выполняться циклическая часть программы, если логическое выражение истинно с самого начала в предложении REPEAT?

Задание к работе

1. Выполнить индивидуальное задание А.
2. Выполнить индивидуальное задание Б.

Методические указания

При выполнении индивидуального задания необходимо соблюдать рассмотренную технологию решения задач на ЭВМ:

1) изучить словесную постановку задачи, выделив при этом все виды данных;
2) сформулировать математическую постановку задачи;
3) выбрать метод решения задачи, если это необходимо;
4) разработать графическую схему алгоритма;
5) записать разработанный алгоритм на языке Паскаль;
6) разработать контрольный тест к программе;
7) отладить программу;
8) представить отчет по работе к защите.

Содержание отчета

1. Титульный лист.
2. Словесная постановка задачи.
3. Математическая постановка задачи.
4. Графическая схема алгоритма решения задачи.
5. Листинг программы.
6. Контрольный тест.
7. Результаты тестирования программы.
8. Ответы на контрольные вопросы по согласованию с преподавателем.

Варианты индивидуальных заданий

Задание А

Методом итераций вычислить корень уравнения вида f(x)=0, расположенный в интервале [A, B], с абсолютной погрешностью в соответствии с вариантом задания. Определить также число итераций, необходимое для нахождения корня.

вариант задания	уравнение	отрезок	точность
1	$e^x - e^{-x} - 2 = 0$	[0;1]	1E-4
2	$3\sin\sqrt{x} + 0{,}35x - 3{,}8 = 0$	[2;3]	1E-4
3	$x - 2 + \sin(1/x) = 0$	[1.2;2]	1E-4
4	$1 - x + \sin x - \ln(1+x) = 0$	[0;1,5]	1E-4
5	$x^2 - \ln(1+x) - 3 = 0$	[2;3]	1E-4

6	$1/(x - 3+\sin 3,6x) = 0$	[0;0,85]	0.5E-4
7	$\ln x - x + 1,8 = 0$	[2;3]	0.5E-4
8	$0,1x^2 - x \ln x = 0$	[1;2]	0.5E-4
9	$x + \cos(x^{0,52} + 2) = 0$	[0,5;1]	1E-4
10	$\sqrt{1 - 0.4x} - \arcsin x = 0$	[0;1]	1E-4

Задание Б

Вычислить значение суммы членов бесконечного ряда с заданной точностью Eps. На печать вывести значение суммы и число членов ряда, вошедших в сумму.

Вариант	Сумма членов ряда	Значение	Точность вычисления
1	$S=-\frac{2x^2}{2}+\frac{((2x)^2)^2}{24}+\ldots.+(-1)^n\frac{(2x)^{2n}}{(2n)!}$	0.20	10^{-5}
2	$S=x-\frac{x^3}{3}+\ldots..(-1)^n\frac{x^{2n-1}}{2n-1}$	0.10	$0.5*10^{-4}$
3	$S=\frac{x^3}{5}-\frac{x^5}{17}+\ldots..+(-1)^{n+1}\frac{x^{2n+1}}{4n^2+1}$	0.15	10^{-3}
4	$S=1+\mathrm{Cos}(\pi/4)\frac{x}{1!}+\ldots..+\frac{\mathrm{Cos}(n*\pi/4)}{n!}x^n$	0.12	10^{-4}
5	$S=1+\frac{x^2}{2!}+\ldots..+\frac{x^{2n}}{(2n)!}$	0.70	10^{-4}
6	$S=4\cdot(1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\ldots.+(-1)^n\frac{1}{2n-1}$	-	10^{-4}
7	$S=\frac{1}{x}-\frac{1}{3x^3}+\frac{1}{5x^5}+\ldots..+(-1)^n\frac{1}{(2n+1)x^{2n+1}}$	1.5	$0.5*10^{-3}$
8	$S=x+\frac{x^3}{3!}+\frac{x^5}{5!}+\ldots..+\frac{x^{2n+1}}{(2n+1)!}$	1.7	10^{-3}
9	$S=1+\frac{x^2}{2!}-\frac{3x^4}{4!}+\ldots..+(-1)^n\frac{2n-1}{(2n)!}x^{2n}$	0.75	$0.5*10^{-3}$
10	$S=1+\frac{\mathrm{Cos}x}{1!}+\frac{\mathrm{Cos}2x}{2!}+\ldots.+\frac{\mathrm{Cos}nx}{n!}$	0.3	10^{-4}

ЛАБОРАТОРНАЯ РАБОТА N 6

Тема: "Организация арифметических циклических процессов с известным числом повторений"

Цель работы

Овладение теоретическими основами и практическими навыками программирования арифметических циклических процессов с известным числом повторений, получение дапльнейших навыков по отладке и тестированию программы.

Краткие сведения из теории

Циклы с фиксированным числом повторений изображаются в графических схемах алгоритма с помощью специального блока, называемого блоком модификации:

Здесь в записи блока модификации переменная I называется параметром цикла, значения N, K задают соответственно начальное и конечное значение параметра цикла.

Блок модификации задает следующую последовательность действий:

1. Параметру цикла I присваивается начальное значение N.
2. Проверяется условие I > K. Если условие выполняется, то происходит выход из цикла, в противном случае осуществляется переход на следующий пункт (п.3).
3. Выполняются предложения, составляющие тело цикла.
4. К параметру цикла добавляется значение шага (на языке Паскаль значение шага может быть равно +1 или -1), то есть вычисляется арифметическое выражение I = I + 1.
5. Выполняется переход к пункту 2.

Функциональная схема арифметического цикла с известным числом повторений приведена на рисунке 4.

При программировании цикла с фиксированным числом повторений на языке Паскаль используется оператор цикла с параметром.

Рис.4. Функциональная схема арифметического цикла с известным числом повторений.

6.1. Оператор цикла с параметром

Синтаксис оператора цикла с параметром :

```
FOR i:= N TO K DO
BEGIN
```

< предложения тела цикла >
END
Здесь FOR (для), TO (до), DO (выполнить) - ключевое слова; i - параметр цикла; N, K - начальное и конечное значение параметра цикла.
Циклическая часть программы выполняется повторно для каждого значения параметра цикла i от его начального значения N до конечного значения K включительно.
В качестве параметра цикла может использоваться только переменная, N и K могут быть заданы выражениями. Если в выражении присутствуют действительные переменные, то необходимо к выражению применить стандартные математические функции Round или Trunc.
Параметр цикла i описывается в разделе переменных как переменная целого типа. Параметр i изменяется на величину шага, равную +1 или -1. Если значение параметра цикла уменьшается, то шаг его изменения равен -1, и в операторе цикла ***FOR*** ключевое слова ***TO*** заменяется на ключевое слово ***DOWNTO***.
Рассмотрим использование оператора цикла с параметром.
Пример1.
Фрагмент программы:

```
For i := 1 to 5 do
Begin
   A := 2 * i;
   B := 2 * i + 1;
   WriteLn(A:3, B:3)
End.
```

В этом фрагменте начальное и конечное значения параметра i равны 1 и 5 (N = 1, K = 5). Тело цикла повторяется пять раз, при этом параметр цикла i принимает значения от 1 до 5. В результате выполнения программы переменные получают следующие значения:

```
I . . . 1  2  3  4   5  - пошаговые значения параметра;
A . . . 2  4  6  8  10  - пошаговые значения результатов (A, B).
B . . . 3  5  7  9  11
```

Фрагмент программы с убыванием значений параметра цикла от 5 до 1 (N = 5 и K = 1) имеет следующий вид:

```
For i := 5 downto 1 do
Begin
   A := 2 * i;
   B := 2 * i + 1;
   WriteLn(A:3, B:3)
End.
```

В процессе выполнения программы переменные принимают следующие значения:

```
I  . . . 5  4  3  2  1  - пошаговые значения параметра;
A  . . . 10 8  6  4  2  - пошаговые значения результатов (A, B).
B  . . . 11 9  7  5  3
```

Если циклическая часть программы содержит только одно предложение, то операторные скобки BEGIN - END можно не указывать.
В этом случае цикл с параметром записывают в следующем виде:
FOR i := N to K DO < оператор >;
FOR i := K DOWNTO N DO < оператор >;
Параметр цикла i не должен переопределяться внутри циклической части.
Если шаг изменения параметра цикла равен +1 и N > K, то циклическая часть не выполнится ни разу.
После естественного завершения цикла значение параметра цикла не определено. Это означает, что при последнем выполнении циклической части значение i = K, а после ухода за пределы цикла значение i теряется.

Примечание. Оператор цикла FOR задает закон изменения только одного целочисленного параметра. Для задания закона изменения других параметров нужно использовать операторы присваивания, одни из которых перед циклом задают их значения, а другие внутри цикла вычисляют их текущие значения. Если в цикле изменяется один параметр, который не является целочисленным, то он не может быть параметром цикла. В этом случае необходимо вводить целочисленный параметр, который будет счетчиком количества повторений.

Рассмотрим ***пример***.

Словесная постановка задачи.

Вычислить значения функции $Y = c^2/(c + X)$ при X, изменяющемся от a до b с шагом h.

Исходные данные: a, b, h, c.

Результат: последовательность значений Y, вычисленных на каждом шаге.

Промежуточные данные: X.

Математическая постановка задачи. Количество повторений цикла будет равно

$$K = \left[\frac{b - a}{h}\right] + i \qquad (1)$$

Таким образом, параметр цикла i изменяется от N до K, причем N = 1, а K вычисляется по формуле (1).

Значение переменной X внутри тела цикла можно вычислять по формуле:

$$X = X + h.$$

Графическая схема алгоритма.

Она представлена на рисунке 5.

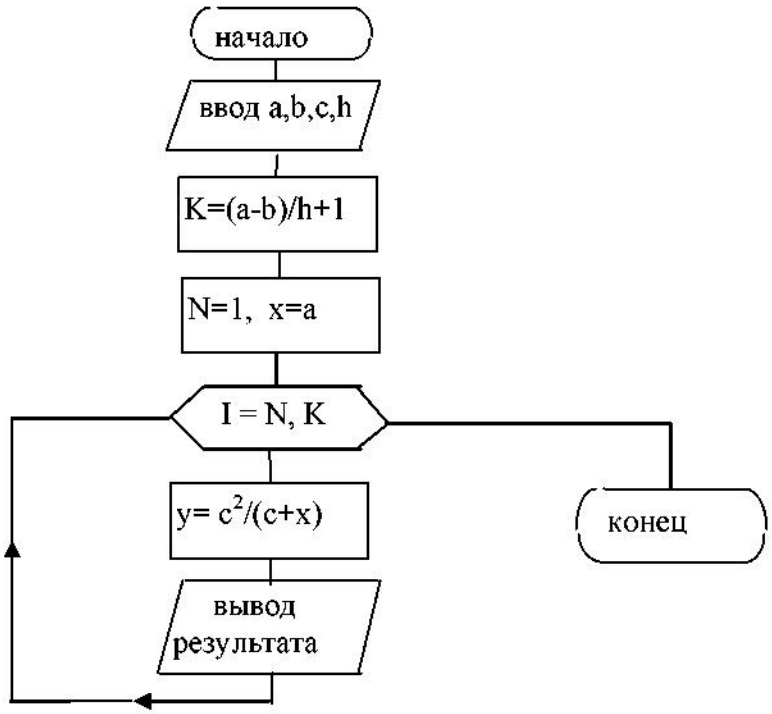


Рис. 5. Принципиальная схема циклического алгоритма с фиксированным числом повторений.

Запись алгоритма на языке Паскаль (операторная запись алгоритма).

```
Program ExampCircle3(Input, Output);
Var
   a, b, c, h, X, Y : real;
   i, N, K          : integer;
Begin
   { Ввод данных }
   WriteLn('Введите значения переменных a, b, c, h:');
   ReadLn(a, b, c, h);
   { инициализация переменных }
   K := trunc((a - b)/h + 1);
```

```
  X := a;
  N := 1;
  { вычисление функции в цикле }
  For i := N to K do
  begin
   Y := c * c /(c + X);
   WriteLn(i, ' Значение функции = ',Y:10:4);
   X:= X + h;
  end;
End.
```

Контрольный тест к программе.

Пусть X изменяется в пределах от 0 до 3 с шагом 0.1, значение константы c = 2, тогда a = 0, b = 2, h = 0.5, N = 1, K = 5.

I	1	2	3	4	5
X	0	0.5	1	1.5	2
Y	4/(2+0)=2	4/(2+0.5)=1.6	4/(2+1)≈1.3	4/(2+1.5)≈1.14	4/(2+2)=1

6.2. Вложенные циклы

Циклы могут быть вложены один в другой. Иногда их называют сложными циклами. При использовании вложенных циклов необходимо составлять программу таким образом, чтобы внутренний цикл полностью входил в тело внешнего цикла. Внутренний цикл может, в свою очередь, иметь другой внутренний цикл (циклы). Структуру вложенных циклов рассмотрим на примере.

Пример. Вычислить значение функции Y = 2L + M при всех значениях переменных M=1, 2, 3 и L=2, 4, 6, 8.

Программа должна содержать два вложенных друг в друга цикла. В качестве параметра внешнего цикла удобно использовать переменнуюM, а внутреннего - переменную L. Тогда при первом значении M переменная L будет принимать значения 2, 4, 6, 8. При следующем значении M переменная L также будет принимать значения 2, 4, 6, 8 и так до конца.

Программа имеет вид:

```
Program Circles;
Var
   M, L, Y : integer;
Begin
   For M := 1 to 3 do
   Begin  { начало тела внешнего цикла }
      L := 2;
      While L <= 8 do
         Begin  { начало тела внутреннего цикла }
            Y := 2 * L + M;
            Writeln('M = ', M:4, ' L = ', L:4, ' Y = ', Y:4);
             L := L + 2
         End   { конец тела внутренного цикла }
    End    { конец тела внешнего цикла }
 End.
```

Здесь внешний цикл организован с использованием предложения For, а внутренний - с использованием предложения While. В процессе выполнения вложенных циклов переменные получат следующие значения:

1-й 2-й 3-й 4-й 9-й 10-й 11-й 12-й

		шаг	шаг	шаг	шаг	. . .				шаг	шаг	шаг	шаг
M	. . .	1	1	1	1	2	2	2	2	3	3	3	3
L	. . .	2	4	6	8	2	4	6	8	2	4	6	8
Y	. . .	5	9	13	17	6	10	14	18	7	11	15	19

Контрольные вопросы

1. Дайте определение арифметического цикла с известным числом повторений.
2. Какой тип данного может иметь параметр цикла?
3. Что такое N, K? Какой тип данного они должны иметь?
4. Может ли значение K превышать значение N, если да, то в каком случае?
5. Обязательно ли реализовывать арифметический цикл с известным числом повторений в программе с помощью предложения FOR?
6. В каких случаях необходимо использовать вложенные циклы?
7. Обязательно ли использование однотипных операторов цикла при организации вложенных циклов?
8. Какова максимальная глубина вложенности циклов?
9. Можно ли изменять программным путем параметр цикла циклической части предложения For?

Задание к работе

Выполнить индивидуальное задание.

Методические указания

При выполнении индивидуального задания необходимо придерживаться технологии решения задач на ЭВМ:

1) изучить словесную постановку задачи, выделив при этом все виды данных;
2) сформулировать математическую постановку задачи;
3) выбрать метод решения задачи, если это необходимо;
4) разработать графическую схему алгоритма;
5) записать разработанный алгоритм на языке Паскаль;
6) разработать контрольный тест к программе;
7) отладить программу;
8) представить отчет по работе к защите.

Содержание отчета

1. Титульный лист.
2. Словесная постановка задачи.
3. Математическая постановка задачи.
4. Графическая схема алгоритма решения задачи.
5. Листинг программы.
6. Контрольный тест.
7. Результаты тестирования программы.
8. Ответы на контрольные вопросы по согласованию с преподавателем.

Варианты индивидуальных заданий

1. Кинетическая энергия движущегося тела $W = mV^2/2$, где m - масса тела, V - его скорость. Получить зависимость W от m при значениях V, изменяющихся от V1 до V2 с шагом h1. Масса m изменяется от m1 до m2 с шагом h2.

2. При сооружении железобетонного фундамента площадь сечения арматуры определяется по формулам:

$F = (N - RS)/(D - R)$, если $F > 0.03S$;

$F = (N - RS)/D$, если $F \le 0.03S$,

где N - расчетная сила, приложенная по оси элемента;

S - площадь сечения бетона;

R - сопротивление бетона осевому сжатию;

D - сопротивление сжатой арматуры.

Рассчитать значения переменной N, если известны R, S и D, а F изменяется от F1 до F2 с шагом h.

3. Вычислить значения функции Y = 10 SinDX / (1+D^2 X^2), если X изменяется от 0.1 до 10 с шагом 0.13, а D - от 1.2 до 5.4 с шагом 1.1.

4. Обьем цилиндрической подковы вычисляется по формуле:

$$V = \frac{n}{3b}\left[a(3r^2 - a^2) + 3r^2(b - r)\frac{f * \pi}{180^\circ}\right].$$

Вычислить значения обьема V, который зависит от угла f, если a, b, h и r известны, а f изменяется от f1 до f2 с шагом h.

5. Температура T молока в бидонах после содержания их на открытом воздухе и перевозки в крытом брезентом автомобиле в течение W часов выражается формулой:

$$T = T0 + (T1 - T0)\exp\left(-\frac{KSW}{V}\right),$$

где T1 - начальная температура молока, °C;

T0 - температура окружающего воздуха, °C;

S - площадь поверхности бидона, $м^2$;

V - обьем бидона, м ;

K = 0.00448 м/г - постоянный коэффициент.

Вычислить значения температуры молока после хранения и перевозки с выдачей результатов для W, изменяющегося от 1 до 10 часов с интервалом 1 час.

6. Производительность станка для резки бетонно-мозаичных плит составляет A $см^2$/ч. После того, как на станке нарезали плиты общей площадью B $см^2$, станок останавливают на T1 мин., после чего вновь нарезают B $см^2$ плит с последующим перерывом на T1 минут и т.д.

Вычислить значение площади нарезанных плит S $см^2$ за общее время работы T2 часов.

7. Вычислить значения функции Z = X * b/(X +b), если X изменяется от начального значения a с шагом h.

8. Вычислить значения функции $Z = X\sqrt{cX}$, где X изменяется от a до b с шагом h. Известно, что c > 0.

9. Вычислить значения функции Y = nSinX - CosnX, если X изменяется от X0 до XK с шагом h.

10. Вычислить значения функции Y = SinX, если X изменяется от X0 до XK с шагом h.

ЛАБОРАТОРНАЯ РАБОТА N 7

Тема: "Структурные циклические процессы"

Цель работы

Овладение практическими навыками работы с векторами и матрицами при программировании.

Краткие сведения из теории

При организации структурных циклов в вычислительном процессе обязательным элементом являются массивы данных.

7.1. Регулярные типы данных или массивы

Массив - это упорядоченная последовательность данных, обозначаемая одним именем (идентификатором). Члены этой последовательности называются элементами массива. Каждый элемент определяется именем массива и его положением в массиве. Положение элемента в массиве определяется его индексом (порядковым номером). Упорядоченность последовательности данных заключается в том, что элементы массива располагаются в последовательных ячейках памяти.

Массивы бывают одномерные, двумерные, трехмерные, четырехмерные и т.д. Понятие массива соответствует таким математическим понятиям как вектор и матрица. Одномерный массив соответствует понятию вектора; двумерные, трехмерные и т.д. массивы соответствуют понятию матрицы. Размерность ("мерность") массива определяет количество индексов отдельного элемента.

На языке Паскаль массивы описываются в разделе переменных следующим образом:

VAR

A : array[1..k] of < тип элементов массива > ; - описание одномерного массива (вектора) A. Переменная k задает количество элементов в массиве, при этом значение индекса элементов лежит в интервале от 1 до k и может принимать только целые значения. Все элементы массива имеют один и тот же тип. При этом тип элементов массива может быть как базовым, так и сложным (типом, обьявленным в разделе TYPE).

B : array[1..k1, 1..k2] of < тип элементов массива >; - описание двумерного массива (двумерной матрицы). Здесь k1 задает количество строк и диапазон изменения индекса строк, k2 - количество столбцов и диапазон изменения индекса столбцов.

C : array[1..k1, 1..k2, 1..k3] of < тип элементов массива> ; - описание трехмерного массива (трехмерной матрицы). Здесь k1 задает количество строк и диапазон изменения индекса строк, k2 - количество столбцов и диапазон изменения индекса столбцов, k3 - количество слоев и диапазон изменения индекса слоев.

Подобным образом описываются массивы и более высокой размерности. Ограничений на максимальное значение размерности не существует. Но программист должен помнить, что обьем массива в байтах не должен превышать 64К в оперативной памяти. Упорядочение элементов многомерных массивов выполняется от внутреннего индекса к внешнему, так, например, элементы двумерной матрицы упорядочены сначала по строкам, а затем по столбцам.

Обращение к отдельным элементам массива осуществляется по имени массива и последовательности его индексов, заключенных в квадратные скобки и разделенных запятой.

Например:

*A[1], A[n], A[2*k+1]* - обращение к элементам вектора. Индекс может быть задан константой, переменной и арифметическим выражением, причем, индекс должен быть целого типа;

B[1,5], B[i,j], B[2, trunc(sqrt(X))] - обращение к элементам двумерной матрицы.

Пример.

Задана двумерная матрица B размерностью 3x2:

$$\begin{vmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{vmatrix}$$

Ее описание может быть следующим:
VAR B : ARRAY [1..3,1..2] OF INTEGER;
Так как тип элементов INTEGER и длина элемента этого типа равна четырем байтам, а количество элементов в массиве 3 * 2 = 6, то обьем оперативной памяти, занимаемый массивов, составляет 24 байта. Элементы двумерного массива B расположены в памяти следующим образом: b_{11}, b_{12}, b_{13}, b_{21} , b_{22} , b_{23} - упорядочение выполнено по строкам, а затем по столбцам.
Обращаясь к элементу второй строки третьего столбца, достаточно записать B[2,3].

7.2. Организация структурных циклических процессов

При организации таких процессов в качестве параметра цикла практически всегда удобно использовать индекс массива. При работе с векторами организуются простые циклы. Если размерность массива превышает единицу, то в этом случае организуются вложенные циклы. При программировании структурных циклов удобно использовать оператор цикла с параметром - FOR.

Пример.
Составить программу для вычисления суммы положительных элементов каждой строки матрицы A(10x8).
Для решения этой задачи небезразлично, какой цикл будет внешним. Внешний цикл определяет изменение индексов строк, а внутренний - номера элементов по строке, то есть индекс столбцов. Перед внутренним циклом необходимо задать начальное значение суммы SUM = 0. Так как значения индексов строк и столбцов определены, можно воспользоваться оператором цикла FOR.
Программа имеет вид:

```
Program Sunstr;
Const
    nm = 10;
    mm = 8;
Var
     sum       : real;
     n, m, i, j : integer;
     a         : array [1..nm, 1..mm] of real;
     astr      : array [1..nm]      of real;
Begin
     { Ввод данных }
     WriteLn('Введите количество строк и столбцов матрицы:');
     Readln(n, m);
     WriteLn('Введите построчно значения элементов матрицы.');
     For i:= 1 to n do
     begin
      WriteLn('Строка ', i:2:0, ' :');
      For j:= 1 to m do
        Read(a[i,j]);
      WriteLn;
     end;
     { Выполнение суммирования }
     For i:= 1 to n do { внешний цикл по индексу строк }
     Begin   { начало тела внешнего цикла }
        sum:=0;
        For j:= 1 to m do { внутренний цикл по индексу столбцов}
```

```
        If a[i,j] > 0 then sum:=sum+a[i,j]; { оператор тела
                                  внутреннего цикла}
        astr[i]:=sum { вычисление текущего элемента вектора }
    End;   { конец тела внешнего цикла }
    { Вывод результата }
    WriteLn('Вектор положительных элементов строк матрицы:');
    For i:=1 to n do Write ('astr[',i:2:0,'] = ',astr[i]);
End.
```

7.3. Приемы программирования с использованием операторов цикла

В практике программирования имеется банк типовых алгоритмов. Рассмотрим их реализацию на следующих примерах:

1. Организация циклов с несколькими одновременно меняющимися параметрами.

Пример.

Составить программу для вычисления функции

$$Z = (X + Y(i)) / (X - Y(i)),$$

если X изменяется одновременно с Y(i) от начального значения A с шагом h. Значения Y(i) являются элементами массива Y[1..20].

Программа имеет вид:

```
Program Func;
Const
   N = 20;
Var
   X, h, A, Z : real;
   I       : integer;
   T       : array[1..N] of real;
Begin
   { инициализация }
   WriteLn('Введите величину шага и начальное значение переменной X:');
   Read (h, A);
   X := A;
   { цикл }
   For i := 1 to N do
   begin
      Read(Y[i]);
      Z := (X + Y[i]) / (X - Y[i]);
      Writeln(' Z = ', Z);
      X := X + H
   end
End.
```

2. Табулирование функции.

Пример.

Составить программу для вычисления и запоминания функции

$$Z = \sqrt{(x+1)/i},$$

где X - элемент вектора X[1..50].

Для вычисления и вывода результатов удобно использовать оператор цикла FOR.

Программа имеет вид:

```
Program Tab;
Const
   nn = 50;
Var
    n, i : integer;
```

```
  X, Z : array[1..nn] of real;
Begin
   { инициализация }
   Write('Введите количество элементов в векторе: ');
   ReadLn(n);
   { цикл }
   For i := 1 to n do
     begin
       Read(X[i]);
       Z[i] := Sqrt((X[i] + 1) / i)
     end;
    WriteLn;
    For i := 1 to n do
      Write(Z[i]:10:3)
End.
```

3. Формирование нового массива по заданному условию (проекция).

Пример.

Переписать элементы целочисленного массива M[1..40], кратные пяти, подряд в массив M5. Если такие элементы отсутствуют, то выдать диагностическое сообщение об этом.

Программа имеет вид:

```
Program Project;
Const
    nn=40;
Var
    k, i, n     : integer;
    M, M5   : array [1..nn] of integer;
Begin
    { инициализация }
    Write('Введите количество элементов в векторе: ');
    ReadLn(n);
    k := 0;
    { цикл }
    For i := 1 to n do
     begin
        Read (M[i]);
        If M[i] div 5*5 = M[i] then
          begin
            k := k + 1;
            M5[i] := M[i]
          end
       end;
     If k = 0 then writeln('Элементов, кратных 5, нет.')
       else
        begin
          WriteLn('Элементы нового массива:');
          For i := 1 to k do
            Write (M5[i]:8);
        end
End.
```

4. Вычисление суммы и произведения элементов массивов.

Пример А.

Составить программу для вычисления значения функции

$Z = \sum X_i^2 / i$, где X_i -элемент массива X[1..20].

Программа имеет вид:

```
Program Summa;
Const
    nn = 20;
Var
    n, i : integer;
    Z    : real;
    X    : array [1..nn] of real;
Begin
   { инициализация }
   Write('Введите количество элементов в векторе : ');
   ReadLn(n);
   Z := 0;
   { цикл }
   For i := 1 to n do
     begin
       Read (X[i]);
       Z := Z + Sqr(X[i])/i
     end;
 End.
```

Пример Б.

Составить программу для вычисления произведения Z положительных элементов массива X[1..100].

Программа имеет вид:

```
Program Mult;
   Const
    nn = 100;
   Var
    i, n : integer;
    Z    : real;
    X    : array[1..nn] of real;

   Begin
    { инициализация }
    Write('Введите количество элементов в векторе : ');
    ReadLn(n);
    Z := 1;
    { цикл }
    For i := 1 to n do
     begin
       Read (X[i]);
       If X[i] > 0  then Z := Z * X[i]
     end;
     Writeln('Произведение элементов Z = ', Z)
    End.
```

5. Вычисление суммы членов бесконечного ряда с заданной точностью.

Пример.

Составить программу для вычисления суммы членов ряда

$Z = 1 + \sum_{n=1}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!}$ с точностью до члена ряда, меньшего Eps.

Программа имеет вид

```
Program SumRow;
Var
    Y, Z, X, Eps : real;
    n            : integer;
Begin
   { инициализация }
   Write('Введите значения переменной X и погрешности: ');
   ReadLn(X, Eps);
   Y := 1;
   Z := 1;
   n := 1;
   { цикл }
   Repeat
      Y := Y * (-X * X / ( 2n - 1) * 2 n )));
      Z := Z + Y;
      n:=n+1
   until Y < Eps;
   Write('Сумма ряда = ', Z)
End.
```

6. Нахождение наибольшего и наименьшего значения элементов массива и функции.

Пример А.

Составить программу для нахождения наименьшего значения функции

Y = a * Exp(- b * X) * Sin(omega * X + fi)

при изменении аргумента X в интервале от 0 до c с шагом h.

При организации программы с предложением For необходимо подсчитать число повторений, зависящее от интервала аргумента и его шага.

При использовании предложений While и Repeat введение дополнительного параметра не требуется, так как цикл предусматривает изменение аргумента.

Программа с оператором For имеет вид

```
Program MinF;
Var
    a, b, c, omega, fi, h, Y, Ymin : real;
    k, i                           : integer;
Begin
    { инициализация }
    WriteLn('Введите значения переменных a,b,c,omega,fi,h:');
    ReadLn(a, b, c, omega, fi, h);
    Ymin := 1e19;
    k := trunc(c/h) + 1;
    X := 0;
    { цикл }
    For i:= 1 to k do
     begin
       Y := a * exp(-b * X) * Sin(omega * X + fi);
       If Y < Ymin then Ymin := Y;
       X := X + h
     end;
    Write('Ymin = ', Ymin)
```

End.

В программе с предложением While отсутствует описание переменных K и I, а собственно цикл имеет вид

```
While X <= c do
 Begin
  Y := a * exp(-b * X) * Sin(omega * X + fi);
  If Y < Ymin then Ymin := Y;
  X := X + h
 End.
```

Пример Б.

Составить программу для нахождения наибольшего значения элемента массива X[1..40] и его порядкового номера.

Программа имеет вид

```
Program Max;
Const
    n = 40;
Var
    Xmax      : real;
    i, n, nmax : integer;
    X         : array[1..n] of real;
Begin
    { инициализация }
    Write('Введите количество элементов в векторе : ');
    Read (n);
    { Ввод знчений элементов вектора }
    For i := 1 to n do
     Write(' X[', i:2:0, '] = ');
     Read(X[i]);
     Xmax := X[1];
     nmax := 1;
    { цикл по поиску максимального значения элементов вектора }
    For i:= 2 to n do
     If X[i] > Xmax then
     Begin
       Xmax := X[i];
       nmax := 1;
     End;
    Writeln('Xmax = ', Xmax);
    Writeln('nmax = ', nmax)
End.
```

7. Нахождение корней нелинейного уравнения методом итераций.

Пример.

Методом итераций найти корень уравнения ArcSin(2X+1)-X=0, расположенный на отрезке [-0.5; 0], с абсолютрой погрешностью Eps = 10. Напечатать число итераций, необходимых для вычисления корня.

Заданное уравнение преобразуем к виду X = F(X) следующим образом:

ArcSin(2X + 1) = X;

Sin(ArcSin(2X + 1)) = SinX;

2X + 1 = SinX;

X = 0.5(SinX -1).

Проверка условия сходимости метода итераций: F'(X) = X CosX. Очевидно, что $|F'(x)| = |x \cos x| \le 0.5$ для всех $-0.5 \le X \ge 0$. Следовательно, рассматриваемый процесс итераций сходится.

Программа имеет вид:

```
Program Prim;
Var
    a, b, X1, X0, delta, Eps : real;
    n                        : integer;
Begin
    { инициализация }
    WriteLn('Введите значения переменных a, b, Eps:');
    ReadLn(a, b, Eps);
    X0 := (a + b)/2;
    n  := 0;
    { цикл }
    Repeat
      X1 := 0.5 * Sin(X0 * X0 - 1);
      n  := n + 1;
      delta:= Abs(X1 - X0);
      X0 := X1;
    until delta < Eps;
    WriteLn('Корень=', X1:9:4);
    WriteLn('Число итераций=',n:5)
  End.
```

Контрольные вопросы

Дайте классификацию циклических процессов с указанием их особенностей.
Укажите отличительные особенности каждого из трех операторов цикла.
Дайте определение массива.
Напишите формулу для вычисления обьема памяти, занимаемого массивом.
В оперативной памяти любой многомерный массив располагается линейно. Выведите формулу, по которой процессор определяет порядковый номер элемента многомерного массива в памяти при известных значениях индексов.
Чему равен шаг изменения параметра цикла в предложении For?
В чем отличия итерационных циклов и циклов с фиксированным числом повторений.
В чем состоят преимущества использования операторов цикла?
Указажите основные правила организации вложенных циклов.
Возможен ли выход из внутреннего цикла до его полного завершения ?
Как организовать ввод матрицы размером N*M элементов ?
Как организовать вывод матрицы ?

Задание к работе

1. Выполнить индивидуальное задание А.
2. Выполнить индивидуальное задание Б.

Методические указания

При выполнении индивидуального задания необходимо соблюдать технологию решения задач на ЭВМ:
изучить словесную постановку задачи, выделив при этом все виды данных;
сформулировать математическую постановку задачи, выделив при этом все виды данных;
сформулировать математическую постановку задачи;
выбрать метод решения задачи, если это необходимо;
записать разработанный алгоритм на языке Паскаль;

разработать контрольный тест к программе;
отладить программу;
представить отчет по работе к защите.

Содержание отчета

1. Титульный лист.
2. Словесная постановка задачи.
3. Математическая постановка задачи.
4. Графическая схема алгоритма решения задачи.
5. Листинг программы.
6. Контрольный тест.
7. Результаты тестирования программы.
8. Ответы на контрольные вопросы.

Варианты индивидуальных заданий

Задание А.

Обработать на ЭВМ массив в соответствии с вариантом задания.

Вариант задания	Массив	Действия	Условия и ограничения
1	X(100)	Вычислить сумму и количество элементов массива X.	$0 \le x[i] \le 1$
2	A(80)	Вычислить среднее арифметическое значение элемента массива A	$a[i] > 0$
3	X(70)	Переписать элементы массива X в массив Y и подсчитать их количество.	$-1\ x[i] \le 1$
4	B(50)	Определить максимальный элемент массива B и его порядковый номер.	$x[i] > 0$
5	C(40)	Вычислить минимальный элемент массива C и его номер.	$x[i] < 0$
6	D(80)	Найти максимальный и минимальный элементы массива D и поменять их местами.	
7	Y(20)	Вычислить среднее геометрическое элемента массива Y.	$y[i] > 0$
8	Z(30)	Расположить в массиве R сначала положительные, а затем отрицательные элементы массива Z.	
9	N(50)	Определить сумму элементов массива N, кратных трем.	$n[i]/3*3=n[i]$
10	X(N)	Вычислить сумму и количество элементов массива X.	$N \le 40$

Задание Б.

Вариант задания	Матрица	Действия	Условия и ограничения
1	A(10,15)	Вычислить и запомнить сумму и число положительных элементов каждого столбца матрицы. Результаты отобразить в виде двух строк.	$a[i,j]>0$
2	A(N,M)	Вычислить и запомнить суммы и числа элементов каждой строки матрицы. Результаты отобразить в виде двух столбцов.	$N <= 20$ $M <= 15$
3	B(N,N)	Вычислить сумму и число элементов матрицы, находящихся под главной диагональю и над ней.	$N \le 12$

4	C(N,N)	Вычислить сумму и число положительных элементов матрицы, находящихся над главной диагональю.	c[i,j]>0 N ≤12
5	D(K,K)	Записать на место отрицательных элементов матрицы нули и отобразить ее в общепринятом виде.	K ≤ 10
6	D(10,10)	Записать на место отрицательных элементов матрицы нули, а на место положительных - единицы. Отобразить нижнюю треугольную матрицу в общепринятом виде.	
7	F(N,M)	Найти в каждой строке матрицы максимальный и минимальный эле менты и поместить их на место первого и последнего элемента строки соответственно. Матрицу вывести в общепринятом виде.	N ≤20 M ≤10
8	F(10,8)	Транспонировать матрицу и вывести на печать элементы главной диагонали и диагонали, расположенной под главной.	
9	N(10,10)	Для целочисленной сатрицы найти для каждой строки число элементов, кратных пяти, и наибольший из полученных результатов.	n_{ij} /5*5= n_{ij}
10	P(N,N)	Найти в каждой строке матрицы наибольший элемент и поменять его местами с элементом главной диагонали. Отпечатать полученную матрицу в общепринятом виде.	N ≤15

ЛАБОРАТОРНАЯ РАБОТА N 8

Тема : " Строковые данные "

Цель работы

1. Ознакомиться со строковыми данными.
2. Получить навыки в организации работы со строковыми переменными: удалением, вставкой, копированием, заменой одной строки на другую и т.д.

Краткие сведения из теории

8.1. Объявление строковых переменных

Турбо-Паскаль предоставляет средства для работы с данными строкового типа, которые в дальнейшем будем называть стрингами. Строковый тип данных представляет собой цепочку символов. Длина цепочки может изменяться от 0 до 255. Для определения строкового типа используется служебное слово STRING, за которым в квадратных скобках указывается максимальная длина строки, ***например***:

```
TYPE
        line = string[25];
VAR
        mline : line;
        ...
```

В данном примере переменная mline представляет собой последовательность, включающую до 25 символов, причем каждый символ имеет стандартный тип CHAR.

Значение строковой переменной может быть назначено оператором присваивания, либо введено с устройства ввода, ***например***:

```
aline := 'ВСТИ';
mline := aline;
readln(mline);
```

Изображение строки строится из цепочки символов и заключается в апострофы. Максимальная длина строки может быть задана целым числом, или константой целого типа. Указание максимальной длины может быть опущено; в этом случае подразумевается число 255, ***например***:

```
TYPE
        line  = string;
        line1 = string[255];
```

Описания типов в данном примере эквивалентны. Основное отличие строк от символьных массивов заключается в том, что строки могут динамически изменять свою длину, ***например***:

```
  . . .
mline := 'строка';
mline := mline + 'стала длинной';
  . . .
```

В приведенном примере после первого присваивания длина переменной mline равна шести. Второе присваивание увеличивает ее длину до 19 символов.

Динамические строки организуются в Турбо-Паскале следующем образом: память под стринги отводится по максимуму (согласно описанию), а используется лишь та ее часть, которая реально занята символами строки. При такой организации работы со стрингами Турбо-Паскаль должен знать реальную длину стринга. Поэтому для строковой переменной длиной N символов отводится (N+1) байтов памяти, из которых N байтов предназначены для хранения символов строки, а один байт - для хранения текущей длины строки.

Символы строки нумеруются целыми числами, начиная с единицы, а байт с текущей длиной строки считается нулевым ее элементом. Длину текущей строки можно определить следующим образом:

```
. . .
len := ord(st[0]);
. . .
```

Здесь st - переменная строкового типа.

Если стрингу присваивается значение строкового выражения с длиной, большей чем максимально допустимая, происходит отсечение строки до максимальной длины, ***например:***

```
VAR
      st : string[5];
BEGIN
      st := 'очень длинная строка';
      writeln(st);  { будет отображено только: 'очень'}
       . . .
```

8.2. Основные операции

Для строковых типов данных определена операция "конкатенация", обозначаемая символом '+'. Смысл операции заключается в формировании новой строки. Динамическая длина сформированной строки равна сумме символов строк-операндов, а ее значение равно последовательности символов исходных строк.

Например:

```
VAR
  str1, str2 : string[10];
  st         : string[25];
BEGIN
  str1 := 'Паскаль - ';
  str2 := 'программа';
  st   := str1 + str2;
  WriteLn(st)
END.
```

В результате выполнения программы будет на экране отображена текстовая строка: 'Паскаль - программа'.

Кроме операции конкатенации над значениями строковых типов разрешены операции сравнения < , <=, > , >=, = , <>, IN, при выполнении которых действуют следующие правила:

а) более короткая строка всегда меньше более длинной;

б) если длины сравниваемых строк равны, то происходит поэлементное сравнение символов этих строк с учетом лексикографической упорядоченности значений

в) компаратор IN определяет вхождение левого операнда в правый. Если левый операнд входит в правый, то результат компарации будет истинным (TRUE), в противном случае - ложным (FALSE). Левым операндом может быть только элементарное данное (здесь символ), а правым - любое множество элементов, в данном случае стринг или литерный ряд.

8.3. Доступ к элементам строковых данных

Доступ к отдельным элементам строк осуществляется аналогично доступу к элементам одномерного массива: после имени строковой переменной необходимо в квадратных скобках указать арифметическое выражение целого типа, например:

```
VAR
  mline : string;
  i     : integer;
BEGIN
   . . .
  for i := 1 to length( mline ) do
```

```
if mline[i]  IN  ['a'...'z'] then
   mline[i] := chr( ord( mline[i] ) + 1);
. . .
```

Можно заметить, что работа со строковыми данными аналогична работе с символьными массивами, однако, это не означает их полную идентичность. Так, распространенной ошибкой является работа с элементами строки без учета ее текущей длины. Необходимо помнить, что если длина символьного ряда статична, то длина стринга динамична.

8.4. Стринги и литерные ряды

Регулярную переменную типа ARRAY OF CHAR (или литерный ряд) можно рассматривать как стринг постоянной длины. Данные указанного типа могут быть использованы в любых стринговых выражениях. При этом согласование операндов по типам обеспечивается компилятором, который в подобных случаях просто преобразует литерный ряд в стринг длиной, равной количеству элементов ряда. Это позволяет, например , сравнивать литерные ряды между собой и обращаться с ними точно так же, как с переменными типа STRING.
Допускается выполнять присваивание, в левой части которого стоит имя литерного ряда, в правой - стринговый литерал (константа) длиной, равной количеству элементов ряда. Однако нельзя присваивать какой-либо переменной типа литерный ряд переменную типа STRING или наоборот. ***Например:***

```
Const
  message = 'верно';
Type
  CharArray = array[1..5] of char;
Var
  FixedString, FiveChar : CharArray;
  VarString             : string[10];
Begin
  . . .
  FiveString := 'мерно';
  FixedString := message;
  if FiveChar > FiveString then
    writeln('"', FiveChar,' "больше, чем"', FiveString,'"');
  VarString := 'при';
  VarString := concat(VarString,FiveChar);
  VarString := 'голос';
  FiveChar  := VarString; { так нельзя}
  . . .
```

Пояснение к программе.
В рассмотренном примере объявлено два литерных ряда (типа CharArray) - FixedString и FiveChar, а также стринг VarString.
1. Стринговый литерал 'мерно' присваивается переменной FiveChar.
2. Переменной FixedString присваивается стринговая константа message, имеющая значение 'верно'.
3. Сравниваются два ряда, в результате которого выясняется, что FiveChar больше FixedString (поскольку 'мерно' > 'верно').
4. Переменная VarString получает значение 'при'.
5. В результате конкатенации VarString получает новое значение - 'примерно', которое на следующем шаге затирается значением 'голос'.
6. В последнем операторе делается попытка литерному ряду назначить стринговую переменную, это недопустимо по обычной в таких случаях ошибке "несоответствие типов". В программе левый операнд объявлен как литерный ряд, а правый как стринг.

8.5. Пустой стринг

Стринг, длина которого равна 0, называется пустым. Пустой стринг изображается в виде двух апострофов, записанных рядом, без пробела: ''.
Например:

```
. . .
If st = '' then
 writeln('стринг st пуст');
. . .
```

Следует отметить, что переменную типа STRING необходимо инициировать пустым значением. В противном случае, если к моменту первого использования переменной она не получила какого-то определенного значения, в программе может возникнуть непредсказуемая ситуация. Дело в том, что при включении компьютера оперативная память очищается, в результате любые переменные инициализированы: числовые переменные обнулены, а стринги получают нулевую длину.
В любое другое время работы ЭВМ переменные будут содержать неопределенные значения, обусловленные прежним содержимым памяти. В этом случае текущая длина переменной типа STRING может быть ненулевой.

8.6. Функции преобразования

Для строковых типов данных определены следующие функции преобразования:

a) STR(x[:width[:decimals]];var s:string) - эта функция преобразует численное значение x в его строковое представление s.
b) VAL(s:string; var x code:integer) - эта функция преобразует строковое значение s в его численное представление x.

8.7. Стандартные процедуры и функции

Для строковых типов данных определены следующие процедуры и функции:

a) INSERT(source: string; var s: string; index: integer) - эта процедура предназначена для вставки строки SOURCE в строку S, начиная с символа с номером INDEX в этой строке.
b) DELETE(var s: string; index, count: integer) - эта процедура производит удаление из строки-параметра S подстроки длиной COUNT, начиная с символа с номером INDEX.
c) CONCAT(s1, [s2,...]: string): string - эта функция выполняет слияние строк-параметров, которых может быть произвольное количество. Каждый параметр является выражением строкового типа. Если длина результирующей строки превышает 255 символов, то она усекается до этой длины.
d) COPY(s: string; index: integer; count: integer): string - эта функция возвращает подстроку, выделенную из исходной строки S, длиной COUNT символов, начиная с символа под номером INDEX.
e) POS(substr, s: string): byte - эта функция производит поиск в строке S подстроки SUBSTR. Результатом функции является номер позиции подстроки в исходной строке.
f) LENGTH(s: string): integer - эта функция возвращает текущую длину строки S.
g) MOVE(var x, y; count: word) - эта функция копирует заданное количество COUNT последовательных байт из источника X в приемник Y.
h) FILLCHAR(var x; count: word; value) - эта функция заполняет заданное количество COUNT последовательных байт переменной X указанным значением VALUE.

Контрольные вопросы

1. Дайте определение строковой переменной.
2. Какие типы данных используются в качестве базовых в строковых данных?
3. Каким образом распределяется память под строковые переменные?
4. Какие операции выполняются над строковыми переменными?
5. В чем состоит сходство и различие строковых переменных и символьных массивов?
6. Возможно ли преобразование строковых переменных?
7. Назовите основные функции над строковыми переменными и их назначение.
8. Каково назначение процедур DELETE,INSERT.

9. Как реализуется ввод и вывод строковых переменных.
10. Предложите схему преобразования действительных чисел в стринги.
11. Что такое литерный ряд?
12. В чем состоит отличие стринга от литерного ряда?

Задание к работе

Выполнить индивидуальное задание.

Методические указания

1. Строка символов формируется при вводе с клавиатуры.
2. При выполнении задания необходимо использовать стандартные функции и процедуры работы со стрингами: COPY, POS, LENGTH, INSERT, DELETE и операцию конкатенации.
3. Написать и отладить программу.
4. Написать отчет по работе.

Содержание отчета

1. Титульный лист.
2. Словесная постановка задачи.
3. Графический или текстуальный алгоритм решения задачи.
4. Листинг программы.
5. Контрольный тест и результаты тестирования программы.
6. Анализ ошибок, допущенных при программировании.
7. Ответы на контрольные вопросы по согласованию с преподавателем.

Варианты индивидуальных заданий

1. Дана строка символов. Подсчитать сколько среди символов данной строки встречается буква x.
2. Дана строка символов. Подсчитать:
   a) сколько раз в данной строке встречается символ + и сколько раз символ *;
   b) общее число вхождений символов +, -, * в строку.
3. Дана строка символов. Преобразовать данную строку, заменив в ней:
   a) все восклицательные знаки точками;
   b) каждую точку многоточием.
4. Дана строка символов S. Выяснить имеются ли в данной строке такие символы s_i, s_{i+1}, что s_i - это запятая, а s_{i+1} - это тире.
5. Даны две строки символов S1 и S2. Выяснить, верно ли, что среди символов строки S1 имеются все буквы строки S2.
6. Дана строка символов. Удалить из данной строки все группы букв вида asdf.
7. Дана строка символов. Преобразовать строку, удалив каждый символ * и повторив каждый символ, отличный от *.
8. Дана строка символов. Заменить в данной строке каждую группу букв child группой букв children.
9. Дана строка символов. Исключить из строки группы символов, расположенные между парами скобок (,), {, }. Сами скобки тоже должны быть исключены. Предполагается, что внутри каждой пары скобок нет других скобок.
10. Дана строка символов. Словом будем называть группы символов, разделенных пробелами (одним или несколькими). Подсчитать количество слов в данной строке.

ЛАБОРАТОРНАЯ РАБОТА N 9

Тема: "Сложный тип данных – множества"

Цель работы

1. Получение навыков в задании переменных типа множество и организации ввода и вывода данных типа множество.
2. Получение практических навыков в выполнении операций над множествами.

Краткие сведения из теории

9.1. Объявление переменной типа множества

В математике под множеством понимается некоторый набор элементов. Например, множество фигур на плоскости (прямоугольник, круг, ромб, квадрат).

К множествам применимы следующие операции:

1. Объединение множеств (C = A ∪ B). Каждый элемент множества C является элементом либо множества A, либо множества B.
2. Пересечение множеств (C = A ∩ B). Каждый элемент множества C является элементом множеств A и B одновременно.
3. Разность двух множеств (C = A \ B). Каждый элемент множества C является элементом множества A, но не является элементом множества B.

Например:

а) { круг, ромб } ∪ { круг, квадрат } = {круг, ромб, квадрат};
б) { круг} ∩ { круг, ромб, квадрат } = { круг };
в) { круг, ромб, квадрат } \ { круг, квадрат } = { ромб }.

Под множеством в языке Турбо - Паскаль понимают ограниченный, неупорядоченный набор различных элементов одинакового типа.

Множественный тип задается с помощью двух служебных слов SET и OF, после которых указывается базовый тип. В качестве базового типа можно использовать следующие типы: INTEGER, BYTE, CHAR, перечислимый и ограниченный.

При определении множественных типов существует два ограничения:

1) вещественный тип в качестве базового в множествах использовать нельзя;
2) число элементов в множестве определяется каждой конкретной реализацией ЭВМ. Обычно число элементов колеблется между 64 и 256 (для Турбо - Паскаля - 256). Такая зависимость приводит к потере переносимости программ, обладающих этим типом, с машины на машину.

Множества объявляются либо в разделе описания переменных VAR, либо в разделе описании типов TYPE. Объявление множества в разделе описания переменных имеет вид:

```
VAR
    < имя множества > : SET OF < базовый тип >;
```

Например:

```
Var
    god    : set of 1900..2000;
    symbol : set of char;
```

Объявление множества с использованием раздела описания типов имеет вид:

```
Type
    < имя типа > = set of < базовый тип >;
Var
    < имя множества > : имя типа;
```

Например:

```
Type
   god    = set of 1900..2000;
   symbol = ( 'A'..'Z');
Var
```

g : god;
s : set of symbol;

Значения переменных и констант множества задаются в разделе операторов с помощью конструктора. Конструктор представляет собой список элементов базового типа, заключенный в квадратные скобки, который затем можно присвоить переменной, или обработать.

Конструктор множества можно рассматривать как константу типа множества.

Например:

figura := [romb];

или

figura := [krug,romb,kvadrat];
simv := ['A','B','C'];
M1 := [1,3,5,10];
M2 := []; { пустое множество }

9.2. Операции над множествами

В языке Турбо-Паскаль имеются следующие группы операций над множествами:

1) объединение, пересечение, вычитание множеств;
2) проверка принадлежности элемента множеству;
3) проверка на равенство и неравенство множеств;
4) проверка на принадлежность одного множества другому.

Операции объединения, пересечения и вычитания являются традиционными действиями над множествами и обозначаются символами '+', '*', '-' соответственно. ***Например***:

[1,2] + [3,4] = [1,2,3,4];
[1..10] + [5..15] = [1..15];
[1..10] * [5..15] = [5..10];
[1,2] * [3,4] = [];
[1..10] - [5..15] = [1..4];

Проверка принадлежности множеству - это логическая операция, которая обозначается служебным словом IN. Правый операнд должен быть множеством, левый - значением базового типа множества. Операция возвращает TRUE, если значение входит в множество, и FALSE в противном случае. ***Например***:

2 in [1..10,12]; { имеет значение true}
5 in [1,2,7,10]; { имеет значение false}

Операцию проверки принадлежности удобно использовать для исключения более сложных проверок, например, оператор вида

if (symb = 'a') or (symb = 'b') or (symb = 'x') or (symb ='y') then s;

может быть переписан в более компактной форме

if symb in ['a','b','x','y'] then s;.

Второй вариант эффективен с точки зрения быстродействия.

Проверка на равенство, неравенство и включение множеств - это бинарные логические операции, которые обозначаются следующими символами:

= равенство (совпадение) двух множеств;
<> неравенство множеств;
<= проверка на вхождение множества из левого операнда в множество из правого операнда;
>= проверка на вхождение множества из правого операнда в множество из левого операнда.

Все эти операции вырабатывают логическое значение TRUE или FALSE в зависимости от успеха проверки. ***Например***:

[1,2,3] = [1,2] - false;
[1,2,3] >= [1,2] - true;
[S] <= [1..10] - true,
если S - целое число из диапазона 1..10;
[1,2,3] <> [1,2,2] - true.

Синонимом логической операции над множествами является слово "компаратор".
Набор операций над множествами в языке Турбо-Паскаль не содержит одной практически важной операции - выборки значений из множества (или близко связанного с ней средства циклического перебора значений множества). Поэтому при необходимости подобных действий приходится организовывать цикл по всему диапазону значений базового типа, проверяя на каждой итерации принадлежность очередного значения данному множеству, ***например***:

```
Var
    symbols : set of char;
              s : char;
Begin
     . . .
   For s := chr(0) to chr(255) do
     if s in symbols then
       < действия с переменной s >
     . . .
```

Контрольные вопросы

1. Что понимается под множеством?
2. Какие вы знаете операции над множесвами в математике?
3. Как записываются операции над множествами в языке Турбо-Паскаль?
4. Как задаются множества на языке Турбо-Паскаль?
5. Что такое пустое множество и как оно задается?
6. Как организовать вывод элементов множества?

Задание к работе

1. Выполнить задание А.
2. Выполнить задание Б.

Методические указания

1. При выполнении индивидуального задания А необходимо:
   a) ознакомиться с конечным и упорядоченным множеством символов, определенным на используемой для выполнения задания ЭВМ;
   b) составить программу для конкретного варианта, работающую для произвольного набора символов.
   c) входная строка символов может быть длиннее строки экрана терминала, при этом программа работает не с функцией EOLN, а с признаком конца строки, который задается программистом.

2. При выполнении индивидуального задания Б необходимо учесть приемы программирования, использованные в приведенной ниже программе ASMAG.

Известен набор продуктов - хлеб, масло, сыр, молоко, имеющихся в ассортименте магазинов. В три магазина доставлены отдельные виды этих продуктов. Требуется построить множества A, B, C, которые содержат соответственно:
- продукты, имеющиеся одновременно во всех магазинах;
- продукты, имеющиеся по крайней мере в одном из магазинов;
- продукты, которых нет ни в одном из магазинов.

```
Program ASMAG;
   Const N=3;
   Type
      product=(bread,butter,cheese,milk); {задается список объектов (продуктов),
определяющий базовый тип PRODUCT}
      assort = set of product;  {на базовом типе PRODUCT определя-ется множественный тип
ASSORT}
```

```
      magazin =  array  [1..N]  of assort;          {информация о наличии продуктов во всех
магазинах  задается как массив множеств}
Var
     m1 : magazin; x : product;
      a,b,c, xm1 : assort;
      i,j,iw,m : integer;
Begin
    for i := 1 to N do      {ввод исходной информации}
      begin
        xm1 := [];
        writeln (' введите номера продуктов',i : '-го магазина =');
        repeat      {в цикле REPEAT формируется множество XM1,
                     характеризующее наличие товаров в одном магазине.}
        read(iw);
        case iw of
          1:  x := bread;
          2:  x := butter;
          3:  x := cheese;
          4:  x := milk
        end;
        xm1 :=  xm1 + [x];
        until eoln;
     m1[i] := xm; {информация о наличии товаров записывается в  массив M1}
      end;
          for i := 1 to 3 do {формирование  множеств  A,B,C  и  их распечатка}
      begin
       case i of
        1: writeln('продукты, имеющиеся одновременно во
                        всех магазинах');
        2: writeln('ассортимент продуктов');
        3: writeln('продукты, которых нет ни в одном магазине')
       end;
       for x := bread to milk do
        if x IN a then
         case x of
          bread:   write('хлеб');
          butter:  write('масло');
          cheese:  write('сыр');
          milk:    write('молоко')
         end;
        if i = 1 then
         a := b
        else
         a := c;
        writeln
      end
   end.
```

Содержание отчета

1. Титульный лист.
2. Словесная постановка задачи.
3. Графический или текстуальный алгоритм решения задачи.

4. Листинг программы.
5. Контрольный тест и результаты тестирования программы.
6. Инструкция по эксплуатации программы.
7. Ответы на контрольные вопросы.

Варианты индивидуальных заданий

Задание А

Дана непустая последовательность символов. Требуется построить и напечатать множество, элементами которого являются символы, встречающиеся в последовательности индивидуального варианта:

Вариант	Последовательность символов
1	Буквы от 'A' до 'F' и от 'X' до 'Z';
2	Цифры от '5' до '9' и знаки арифметических операций;
4	Буквы от 'T до 'X' и цифры от '1' до '4';
5	Знаки препинания и операций отношения;
6	Знаки арифметических операций и буквы от 'E' до 'N';
7	Буквы от 'A' до 'Z' и знаки препинания;
8	Знаки операций отношения;
9	Цифры от '0' до '9';
10	Знаки арифметических операций и операций отношения.

Задание Б

1. Даны два конечных множества A и B, элементами которых могут быть любые целые числа в диапазоне от 1 до 30. Найти прямое произведение этих множеств и вывести его на экран.
2. Даны два прямоугольника. Множества A и B - это множества точек, принадлежащих соответствующим прямоугольникам. Координаты точек - это натуральные числа от 1 до 10. Определить пересекаются ли данные прямоугольники, если пересекаются, то вывести на экран их общие точки.
3. Даны два конечных множества X и У, состоящие из целых чисел. Определить выполняется ли равенство: $(A \cup B) \setminus B = A$.
4. Даны два конечных множества X и У, состоящие из целых чисел. Определить выполняется ли равенство: $(A \setminus B) \cup (B \setminus A) = (A \cup B) \setminus (A \cap B)$.
5. Пусть A, B, C - конечные множества, такие что $B \subseteq A \subseteq C$. Найдите множество X, удовлетворяющее условиям $A \cap X = B$ и $A \cup X = C$.
6. Пусть A, B, C - конечные множества, такие что $B \subseteq A$, $A \cap C = \varnothing$. Найдите множество X, удовлетворяющее условиям $A \setminus X = B$ и $X \setminus A = C$.
7. Даны следующие множества A={1, 2, 3}, B={2, 3, 5, 4}, U={0, 1, 2, 3, .. ,9 }. Найти и вывести на экран $A \cup B$, , $A \setminus B$, $B \setminus A$, $U \setminus A$.
8. Даны два конечных множества A и B, состоящие из целых чисел. Найти и вывести на экран $(A \cup B) \setminus (A \cap B)$.
9. Даны два множества A={1, 2}, B={3, 4, 5}. Выведите на экран элементы множеств $A \times B$, $B \times A$.
10. Пусть A={b, o}. Перечислите элементы множеств A^3 и A^4.

ЛАБОРАТОРНАЯ РАБОТА N 10

Тема: "Комбинированный тип данных - записи"

Цель работы

1. Получить навыки в организации ввода и вывода значений комбинированных типов данных.
2. Получить навыки программирования задач с использованием записей.

Краткие сведения из теории

10.1. Объявление данных типа записи

Комбинированный тип характеризует объекты, называемые записями. Синонимом понятия " комбинированный тип" является понятие "структурный тип". Запись (структура) - это сложная переменная с несколькими компонентами. При определении комбинированного типа задаются имя всей записи, имя и тип каждой компоненты. Описание комбинированного типа начинается со служебного слова RECORD и заканчивается словом END. Записи, как и другие данные, объявляются в разделе описаний и используются в разделе операторов.

Записи можно объявлять в разделе TYPE либо VAR.

Объявление записи в разделе VAR имеет следующий вид:

```
VAR
    < имя записи > : RECORD
        < имя компоненты 1: тип >;
        < имя компоненты 2: тип >;
        . . .
        < имя компоненты N: тип >
    END;
```

Здесь служебное слово RECORD (запись) выполняет роль открывающей операторной скобки, END - закрывающей. Внутри операторных скобок описываются компоненты записи. Допускается вместо имени записи указывать список имен, т.е. имена записей, разделенные запятыми. Компоненты записи вместе с их описанием называются полями записи.

Более универсальной формой объявления записи является описание с использованием раздела TYPE, которое имеет вид:

```
TYPE
    < имя типа > = RECORD
        < имя компоненты 1 >: тип;
        < имя компоненты 2 >: тип;
        . . .
        < имя компоненты N >: тип
    END;
VAR
    < имя записи >: имя типа;
```

Пример. Дана ведомость списка студентов с их оценками (рис.1) Для представленной ведомости объявление записи в разделе переменных выглядит следующим образом:

```
VAR
    vedom : RECORD
        n          : integer;
        fio        : string[15];
        progr,fizika : integer
    END;
```

№ п/п	Фамилия имя, отчество	Оценка	
		программирование	физика
1	Бадмаев И.П.	5	4
2	Иванов А.Р.	4	4
3	Павликова Ю.Т.	5	3

Рис.1. Ведомость успеваемости.

В данном примере фамилия имеет тип STRING, состоящий из 15 элементов, порядковый номер и оценки по предметам - тип INTEGER.

Длина записи Vedom равна 21 байту.

Объявление ведомости с использованием раздела типов имеет вид:

```
TYPE
 vedom = RECORD
        n          : integer;
        fio        : string[15];
        progr,fizika : integer
       END;
VAR
 v : vedom;
```

Здесь сначала введен тип с именем VEDOM, а затем указана переменная V, имеющая тип записи.

Поле записи используется в программе так же, как обычная переменная. Таким образом, поле записи можно указывать как в левой части оператора присваивания, так и в выражениях. Над полем записи можно выполнять действия, допустимые для данных его типа. Если тип поля записи - INTEGER, то выполняются все операции, допустимые для целых данных.

10.2. Доступ к полям записи

Доступ к полям записи производится с помощью селектора записи, имеющего следующий вид:

NAME_Z.NAME_P ,

где NAME_Z - имя переменной комбинированного типа (всей записи);

NAME_P - имя поля.

В практическом программировании такая запись называется уточненным именем данного.

Для переменных, введенных выше, допустимы следующие конструкции:

```
Vedom.n := 5;
Vedom.fio := 'Иванов А.Р.';
```

или

```
V.n := 35;
V.fio := ' Павликова Ю.Т.';
```

Комбинированные типы можно использовать для построения более сложных структур: массивов; файлов; вложенных структур с одним или более полей, которые, в свою очередь, могут быть записью. Например:

```
VAR
        group    : array[1..10] of vedom;
        database : file of vedom;
```

Для переменных GROUP доступ к полям записей, составляющих этот массив, производится следующим образом:

```
        . . .
        Group[i].fio := ' Бадмаев И.П. ';
        If group[i].fio = ' Бадмаев И.П. ' then
        WriteLn (group[i].progr)
        Else writeln ('Нет такой фамилии!');
```

. . .

Рассмотрим случай, когда в составе записи содержатся поля, имеющие тип записи. Пусть для комбинированного типа VEDOM необходимо хранить информацию о дате сдачи экзамена. Эту информацию можно представить в виде трех полей: месяц, день, год, дополняющих предыдущий состав типа VEDOM. Однако, логичнее дату сдачи экзамена определить как отдельный тип. Это позволит использовать тип DATE в описании других типов и переменных:

```
TYPE
    Date = RECORD
        Mounth : (jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec);
        Day    : 1..31;
        Year   : 1900..2000
    END;
```

Теперь тип DATE можно использовать в записи VEDOM:

```
TYPE
    Vedom = RECORD
        N          : integer;
        Fio        : string[15];
        Progr,Fizika : integer;
        D_exem     : date
    END;
```

Доступ к полям D_EXEM осуществляется по общим правилам, т.е. при записи селектора слева от символа 'точка ' всегда должна находиться переменная типа запись, а справа идентификатор поля этой записи, например:

```
V.D_exem.Mounth := jan;
V.D_exem.Day := 25;
```

Комбинированный тип может употребляться для спецификации параметров подпрограмм. Например, можно определить специальный тип для представления комплексных чисел как пары вещественных переменных (действительную и мнимую части комплексного числа):

```
TYPE
    Complex = RECORD
        Re,Im : real
    END;
```

Далее можно с помощью процедур определить операции над комплексными числами (сложение, умножение, деление):

```
Procedure Addc( c1,c2: complex; var R: complex);
Procedure Mulc( c1,c2: complex; var R: complex);
Procedure Divc( c1,c2: complex; var R: complex);
```

10.3. Записи с вариантами

Часто в зависимости от конкретного значения некоторого поля возникает необходимость в пределах одной записи иметь различную информацию. В таких случаях используются записи с вариантами.

Рассмотрим тип PERSON, содержащий информацию о человеке.

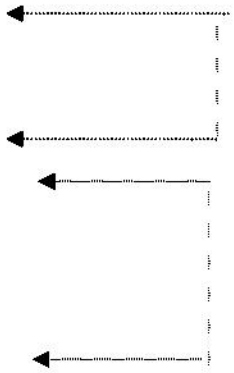
Если поле POL имеет значение M (мужской), то пусть необходимо предусмотреть такие поля:

- служил в армии или нет;
- если служил, то дату последних военных сборов.

Если поле POL имеет значение W (женский), то необходима информация о цвете глаз. Запись с вариантами типа PERSON имеет вид:

```
Type
  Date    = Record
          Mounth : (jan,feb,mar,apr,may,jun,jul,aug,
```

```
                sep,oct,nov,dec);
            Day   : 1..31;
            Year  : 1900..2000
              End;
Person   = Record
        Fio     : string[20];
        Special  : word;
        Birthday : date;
        PersonPol: (M,W);
        Case pol : PersonPol of
         M: (Army     : boolean;
           D_Army    : date);
         W: (EyesColor : (blue,brown,
                  gray,green))
        End;
```



фиксированная часть

вариантная часть

Записи с вариантами имеют фиксированную и вариантную части. Изменяющаяся часть записи называется вариантом. Вариант всегда располагается в конце записи. Поле (в данном случае POL), позволяющее различать варианты, называется полем признака.

Вариантная часть содержит несколько альтернатив (в данном примере - M и W), в каждой из которых в круглых скобках задается список полей, присущих данному варианту (ARMY и D_ARMY -> M, EYESCOLOR -> W). Списку полей предшествует метка, являющаяся конкретным значением признака POL. Метка служит критерием выбора вариантов. Перечисление альтернатив начинается с определения признака POL.

Началом вариантной части является служебное слово CASE; после признака выбора вариантов записывается служебное слово OF. Вариантная часть завершается служебным словом END вместе с завершением всей записи. В определении комбинированного типа может быть только одна вариантная часть и она должна быть задана в конце записи.

Альтернативы вариантной части помечаются допустимыми значениями поля POL, которое определено в фиксированной части. Иногда поле, значения которого задают варианты, называют дискриминантом записи.

Идентификаторы полей во всех вариантах должны быть различными и отличаться от идентификаторов полей фиксированной части. В этом случае после метки, соответствующей этим значениям может стоять пустой список вида ( ).

Любой вариант, в свою очередь, может иметь свою вариантную часть, которая должна располагаться в конце списка полей данного варианта.

При использовании вариантных записей необходимо учитывать следующие особенности:

1. Для размещения переменной комбинированного типа всегда отводится фиксированный объем памяти, причем если в записи есть варианты, то объем определяется по самому большому варианту. Различные варианты одной записи как бы накладываются " друг на друга" в памяти, занимая одну и ту же область.
2. Система Турбо-Паскаль не содержит никаких средств контроля за правильностью работы с вариантами записей. За соответствием текущего значения дискриминанты и доступа к полям записи должен следить программист.

10.4. Оператор присоединения

Оператор присоединения предназначен для более наглядной и эффективной организации работы с данными комбинированного типа и используется для доступа к полям записи. Оператор присоединения начинается со служебного слова WITH, далее следует имя записи и служебное слово DO. Операторы, содержащие имена полей записи, заключаются в операторные скобки:

```
WITH < имя записи > DO
BEGIN
        < операторы, содержащие имена полей записи >
```

```
END;
```

Например: для рассмотренной записи (списка студентов) операции присваивания, суммирования и ввода можно объединить в один оператор:

```
. . .
With v do
Begin
        N := 4;
        SUM := progr + fizika;
        Read (N)
End.
```

Контрольные вопросы

1. Что понимается под записью в языке Паскаль?
2. Как объявляются записи?
3. Какие операции допустимы над полями записи?
4. Как организовать ввод и вывод данных типа записи?
5. Как осуществляется доступ к полям записи?
6. Можно ли использовать в записи поля одного типа?
7. Чем отличается запись от массива?
8. Каково назначение оператора присоединения?

Задание к работе

Выполнить индивидуальное задание

Содержание отчета

1. Титульный лист.
2. Словесная постановка задачи.
3. Графический или текстуальный алгоритм решения задачи.
4. Листинг программы.
5. Контрольный тест и результаты тестирования программы.
6. Инструкция по эксплуатации программы.
7. Ответы на контрольные вопросы.

Методические указания

1. При выполнении работы использовать массив записей.
2. Разработать алгоритмы и программы для решения задач заданий.
3. Скомпилировать программы.
4. Составить контрольные тесты и протестировать программы.
5. Составить отчет и представить его к защите.

Варианты индивидуальных заданий

1. Дан список учебной группы, включающий 20 человек. Для каждого студента известны: фамилия, имя, дата рождения, оценки по всем дисциплинам за последний семестр. Составить программу, которая обеспечивает ввод информации и отображение ее на экран в виде таблицы. Отобразить на экран анкетные данные студентов-отличников в виде таблицы. Отобразить на экран фамилию и имя студентов, родившихся зимой и весной.
2. Сведения об экзамене содержат следующие данные: дисциплину (программирование, алгебра, история, геометрия), дату сдачи экзамена (год, месяц, день), сведения о студенте (факультет, курс, группа, номер в журнале) и экзаменационную оценку. Задан набор сведений об экзаменах, сданных студентами за последние два года; в них факультет и предмет кодируются первыми буквами названия. Определить количество неуспевающих по программированию на экономическом факультете среди студентов первого курса, сдававших экзамены зимой 1995 года, вывести на экран их группу и номер в журнале.
3. Сведения об экзамене содержат следующие данные: дисциплину (программирование, социология, иностранный язык, физика), дату сдачи экзамена (год, месяц, день), сведения

о студенте (фамилия, факультет, курс, группа) и экзаменационную оценку. Задан набор сведений об экзаменах, сданных студентами за последние несколько лет; в них факультет и предмет кодируются первыми буквами названия. Определить количество отличников по программированию на технологическом факультете среди студентов первого курса, сдававших экзамены летом 1995 года, вывести на экран их фамилии и группу.

4. Сведения об экзамене содержат следующие данные: дисциплину (программирование, вычислительная техника, информатика), дату сдачи экзамена (год, месяц, день), сведения о студенте (факультет, курс, группа, номер в журнале) и экзаменационную оценку. Задан набор сведений об экзаменах, сданных студентами за последние несколько лет; в них факультет и предмет кодируются первыми буквами названия. Определить, на каком факультете самый высокий средний балл по программированию среди студентов первого и второго курсов, сдававших экзамены зимой 1995 года.
5. Сведения об экзамене содержат следующие данные: дисциплину (программирование, вычислительная техника, информатика), дату сдачи экзамена (год, месяц, день), сведения о студенте (факультет, курс, группа, номер в журнале) и экзаменационную оценку. Задан набор сведений об экзаменах, сданных студентами за последние несколько лет; в них факультет и предмет кодируются первыми буквами названия. Определить, на каком факультете самый высокий показатель качества успеваемости по информатике (то есть самый высокий процент отличников и хорошистов) среди студентов первого курса, сдававших экзамены зимой 1995 года или летом 1996 года.
6. Справка о междугороднем телефонном разговоре содержит: номер телефона абонента (6 цифр), дату (год, месяц, день), время (час, минута), код города (3 цифры), номер телефона в другом городе (7 цифр), продолжительность разговора (в минутах), категорию (срочный, обычный) и тариф (плата в рублях за минуту). Определить дату такого телефонного разговора, которой является максимальным по продолжительности среди срочных разговоров за указанный месяц.
7. Справка о междугороднем телефонном разговоре содержит: номер телефона абонента (6 цифр), дату (год, месяц, день), время (час, минута), код города (3 цифры), номер телефона в другом городе (7 цифр), продолжительность разговора (в минутах), категорию (срочный, обычный) и тариф (плата в рублях за минуту). Вывести на экран код города и номер телефона в другом городе для телефонных разговоров, состоявшихся с телефона 235678 8 марта 1996 года.
8. Справка о междугороднем телефонном разговоре содержит: номер телефона абонента (6 цифр), дату (год, месяц, день), время (час, минута), код города (3 цифры), номер телефона в другом городе (7 цифр), продолжительность разговора (в минутах), категорию (срочный, обычный) и тариф (плата в рублях за минуту). Вывести на экран номер телефона абонента, код города и номер телефона в другом городе для срочных телефонных разговоров, состоявшихся между 15 марта и 12 апреля 1996 года .
9. Деталь автомобиля описывается инвентарным номером (положительное целое число), весом (в килограммах), ценой и стоимостью (в рублях), датой начала производства (год, месяц, день), статусом (имеет или не имеет знак качества) и объемом производства (в штуках за смену). В заданной последовательности сведений о деталях найти инвентарные номера деталей с наибольшей датой начала производства среди всех заданных деталей. Вывести на экран инвентарный номер, объем производства, цену и стоимость деталей со знаком качества.
10. Деталь автомобиля описывается инвентарным номером (положительное целое число), весом (в килограммах), ценой и стоимостью (в рублях), датой начала производства (год, месяц, день), статусом (имеет или не имеет знак качества) и объемом производства (в штуках за смену). В заданной последовательности сведений о деталях найти инвентарные номера деталей с минимальным весом среди деталей без знака качества. Вывести на экран инвентарный номер, объем производства, цену и стоимость деталей, выпускаемых с февраля 1977 года.

ЛАБОРАТОРНАЯ РАБОТА N 11

Тема: "Процедуры и функции"

Цель работы

1. Приобретение практических навыков в программировании процедур и функций.
2. Изучение механизма передачи параметров.
3. Знакомство с локальными и глобальными переменными.

Краткие сведения из теории

Процедуры и функции являются мощным средством языка программирования. Это средство удобно применять в тех случаях, когда при решении задачи возникает необходимость в программе некоторую совокупность операторов повторять несколько раз. Так например, может возникнуть необходимость один и тот же цикл использовать в нескольких местах программы.

Функцию или процедуру можно сравнить с мини-программой, именно поэтому их называют иногда одним общим именем - "подпрограмма (ПП)". ПП оформляется подобно программе: в начале записывается заголовок ПП, затем следует декларативная часть ПП и после процедурная. В декларативной части описываются все данные, область действия которых ограничена телом данной ПП. Эти данные называются локальными. Данные, объявленные в основной (главной) программе, называются глобальными и они могут использоваться в любой ПП, входящей в основную программу. В процедурной части описывается тело ПП, реализующее алгоритм решения, и которое заключается в операторные скобки BEGIN, END.

ПП помещается сразу же после объявления всех переменных. Заголовок ПП для подпрограмм-функций начинается с ключевого слова FUNCTION, для подпрограмм-процедур с ключевого слова PROCEDURE. Эти ключевые слова играют роль признаков, которые распознает компилятор. Так как ПП исполняется не сразу и возможно не один раз, то компилятор, встретив тело ПП, должен его пропустить.

11.1. Функции

Общая схема функции следующая:

FUNCTION < идентификатор функции >(<параметр: тип параметра> [,<параметр: тип параметра>, ...]) : <тип результата функции>;

< декларативная часть (разделы LABEL, CONST, TYPE, VAR) -объявление локальных данных >

BEGIN

< процедурная часть функции - тело функции >

END;

В теле основной программы функция вызывается по имени, это значит, что после FUNCTION необходимо записать идентификатор функции, а затем в круглых скобках перечислить все параметры функции. Так как язык Паскаль сильно типизирован, то он требует, чтобы после каждого параметра был указан его тип. Результатом вычисления функции всегда получается одно значение. Поэтому после круглых скобок в заголовке функции необходимо указать тип результата, вычисляемого функцией.

Ко всем функциям обращаются одинаково: в любом предложении программы они играют роль переменной.

Самым простым и наглядным примером использования функций являются стандартные функции, например, функция LENGTH(St). Эта функция может применяться в программе всякий раз, когда необходимо вычислить длину стринга, в данном случае стринга St. Все стандартные функции входят в состав компилятора, то есть они описаны в теле самого компилятора. LENGTH - это идентификатор функции, St - аргумент функции. Заголовок этой функции может быть следующий:

```
        FUNCTION Lenght(St : string) : byte;
```
Тип результата, возвращаемого этой функцией, BYTE.

Пример.

Напишем подпрограмму, позволяющую вычислять степенную функцию a^n, причем n может иметь только положительные целые значения.

В этом примере параметрами функции должны быть: основание и степень.

```
        { вычисление степенной функции }
FUNCTION Degree(Base: real; Power: integer) : real;
        { обьявление локальных переменных i, Y
        i - параметр цикла, считающий степени;
        Y - промежуточная переменная, содержащая текущие значения степенной функции }
VAR
    i : integer;
    Y : real;
        { тело функции или процедурная часть }
BEGIN
   Y:= 1;
   For i:= 1 to Power do
   Y:= Y * Base;
   Degree:= Y;
END;
```

В основной программе обращаться к этой подпрограмме (функции) можно многократно, например:

```
   . . .
VAR
    B, X, Z : real;
    P       : integer;
   . . .
BEGIN { основная или главная программа }
    . . .
    B:= 1.5;  P:= 3;
    Z:= Degree(B, P); { основание = 1.5, степень = 3 }
    . . .
    WriteLn('Значение степенной функции = ', Degree(X*2, 10));
    . . .
END.
```

Проследим работу функции по первому вызову. Значения переменных B и P: 1.5 и 3 передаются подпрограмме DEGREE. Однако после входа в подпрограмму мы эти переменные называем уже не B и P, а Base и Power. Имена Base и Power, фигурирующие в подпрограмме, являются просто "пустышками", замещаемыми при работе функции (вычислении) конкретными значениями. По терминологии ПП параметры, используемые в самой подпрограмме, называются формальными, а параметры, используемые в основной программе при ссылке к функции, называются фактическими. В нашем примере B и P - фактические параметры, Base и Power - формальные. Этот прием введения фактических и формальных параметров удобен тем, что при многократном вызове подпрограммы мы можем передавать различные параметры, как это сделано в представленном выше фрагменте программы:

Degree(B, P) и Degree(X*2, 10).

Имена формальных и фактических параметров не обязаны совпадать, хотя если случайно они и оказываются одинаковыми, никаких проблем не возникнет. Однако необходимо,

чтобы типы формального и фактического параметров были согласованы (например, оба были типа INTEGER или REAL).

При выполнении любой ПП в первую очередь происходит сопоставление формальных и фактических параметров. С этого момента ссылка на Base и Power в подпрограмме фактически означает обращение к B и P. Следовательно, команды, вычисляющие степенную функцию, используют значения B и P (1.5 и 3). Результат вычислений помещается в имя DEGREE, в этом и состоит механизм, посредством которого функция "посылает" свой ответ в основную программу. После возврата в основную программу результат, переданный функцией, назначается переменной Z. При этом значение, вычисленное функцией и помещаемое в DEGREE, имеет тот же тип, что и переменная Z.

Структуру программы, содержащей подпрограмму-функцию, можно представить следующим образом:

```
PROGRAM Main;
CONST      { декларативная
   . . .;  часть
VAR          главной
   . . .;  программы }
{ процедурная часть главной программы }
FUNCTION Sample( . . . ) : . . .; { заголовок функции }
VAR        { декларативная часть подпрограммы }
   . . .;
{ процедурная часть ПП }
BEGIN  { Sample }
   . . .
   Sample:= . . .;
END;   { Sample }
BEGIN  { главная программа }
   . . .
 < любое количество вызовов функции Sample >
   . . .
END.   { конец главной программы }
```

11.2. Процедуры

Подпрограмму следует оформить в виде процедуры, если она предназначена для решения задачи одного из двух типов. Задача первого типа: требуется выполнить некую последовательность действий, не возвращая результирующего значения. Задача второго типа: требуется изменить значения одного или нескольких фактических параметров.

Процедура, как и функция, помещается в конце декларативной части программы. Компилятор распознает процедуру по ключевому слову PROCEDURE. С этого слова начинается заголовок процедуры, после которого следует идентификатор процедуры, а затем в круглых скобках перечисляются формальные параметры процедуры.

В Паскале передать параметры подпрограмме можно двумя способами. До сих пор мы имели дело только с одним из них. Этот способ называют передачей параметров по значению. Он состоит в том, что значение фактического параметра назначается соответствующему формальному параметру. Другими словами, перед началом выполнения процедуры вычисляется конкретное значение фактического параметра (например, 1.5 или 3). Затем полученное значение копируется в соответствующий формальный параметр, принадлежащий процедуре. Как только начинается выполнение процедуры, никакие изменения значения формального параметра уже не оказывают влияния на значение соответствующего фактического параметра. Это значит, что по окончании работы процедуры фактический параметр будет иметь точно такое же значение, каким он обладал до начала работы процедуры, вне зависимости от того, что происходило с формальным параметром. Такой способ передачи параметров действует при работе с функциями.

Второй способ передачи параметров называется передачей параметров по ссылке или по адресу. При передаче параметра по ссылке в процедуру пересылается уже не значение аргумента, а его местоположение (адрес) в памяти компьютера. Чтобы сообщить компилятору о намерении передать такой параметр, в заголовке процедуры в списке формальных параметров следует указать слово VAR. Если формальный параметр снабжен атрибутом VAR, а соответствующий ему фактический параметр является переменной, то любые изменения формального параметра будут отражаться в значениях фактического параметра, поскольку теперь формальный и фактический параметры занимают одну и ту же область памяти.

Общая схема процедуры аналогична схеме функции со следующими изменениями:

ключевое слово FUNCTION заменяется на PROCEDURE;

отсутствует тип результата.

Вызов процедуры в основной программе оформляется как отдельное предложение, состоящее из имени процедуры и пары круглых скобок, в которых через запятую перечислены фактические параметры. Предложение заканчивается как обычно символом ";".

Рассмотрим пример на задачу первого типа: необходимо выполнить перемножение двух матриц A(3x4) и B(3x4).

Оформим в виде процедуры ввод матрицы. Программа в этом случае может иметь вид:

```
{*************************************************************}
{*      программа перемножения двух матриц A, B              *}
{*************************************************************}
Program Main;
Const
  N = 3; { количество строк   в матрице }
  M = 4; { количество столбцов в матрице }
Type
  MatrTyp = array[1..N, 1..M] of real;
Var
  A, B, C   : MatrTyp; { массивы можно передавать только как
                         простой тип }
  i1, i2, i3 : byte;   { индексы массивов }


{*************************************************************}
{*        процедура ввода значений матрицы                   *}
{*************************************************************}
Procedure GetMatr(NameMatr : string; var Matr : MatrTyp);
Var
  i, j : byte;
Begin   { GetMatr }
  WriteLn('Введите значения матрицы ', NameMatr);
  For i:= 1 to N do
  begin
    WriteLn('Строка ', i);
    For j:= 1 to M do
      Read(Matr[i,j]);
    WriteLn;   { перевод на новую строку }
  end;
End;   { GetMatr }

{*************************************************************}
```

```
{*                    главная программа                    *}
{*********************************************************}
Begin
  GetMatr('A', A); {Вызов процедуры для ввода значений матрицы A}
  GetMatr('B', B); {Вызов процедуры для ввода значений матрицы B}
  { алгоритм перемножения матриц }
  For i1:= 1 to N do
   For i2:= 1 to M do
   begin
    C[i1,i2]:= 0;
    For i3:= 1 to N do
     C[i1,i2]:= C[i1,i2] + A[i1,i2] * B[i3,i1];
   end;
  { вывод значений результирующей матрицы }
  For i1:= 1 to N do
  begin
   For i2:= 1 to M do
    Write(' C[',i1:1, ',', i2:1, '] = ', C[i1,i2]:8:3);
   WriteLn; { перевод на новую строку }
  end;
End.    { конец главной программы }
```

11.3. Различия между процедурами и функциями

Главное различие (из которого следуют все остальные) состоит в том, что функция всегда возвращает, причем в явной форме, одно-единственное значение, которое может быть использовано в качестве составной части выражения; процедура такого значения не возвращает. Однако применительно к процедуре все же можно говорить о возвращаемой информации - процедура способна изменять значения своих параметров (тех, что описаны с атрибутом VAR).

Помимо главного различия можно отметить ряд второстепенных различий синтаксического характера. Так, например, заголовок функции всегда завершается указанием типа возвращаемого значения. В заголовке процедуры такая информация не нужна. Для функции типично, чтобы в качестве последнего шага имени функции было назначено некоторое вычисленное значение. В процедурах этого нет. И наконец, еще одно различие. Поскольку функция возвращает какое-то значение, вызов функции может появляться прямо в выражении. Вызов процедуры не может быть частью выражения - это всегда отдельное предложение.

Контрольные вопросы

1. Для чего предназначены функции?
2. Для чего предназначены процедуры?
3. Чем отличаются формальные и фактические параметры?
4. Опишите способы передачи параметров в подпрограммы и их особенности?
4. Что включает в себя заголовок подпрограммы?
5. Чем отличаются глобальные и локальные переменные?
6. Какая разница между процедурой и функцией?

Задание к работе

1. Модифицируйте подпрограмму, вычисляющую степенную функцию так, чтобы она вычисляла и отрицательные степени.
2. Напишите подпрограмму, способную вычислять любые степени: положительные и отрицательные, целочисленные и действительные.
3. Выполните индивидуальное задание:

Дано несколько массивов чисел. Длины массивов заданы в варианте индивидуального задания. Требуется в каждом массиве найти наибольший и наименьший элементы и отобразить их на экране, затем все компоненты каждого массива возвести в квадрат и снова найти наибольший и наименьший элементы. Вычисление максимальной и минимальной величин оформить в виде процедуры, глобальные параметры в процедуре не использовать.

Методические указания

1. При выполнении пункта 2 задания необходимо использовать экспоненциальную и логарифмическую функции.
2. При выполнении пункта 3 задания необходимо знать, что:
   a) если в качестве исходной информации в процедуру передается массив, то его следует передавать по ссылке для экономии памяти, так как в этом случае при вызове процедуры не образуется локальный массив;
   b) несмотря на то, что обрабатываемые массивы разной длины, они описываются в программе как массивы одного и того же типа, так как при обращении к процедуре типы соответствующих формальных и фактических параметров должны совпадать.
3. Составить алгоритм решения задачи.
4. Написать программу и откомпилировать ее.
5. Составить контрольный тест и отладить (протестировать) программу.
6. Составить отчет и представить его к защите.

Содержание отчета

1. Титульный лист.
2. Словесная постановка задачи.
3. Графический или текстуальный алгоритм решения задачи.
4. Листинг программы.
5. Контрольный тест и результаты тестирования программы.
6. Ответы на контрольные вопросы.

Варианты индивидуальных заданий

Ввести и обработать:

1) два двумерных массива, содержащие соответственно 3x5 и 4x8 вещественных элементов;
2) три массива, содержащие соответственно 3, 6 и 8 целых элементов без знака;
3) четыре массива, содержащие соответственно 4, 6, 3 и 5 целых элементов со знаком;
4) два массива, содержащие соответственно 4 и 6 вещественных элементов;
5) три массива, содержащие соответственно 5, 10 и 4 целых элементов без знака;
6) четыре массива, содержащие соответственно 3, 5, 8 и 6 вещественных элементов;
7) два трехмерных массива, содержащие соответственно 2x3x2 и 3x4x2 вещественных элементов;
8) четыре массива, содержащие соответственно 4, 7, 3 и 5 вещественных элементов;
9) три двумерных массива, содержащие соответственно 2x5, 3x6 и 3x4 целых элементов без знака;
10) два двумерных массива, содержащие соответственно 6x2 и 3x2 вещественных элементов.

ЛАБОРАТОРНАЯ РАБОТА N 12

Тема: " Работа с файлами"

Цель работы

1. Изучение файловых типов в языке Турбо-Паскаль.
2. Получение навыков в организации файлов и использовании их для обработки информации.

Краткие сведения из теории

До сих пор данные вводились в программу через клавиатуру, т.е. с непременным участием человека. Такой способ ввода информации называется интерактивным режимом ввода. Возможен и иной подход, основывающийся на использовании набора данных, подготовленных заранее и хранящихся в виде файла на магнитном носителе. Иначе говоря, альтернативой интерактивному режиму является такой способ ввода, при котором информация поступает из источника, физически существующего вне программы. Этот процесс обычно называют считыванием данных из внешнего файла (или просто из файла). Указанный способ находит широкое применение при обработке информационных массивов весьма значительного объема, когда интерактивный режим становится слишком обременительным и малоэффективным. Вторым мотивом использования файлов является то, что он может быть создан какой-то другой программой. Таким образом, файл становится связующим звеном между разными задачами. Наконец, последнее немаловажное соображение: если входные данные поступают в программу из внешнего файла, то присутствие пользователя в момент фактического исполнения программы становится необязательным.

Файл - это именованная область внешней памяти компьютера (жесткого диска, гибкой дискеты,...), содержащая логически связанную совокупность данных.

На языке Паскаль можно создавать три типа файлов : текстовый, типизированный, нетипизированный, которые в программе (в разделе VAR) объявляются следующим образом:

< файловая переменная > : text;
< файловая переменная > : file of < тип компоненты >;
< файловая переменная > : file.

< файловая переменная > - это логическое имя файла, используемое в программе.

< тип компоненты > - компонентой файла может быть как переменная базового типа, так и структурного. Структурный тип определяет данные типа "запись" (record).

"Запись" - это логически связанная совокупность нескольких данных. К данным типа RECORD можно обращаться как ко всей совокупности данных, так и к ее отдельным данным (элементам). Для обращения к отдельным элементам "записи" используется уточненное имя.

Уточненное имя состоит из идентификатора "записи", десятичной точки и идентификатора элемента записи. В свою очередь каждый элемент "записи" может быть "записью". Тогда, для обращения к внутреннему элементу необходимо последовательно перечислить через десятичную точку все идентификаторы иерархически вложенных "записей", начиная от внешнего имени к внутреннему, последним в этой последовательности является идентификатор самого элемента.

При использовании структурного типа компоненты "запись" необходимо объявлять его в программе в разделе TYPE.

Пример:

```
TYPE
{ запись данных по студенту }
RecFile = RECORD { признак начала записи }
        Fam, Name, Otch      : string[15];
```

```
        GodR            : word;
        Ngrup           : string[10];
END;  { конец записи }
VAR
        F1  :  Text;
        F2  :  File of byte;
        F3  :  File of RecFile;
        F4  :  File;
        Buf :  RecFile; {_2буфер ввода-вывода_0, в который считываются  данные компоненты
        файла}
```

В этом фрагменте программы F1- файловая переменная текстового типа, F2 и F3 - файловые переменные типизированного типа, причем F2 может содержать только байтовые компоненты, а в файле F3 каждая компонента представляет из себя "запись" из трех элементов, F4- файловая переменная нетипизированного типа.

Уточненные имена элементов записи Buf: Buf.Fam, Buf.Name, Buf.Otch, Buf.GodR, Buf.NGrup.

12.1 Доступ к файлам

В первую очередь при работе с файлами необходимо привести в соответствие файловую переменную и имя файла, с которым он хранится на внешнем носителе. С этим именем работает операционная система(ОС) DOS. Соответствие устанавливается с помощью процедуры:

ASSIGN (< ф.п.>, < имя файла или л.у.>);

Здесь < ф.п.> - файловая переменная; < имя файла или л.у.> - это полное имя файла, которое включает в себя путь доступа, непосредственно имя файла и расширение. "л.у." - стандартное логическое устройство.

Например:

```
ASSIGN (F1, 'a:\Tp5\DAT\St629.DAT');
ASSIGN (F2, 'Dannye.DAT').
```

Если не указан путь к файлу, то запись или считывание осуществляется с текущего директория или в текущий директорий.

В качестве имени файла в процедуре ASSIGN можно указывать логическое устройство из следующего списка: CON, PRN, AUX.

CON - это имя консоли. На персональном компьютере (ПК) под консолью понимается совокупность двух устройств: клавиатуры и дисплея. Клавиатура используется для ввода информации, а дисплей для вывода.

PRN - это стандартное имя принтера. В ОС PRN стандартно назначается LPT1. В модуле Printer Турбо-Паскаля объявлена файловая переменная Lst. Поэтому при отображении данных на принтер, достаточно подключить модуль Printer к программе, а в предложениях Write и Writeln первым аргументом записать имя файловой переменной Lst:

Write (Lst, < список выводимых данных >);

Пример вывода информации без использования модуля Printer :

```
...
VAR F : File;
BEGIN
        Assign(F, PRN);
        ReWrite(F);
        Writeln(F, 'Пример использования Л.У. - PRN');
        Close(F);
END;
```

AUX - это имя коммуникационного порта. Обычно их бывает два у ПК: COM1 и COM2. Стандартно AUX назначается COM1. Этот порт обычно используется для подключения нестандартных устройств. Например, "мыши", дигитайзера, графопостроителя и т.п.

12.2 Инициация файла и завершение работы с ним

Прежде чем начать обработку файла необходимо выполнить некоторые операции по работе с устройством, на котором хранится или будет храниться файл. Так например, при создании файла необходимо:
выделить область памяти на внешнем устройстве, в которую будут записываться данные файла;
запомнить имя файла и адрес этой области.
Если предстоит работа с файлом, уже существующим на внешнем носителе, то необходимы следующие действия:
по указанному имени файла найти адрес, с которого записаны данные этого файла;
установить головку устройства на начало файла.
Эта совокупность операций называется инициацией файла или "открытием" файла.
Инициируется файл с помощью процедур Reset и ReWrite. С помощью процедуры Reset инициируется, т.е. открывается ранее созданный файл. С помощью процедуры ReWrite инициируется файл для записи, то есть вновь создаваемый файл.
Синтаксис:

Reset(< ф.п.> [,< размер записи в байтах >]);
ReWrite(< ф.п.> [,< размер записи в байтах >]);

Второй аргумент указывается только для нетипизированных файлов. Текстовые файлы можно инициировать также и процедурой Append:

Append(< ф.п.>);

В этом случае ранее созданный файл открывается для добавления данных в конец файла.
Завершив работу с файлом, необходимо его закрыть. При закрытии файла ОС подсчитывает размер файла в байтах и запоминает его. Кроме того, запоминается также информация о дате и времени создания файла или его последней модификации (корректировки).
Закрытие файла данных осуществляется процедурой Close:

Close(< ф.п.>);

При считывании данных из ранее созданного файла конец файла можно определить с помощью функции EOF:

EOF(< ф.п.>);

Эта функция имеет значение TRUE при считывании маркера конца файла. В противном случае она будет иметь значение FALSE. Данная функция обычно используется для организации цикла по чтению всех компонент файла:

```
...
while not EOF(F1) do begin
    ...
    < считывание и обработка компонент файла >
    ...
end;
```

12.3. Считывание данных из файла и запись их в файл

Непосредственный ввод информации осуществляется предложениями READ и READLN, а вывод (запись) информации - WRITE и WRITELN. Особенностью их применения к файлу является обязательность указания файловой переменной в качестве первого параметра в списке элементов ввода или вывода:

Read(< файловая переменная >, < список ввода >);
ReadLn(< файловая переменная >, < список ввода >);
Write(< файловая переменная >, < список вывода >);
WriteLn(< файловая переменная >,< список вывода >).

12.4. Текстовые файлы

Текстовый файл - это совокупность строк переменной длины. Переменная длина строк определяет наличие маркеров, которые отмечают конец строки. В качестве маркеров используются два управляющих символа "Перевод строки" и "Возврат каретки", их десятичные коды: #10, #13. Названия управляющих символов "Перевод строки"(LF - Line Feed) и "Возврат каретки"(CR - Carriage Return) взяты по аналогии работы с пишущей машинкой.

Конец строки можно определить с помощью функции EOLn:

EOLn(< ф.п.>);

Для записи данных в файл используются процедуры WRITE и WRITELN:

Write(< ф.п.>, < список вывода стрингов >);

Writeln(< ф.п.>, < список вывода стрингов >).

По предложению WRITE значения данных из списка запишутся в файл подряд без всяких разделителей. Поэтому программист, используя предложение WRITE, должен позаботиться о разделителях между данными, если они нужны.

По предложению WRITELN в файле после каждого выведенного стрингового значения будут записаны признаки конца строки.

Для чтения данных из файла используются процедуры READ и READLN:

Read(< ф.п.>, < список вводимых стрингов >);

Readln(< ф.п.>, < список вводимых стрингов >);

По предложению READ из файла выбирается столько символов, сколько указано в описании текущего стринга, принадлежащего списку ввода. Выбранная последовательность символов присваивается текущему стрингу. Эта совокупность операций повторяется для всех элементов списка ввода. По предложению READLN из файла последовательно считываются строки и присваиваются стрингам из списков. Если выбранная строка имеет большее количество символов, чем указано в описании текущего стринга, то она обрезается до указанной длины, при этом часть информации теряется. Поэтому необходимо следить за соответствием длин стрингов, записываемых в файл и считываемых из файла.

Пример:

```
...
Var
        Fio, Otch   : string[15];
        Name        : string[10];
        i           : integer;
        F           : text;
 Begin
        Assign(F, 'St629.DAT'); { файл будет создаваться в текущем каталоге }
        { создание файла или первичная запись данных в файл}
        ReWrite(F); { открытие файла для записи }
        for i:=1 to 5 do { ограничимся вводом пяти студентов }
        begin
                Write('Фамилия: '); Readln(Fam);
                Write('Имя: ');     Readln(Name);
                Write('Отчество: '); ReadLn(Otch);
                Write(F, Fam, Name, Otch);
        end;
        close(F);
        { чтение данных из файла и вывод их на экран }
        WriteLn('  Фамилия      Имя      Отчество');
        Reset(F); { открытие существующего файла }
        for i:=1 to 5 do
        begin
```

```
            Read(F, Fam, Name, Otch);
            Write(Fam:16, Name:11, Otch:15);
        end;
        close(F);
End.
```

12.5. Типизированные файлы

Компоненты этого файла могут быть следующих типов:
базового: byte, word, longint, integer, real, запись, char, string;
структурного;
регулярного.
При этом все компоненты файла имеет один и тот же тип. Это означает, что длина компоненты фиксирована.
Объявляется такой файл в программе следующим образом :

```
Var
        F1 : File of byte;
        F2 : File of string[80];
        F3 : File of real;
        . . .
        F  : File of RecFile;
```

Здесь F1, F2, F3, F - это файловые переменные, которые указывают на файлы, компонеты которых соответственно являются типа byte, string, real и record.
Чтение компонент файла выполняется процедурой:

```
        Read(< ф.п.>, < список ввода >);
```

Запись компонент в файл выполняется процедурой:

```
        Write(< ф.п.>, < список вывода >);
```

Пример:

```
Var
        X, Y : array[1..100] of integer; { массивы координат }
        F    : file of real;
        i    : byte;
Begin
        . . .
        < операторы по вводу 100 значений координат X, Y >
        . . .
        Assign(F, 'Coor.dat');   { файл будет создаваться в текущем каталоге }
        ReWrite(F);   { открытие файла для записи }
        For i:= 1 to 100 do
        Write(F, X[i], Y[i]);   { запись координат в файл }
        Close(F);
End.
```

В приведенном фрагменте программы координаты записаны последовательными парами X, Y. При такой организации файла происходит частое обращение к внешнему носителю, это приводит к замедлению работы программы, что особенно заметно при работе с большими объемами данных. Поэтому рекомендуется данные записывать в файл и считывать из файла большими блоками, примерно кратными 512 байтам.
Согласно этому модифицируем программу следующим образом:

```
Type
        Coord = array[1..100] of integer; { массивы координат }
        . . .
Var
```

```
    X, Y : Coord; { массивы координат }
    F    : file of Coord;  { файл регулярного типа }
    i    : byte;
Begin
    . . .
    < операторы по вводу 100 значений координат X, Y >
    . . .
    Assign(F, 'Coor.dat');   { файл будет создаваться в текущем каталоге }
    ReWrite(F);   { открытие файла для записи }
    Write(F, X);   { запись массива координат X в файл }
    Write(F, Y);   { запись массива координат Y в файл }
    Close(F);
End.
```

Теперь в файле сначала записаны 100 координат X, а затем 100 координат Y. Данные файла мы записали двумя большими блоками по 600 байтов каждый. Следует помнить, что это не самый лучший способ организации файла для данных примера, но достаточно понятный.

Считывание координат из файла выполняется аналогично, в программе нужно вместо процедуры REWRITE использовать процедуру RESET, а вместо предложения WRITE использовать предложение READ.

Хорошо структурируемые данные, например, данные о каком-либо объекте, удобно описывать типом "запись". В этом случае компонента файла будет структурного типа.

Пример:

```
Type
    RecFile = record    { запись данных по студенту }
        Fam, Name, Otch : string[15];
        GodR            : word;
        NGrup           : string[10];
    end;
Var
    i      : integer;
    Buf    : RecFile;
    FilStud : file of RecFile;
Begin
    Assign(FilStud, 'Stud.dat');
    ReWrite(FilStud);
    WriteLn('Введите данные по студентам:');
    For i:= 1 to 10 do { ограничимся 10 записями }
    begin
    { интерактивный ввод данных }
        Write('Фамилия     : '); Readln(Buf.Fam);
        Write('Имя         : '); Readln(Buf.Name);
        Write('Отчество    : '); Resdln(Buf.Otch);
        Write('Год рождения : '); Readln(Buf.GodR);
        Write('N группы    : '); Readln(Buf.NGrup);
    { запись данных в файл }
        Write(FilStud, Buf); { Buf - обращение ко всей записи }
    end;
    Close(FilStud);
End.
```

Чтение компонент типизированного файла можно осуществлять как последовательным, так и прямым методом доступа.

Последовательный доступ - это есть доступ к компоненте файла только после перебора всех предыдущих.
Прямой доступ - это есть доступ сразу к указанной компоненте.
Так как типизированные файлы обладают компонентами фиксированной длины, существует возможность организовать прямой доступ. Для организации прямого доступа к компонентам файла существуют стандартные процедуры Seek, FilePos, FileSize :

Seek(< файловая переменная >,< номер компоненты >);
FilePos(< файловая переменная >);
FileSize(< файловая переменная >).

Процедура Seek осуществляет прямой доступ к любой компоненте файла.
Здесь < номер компоненты > - позиция указателя компонент файла.
Она может принимать следующие значения:

+1 - установить указатель на следующую компоненту;
-1 - установить указатель на предыдущую компоненту;
i - установить указатель на i-ую компоненту.

Процедура FilePos определяет номер текущей позиции в файле, а точнее номер текущей компоненты.
Процедура FileSize определяет размер указанного файла - количество компонент.
Нумерация компонент начинается с нуля.

Пример.

```
Type
 RecFile = record   { запись данных по студенту }
           Fam, Name, Otch : string[15];
           GodR          : word;
           NGrup         : string[10];
          end;
Var
 i      : integer;
 Buf    : RecFile;
 FilStud : file of RecFile;
Begin
 Assign(FilStud, 'Stud.dat');
 Reset(FilStud);
 i:= FileSize;
 WriteLn('В файле ', i, ' компонент');
 Seek(FilStud, i-1);  { встали перед последней записью }
 Read(FilStud, Buf);  { прочитали ее }
 { можно ее скорректировать и записать вновь в файл    }
 Buf.GodR:= '1973';
 { перед записью нужно вновь установить указатель перед этой
 записью   }
 Seek(FilStud,  -1);
 Write(FilStud,  Buf);
 Close(FilStud);
End.
```

Примечание: Открывая типизированный файл процедурой RESET, можно этот файл не только читать, но и записывать в него новую информацию. При этом файл должен уже существовать на диске.

12.6. Нетипизированные файлы

Нетипизированные файлы могут содержать в своем составе любые типы компонент. При этом правильность записи и считывание этих компонент полностью возлагается на

программиста. Длина компонент может быть различной. Для открытия нетипизированных файлов используются процедуры Reset, ReWrite:
Reset(< файловая переменная >, < max размер буфера >);
ReWrite(< файловая переменная >, < max размер буфера >).
Так как за одно обращение к нетипизированному файлу можно считывать не одну компоненту, а несколько, и так как длины компонент могут быть различны, то в процедурах Reset и ReWrite указывается максимальный размер буфера ввода-вывода в байтах.
Чтение компонент из файла и запись их в файл выполняется процедурами BlockRead и BlockWrite:
BlockRead(< файловая переменная >, < буфер >,< кол-во компонент, считываемых за один раз >, [,< кол-во считанных компонент >]);
BlockWrite(< файловая переменная >,< буфер >,< кол-во записываемых компонент >, [,< кол-во записанных компонент >]).
Четвертый параметр необязателен. Он формируется системой и используется для проверки правильности завершения операций чтения или записи.
Нетипизированные файлы рекомендуется использовать для организации эффективной работы с файлами, так как они позволяют работать, во-первых, с компонентами различной длины, и, во-вторых, с переменным числом обрабатываемых компонент.

2.7. Процедуры и функции для работы с файлами

Все рассматриваемые функции и процедуры принадлежат стандартному модулю DOS, поэтому его необходимо подключить к программе с помощью предложения USES.
1. ReName(< файловая переменная >,< новое имя файла >) - переименование файла.
2. Erase(< файловая переменная >) - удаление файла.
3. ChDir(< путь >) - изменение директория, где <путь> - путь к новому директорию.
4. GetDir(< устройство >, < директорий >) - определение текущего каталога, где <устройство> задается следующим образом:
   0 - текущее устройство;
   1 - устройство A;
   2 - устройство B и т.д.
5. MkDir(< директорий >) - создание нового каталога. В аргументе <директорий> указывается полный путь до того каталога, который создается.
6. PmDir(< директорий >) - удаление каталога. В качестве аргумента указывается полный путь до удаляемого каталога. При этом удаляемый каталог должен быть обязательно пустым.
7. IOResult - проверка правильности завершения работы той или иной операции ввода-вывода. Эта функция имеет тип WORD и возвращает значение 0, если операция ввода-вывода выполнилась успешно, и в противном случае следующие значения:
   1 - файл не найден,
   2 - путь не найден,
   3 - слишком много открытых файлов,
   5 - запрет доступа к файлу,
   12 - некорректный код доступа к файлу
   и так далее.
При применении этой функции в программе необходимо с помощью директивы компилятора отключить стандартную проверку - {$I-}, а после выполнения операций ввода-вывода включить - {$I+}.
Данная функция записана в стандартном модуле SYSTEM.
8. DiskFree(< устройство >) - определение числа свободных байтов на заданном диске. Эта функция типа LONGINT.

В качестве аргумента указывается номер устройства. Если указано несуществующее устройство, то вместо объема свободной памяти на диске эта функция возвращает значение -1. Функцию рекомендуется применять перед созданием файла, чтобы выяснить, достаточно ли места для создаваемого файла на указанном накопителе.
9. DiskSize(< устройство >) - определение числа свободных байтов на диске. Тип функции LONGINT. Аргумент задается так же, как и в предыдущей функции.
10. FindFirst(< уточненное имя файла>, < атрибуты >, < доп.инф-я >) - поиск указанного файла.
В процедуре входным параметром является только первый. Два последних параметра являются выходными. Параметр < атрибуты > имеет тип BYTE, параметр < дополнительная информация > должен быть объявлен как SearchRec. Этот тип описан в стандартном модуле Dos.
11. FindNext(< следующий файл >) - поиск указанного файла. Процедуры FindFirst и FindNext зачастую используются для просмотра всех файлов, находящихся в каталоге.
12. FSearch(< имя файла>,<список каталогов >) - поиск файла в списке каталогов. Функция имеет тип PathStr (описана в стандартном модуле Dos).
13. FSplit(< уточненное имя файла >, < путь >,< имя >,< расширение >) - выделение из уточненного имени файла трех переменных:
< путь >, < имя файла >, < расширение >.
14. FExpand(< имя файла >) - добавление к имени файла, находящегося в текущем каталоге, полного пути доступа к нему.
Примечание: перед использованием первых четырех процедур файл должен быть обязательно закрыт.

Контрольные вопросы

1. Укажите режимы ввода информации.
2. В каких случаях удобно использовать файлы?
3. Дайте определение файла и укажите его характеристики.
4. Что такое путь доступа к файлу?
5. Где хранятся файлы ?
6. Выведите формулу подсчета объема файла в байтах.
7. Каким образом описываются переменные файловых типов ?
8. Как подразделяются файлы по видам доступа к его компонентам ? Как осуществляется доступ к компонентам файлов ?
9. Какие операции определены над файлами ?

Задание к работе

Задание А. Разработать программу в соответствии с вариантом задания, которая должна выполнять следующие функции:
создание файла;
чтение данных из файла;
вывод считанных данных на экран дисплея.
Задание Б. В программу, разработанную по заданию А, добавить блок обработки данных, инцидентных файлу, в соответствии с индивидуальным заданием. Все полученные результаты отобразить на экране.

Методические указания

1. При разработке процедуры создания файла необходимо придерживаться следующей схемы действий:
   a) проверить с помощью процедуры DISKSIZE, есть ли место на диске;
   b) проверить, нет ли файла с таким же DOS - им именем на диске (процедура FINDFIRST, FINDNEXT);
   c) привести в соответствие DOS - ое имя файла с файловой переменной, используемой в программе (процедура ASSIGN);

d) открыть файл (процедура REWRITE);
e) ввести данные, предназначенные для записи в файл;
f) записать данные в файл (предложения WRITE / WRITELN);
g) закрыть файл (процедура CLOSE).

2. При создании процедуры чтения необходимо:
   a) проверить, существует ли такой файл на диске (процедура FINDFIRST, FINDNEXT);
   b) если файл не существует, то необходимо уточнить имя в интерактивном режиме и снова перейти к пункту а);
   c) если файл существует, привести в соответствие DOS - ое имя файла с файловой переменной, используемой в программе (процедура ASSIGN);
   d) открыть файл (процедура RESET);
   e) считать данные из файла (предложения READ / READLN);
   f) отобразить считанные данные на экране дисплея;
   g) закрыть файл (процедура CLOSE).
3. В начале каждой процедуры необходимо:
   a) отключить стандартную проверку выполнения операций ввода-вывода, используя директиву компилятора {$I-};
   b) после выполнения каждой операции ввода-вывода самостоятельно проверять код ее завершения с помощью функции IORESULT;
   c) при неуспешном завершении операции ввода-вывода устранить причину , приведшую к этой ситуации.

Содержание отчета

1. Титульный лист.
2. Словесная постановка задачи.
3. Математическая постановка задачи.
4. Таблица идентификаторов входных и выходных данных, а также их типов.
5. Листинг программы.
6. Контрольный тест.
7. Результаты тестирования.
8. Анализ допущенных ошибок.
9. Инструкция по эксплуатации программы.
10. Ответы на контрольные вопросы.

Варианты индивидуальных заданий

Вариант 1

А. Создать файл, содержащий сведения о месячной заработной плате рабочих завода. Каждая запись содержит поля - фамилия рабочего, наименование цеха, размер заработной платы за месяц. Количество записей - произвольное.

Б. Вычислить общую сумму выплат за месяц по цеху X, а также среднемесячный заработок рабочего этого цеха. Вывести ведомость для начисления заработной платы рабочим этого цеха.

Вариант 2

А. Создать файл, содержащий сведения о количестве изделий, собранных сборщиками цеха за неделю. Каждая запись содержит поля - фамилия сборщика, количество изделий, собранных им ежедневно в течение шестидневной недели (в понедельник, вторник и т.д.). Количество записей - произвольное.

Б. По каждому сборщику просуммировать количество деталей, собранное им за неделю. Определить сборщика, собравшего наибольшее число изделий, и день, когда он достиг наивысшей производительности труда.

Вариант 3

А. Создать файл, содержащий сведения о количестве изделий категорий A, B, C, собранных рабочим за месяц. Структура записи имеет поля - фамилия сборщика, наименование цеха, количество изделий по категориям, собранных рабочим за месяц. Количество записей - произвольное.

Б. Считая заданными значения расценок Sa, Sb, Sc за выполненную работу по сборке единицы изделия категорий A, B, C соответственно, подсчитать:

- общее количество изделий категорий A, B, C, собранных рабочим цеха X;
- ведомость заработной платы рабочих цеха X;
- средний размер заработной платы работников этого цеха.

Вариант 4

А. Создать файл, содержащий сведения о телефонах абонентов. Каждая запись имеет поля - фамилия абонента, год установки телефона, номер телефона. Количество записей - произвольное.

Б. По вводимой фамилии абонента выдать номер телефона. Определить количество установленных телефонов с XXXX года. Номер года вводится с терминала.

Вариант 5

А. Создать файл, содержащий сведения об ассортименте игрушек в магазине. Структура записи - название игрушки, цена, количество, возрастные границы, например 2 ÷ 5, т.е. от двух до пяти лет. Количество записей - произвольное.

Б. Найти игрушки, которые подходят детям от 1 до 3 лет. Определить стоимость самой дорогой игрушки и ее наименование. Определить игрушку, которая по стоимости не превышает X руб. и подходит ребенку в возрасте от A до B лет. Значения X, A, B ввести с терминала.

Вариант 6

А. Создать файл, содержащий сведения о сдаче студентами первого курса сессии. Структура записи - индекс группы, фамилия студента, оценки по пяти экзаменам, признак участия в общественной работе: "1" - активное участие, "0" - неучастие. Количество записей - 30.

Б. Зачислить студентов группы X на стипендию. Студент, получивший все оценки "5" и активно участвующий в общественной работе, зачисляется на повышенную стипендию (доплата 50 %), не активно участвует - доплата 25 %. Студенты, получившие "4" и "5" , зачисляются на обычную стипендию. Студент, получивший одну оценку "3", но активно занимающийся общественной работой, также зачисляется на стипендию, в противном случае зачисление не производится. Индекс группы вводится с терминала.

Вариант 7

А. Создать файл, содержащий сведения о сдаче студентами сессии. Структура записи - индекс группы, фамилия студента, оценки по пяти экзаменам и пяти зачетам ("З" означает зачет, "Н" - незачет). Количество записей - 25.

Б. Определить фамилии неуспевающих студентов с указанием индексов групп и количества задолженностей. Найти средний балл, полученный каждым студентом группы X, и всей группой в целом.

Вариант 8

А. Создать файл, содержащий сведения о личной коллекции книголюба. Структура записи - шифр книги, автор, название, год издания, местоположение (номер стеллажа, шкафа и т.д.). Количество записей - произвольное.

Б. Найти:

1) местонахождение книги автора X названия Y;
2) список книг автора Z, находящихся в коллекции;
3) число книг издания XX года, имеющееся в библиотеке.

Значения X, Y, Z, XX ввести с терминала;

Вариант 9

А. Создать файл, содержащий сведения о наличии билетов и рейсах Аэрофлота. Структура записи - номер рейса, пункт назначения, время вылета, время прибытия, количество свободных мест в салоне. Количество записей - произвольное.

Б. Найти время отправления самолетов в город X; наличие свободных мест на рейс в город X с временем отправления Y.

Значения X, Y вводятся по запросу с терминала.

Вариант 10

А. Создать файл, содержащий сведения об ассортименте обуви в магазине фирмы. Структура записи - артикул, наименование, количество, стоимость одной пары. Количество записей - произвольное. Артикул начинается с буквы Д для дамской обуви, М для мужской, Р для детской.

Б. Определить наличие в файле обуви артикула X, узнать ее стоимость; ассортиментный список дамской обуви с указанием наименования и имеющегося в наличии числа пар каждой модели.

Значение X вводится по запросу с терминала.

ЛАБОРАТОРНАЯ РАБОТА N 13

Тема: Ссылочный тип данных

Цель работы

1. Ознакомиться с простой динамической структурой данных -однонаправленным списком.
2. Получить навыки работы с переменными ссылочного типа.
3. Получить навыки программирования списков и операций над ними.

Краткие сведения из теории

13.1. Объявление переменной ссылочного типа

Любой ссылочный тип определяет множество значений, которые являются указателями на значения некоторого типа.

Тип, на значения которого можно конструировать указатели, может быть любым: простым (базовым и переменным) и сложным. Будем называть этот тип базовым.

Переменная ссылочного типа содержит адрес ячейки памяти, в которой расположено конкретное значение базового типа. Для описания ссылочных типов используется символ " ^ " (англ. CARET - ссылка, синоним POINTER - указатель) и идентификатор типа, например:

```
TYPE
    P = ^integer;
```

Это описание определяет множество указателей на целые значения.

Ссылочные типы в описаниях переменных можно задавать как осредством идентификаторов, так и явно. ***Например***:

```
TYPE
        Person = RECORD
                Name,SecondName,SurName : string[20];
                Sex : (male,female);
                Speciality: word
        END;
VAR
        P1,P2 : P;    { тип P введен выше }
        Oneman : ^ Person;
```

Значение переменной Oneman ссылается (указывает) на некоторое значение типа Person.

Данные ссылочного типа можно описать в разделах TYPE или VAR. Описание ссылочных типов позволяет единственное исключение из общего правила, согласно которому идентификатор может быть не описан перед использованием. В данном случае допускается описание вида

```
TYPE
        PtrType = ^ BaseType;
```

даже, если тип BaseType еще не был описан. Однако BaseType должен быть описан дальше в _1 _0той же части описания типов, что и тип PtrType:

```
TYPE
        PtrType = ^ BaseType;
                BaseType = Record
                x,y : real
        End;
```

Для того, чтобы присвоить переменной ссылочного типа некоторое значение, необходимо воспользоваться унарной операцией взятия указателя. Знак этой операции - символ " @ ". Операнд - переменная любого типа, например: если имеется описание

```
VAR
    i : integer;
```

то применение этой операции к переменной i - @i дает в качестве результата значение типа "указатель на целое". ***Например***:

```
P1 := @i ;
```

В обеих частях оператора стоят конструкции одного и того же типа.
В результате такого присваивания P1 получит в качестве своего нового значения указатель на переменную i (адрес переменной i).
Операция взятия указателя допустима для любых переменных, в том числе для элементов массивов, полей записи и т.д. Например, если есть описание вида

```
VAR
    A : array [1..10] of integer;
```

то конструкция

```
@A[i]
```

имеет смысл "указателя на i - тое целое значение в массиве A" и может участвовать в присваивании

```
P1 := @A[i];
```

Ссылочные типы можно образовывать от любых типов, поэтому допустимо определение вида "указатель на указатель". Так, переменная из следующего описания

```
VAR
    PP1 : ^P;
```

в качестве своих возможных значений имеет множество указателей, ссылающихся на указатели, которые, в свою очередь, ссылаются на целые значения.

13.2. Пустой указатель

Среди всех возможных указателей в языке Паскаль выделяется один специальный указатель, который "никуда не указывает". В этом случае в адресном пространстве оперативной памяти выделяется адрес, в котором заведомо не может быть размещена никакая переменная. На это место в памяти и ссылается пустой или "нулевой" указатель, который обозначается служебным словом NIL.
Указатель NIL считается константой, совместимой с любым ссылочным типом, т.е. это значение можно присваивать любому ссылочному типу.

13.3. Основные операции

Над значениями ссылочного типа допускается две операции сравнения:

а) равенство - " = ";
б) неравенство " <> ".

Эти операции проверяют, ссылаются ли два указателя на одно и то же место в памяти. ***Например***:

```
. . .
sign := p1 = p2;  { sign является переменной ссылочного типа }
```

или

```
if p1 <> nil then
. . .
```

13.4. Доступ к переменной по указателю

Доступ к переменной ссылочного типа может быть осуществлен двумя способами:
1) прямой доступ - с помощью идентификатора;
2) косвенный доступ - путем использования адреса переменной, который хранится в указателе. Например:

```
p1 := @i;
```

Для реализации 1-го способа достаточно использовать оператор присваивания, например:

```
i := i+2; { здесь считывается текущее значение i и увеличивается на 2}
```

Для реализации косвенного доступа к переменной через указатель (ссылку) используется операция, называемая разыменованием. В этом случае необходимо после переменной - указателя поставить символ " ^ ". Например, запись

p1^

является переменной, на которую ссылается указатель p1.

Следовательно, операторы i := i+2 и p1^ := p1^+2 считаются эквивалентными.

Разыменование по определению имеет тип, совпадающий с базовым типом переменной - указателя, в частности, p1^ является переменной целого типа. Разыменование допускается для любых ссылочных типов. Например, для переменной OneMan допускаются конструкции:

```
OneMan^.Name := 'Иван';
if OneMan^.speciality = 22.01 then ...
```

В случае "указатель на указатель" возможно многократное разыменование: например, для переменной pp1 допустима конструкция

pp1^^,

которая имеет статус целой переменной.

Разыменование считается некорректным, если ссылочная переменная имеет значение NIL. В этом случае не существует переменной, на которую ссылается указатель, поэтому операторы:

```
p1 := nil;
p1^ := 2;
```

являются недопустимыми, хотя и не приводят к аварийному их прекращению.

13.5. Динамические переменные. Динамическая память

Переменные, для которых память распределяется автоматически, называются статическими. Статические переменные описываются в разделах описаний в каком-либо блоке Паскаль - программы и обозначаются идентификатором.

Переменные, созданием и уничтожением которых может явно управлять программист, называются динамическими переменными. Динамические переменные, количество которых и место расположения в памяти заранее не известно, невозможно обозначить идентификатором. Поэтому единственным средством доступа к динамическим переменным является указатель, который указывает текущий адрес ячейки памяти.

Следует отметить, что для распределения памяти под статические переменные отводится специальный сегмент оперативной памяти (сегмент данных). Аналогично, образование динамической переменной реализуется в другой области оперативной памяти, которая существует отдельно от сегмента данных и называется кучей или динамической областью памяти.

13.5.1. Основные действия над динамическими переменными

1. Процедура NEW предназначена для создания динамических переменных и имеет вид: New (< имя ссылочной переменной >);
2. Процедура DISPOSE удаляет переменную, на которую указывает значение ссылочной переменной, и имеет вид: Dispose (< имя ссылочной переменной >);

Например:

```
Var
    P : ^ Person;
Begin
    New (P); {В куче отводится область памяти, достаточной для хранения записи типа
    Person}
    < Действия с указателем P >
    Dispose (P) {Область памяти, занимаемая удаляемой переменной, возвращается в
    кучу и может быть использована для других переменных}
End.
```

3. Процедура MARK присваивает ссылочной переменной значение, указывающее на вершину кучи, и имеет следующий вид: Mark (< имя ссылочной переменной >);
4. Процедура RELEASE удаляет из кучи переменную, указанную указателем, и все переменные, следующие за ней. Release (< имя ссылочной переменной >);
5. Процедура GETMEM резервирует в куче область оперативной памяти: GetMem (< имя ссылочной переменной, размер памяти в байтах >). Процедура FREEMEM возвращает в кучу область оперативной памяти, указанную переменной ссылочного типа: FreeMem (< имя ссылочной памяти, размер памяти в байтах >);
6. Размер области, возвращаемой с помощью процедуры FREEMEM, должен быть таким же, как и у области, определенной с помощью GetMem. Процедуры NEW - DISPOSE, MARK - RELEASE и GETMEM - FREEMEM образуют три механизма управления памятью кучи. Первые два механизма используются для однотипных данных, а их отличие можно пояснить с помощью следующей схемы (рис.2а. и 2б.)

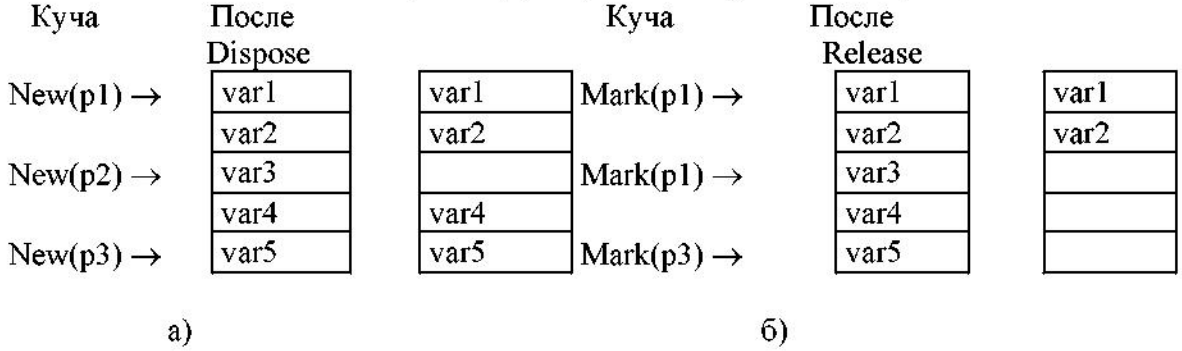


Рис.1. Структура памяти типа "куча".

Процедуры GetMem - FreeMem используются для данных любого типа. В программе следует пользоваться одним из этих механизмов в зависимости от постановки задачи.

7. Функция MAXAVAIL - эта функция определяет размер наибольшей цельной области в куче. Ее результатом является величина целого типа.

13.6. Динамические списковые структуры.

Часто ссылочные типы данных используются для организации динамических списковых структур. К ним относятся очереди, списки, деревья и подобные структуры.

Для задания списковых структур необходимо определить объект комбинированного типа, в состав которого входит ссылка на объект данного типа. В языке Паскаль разрешено определять ссылки на объекты до описания объектов, например:

```
Type
    ukaz  = ^ pole;
    dannye = < тип >;
    pole  = record
            c : ukaz;
            d : dannye
              end;
```

13.6.1. Однонаправленные списки.

Определив ссылочный тип, можно построить связанный однонаправленный список (рис.3).

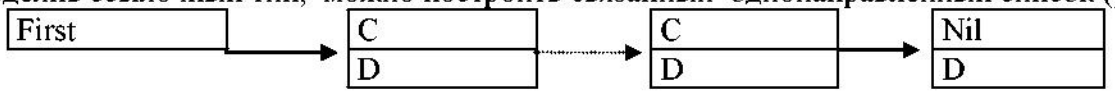


Рис. 2. Однонаправленный список.

Указатель First содержит ссылку на первый элемент списка. Каждый элемент списка содержит поле C - ссылку на следующий элемент - и поле данных D, которое, в свою очередь, может быть достаточно сложной структурой. Поле ссылки последнего элемента списка содержит пустую ссылку - NIL. Двигаясь по указателям, можно последовательно просмотреть весь список. Если из списка необходимо удалить любой элемент, кроме первого, то для этого достаточно изменить поле ссылки предыдущего элемента.

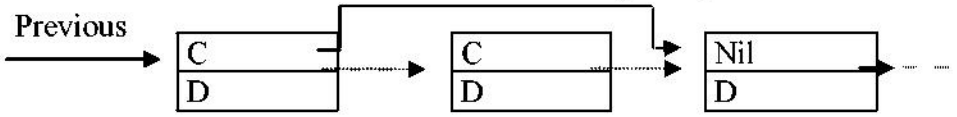


Рис. 3. Механизм исключения элемента из списка.

На рис.4 показано исключение из списка элемента F. Для этого адрес элемента B, который хранился в поле ссылки элемента F, записывается в поле ссылки элемента A, после чего элемент A будет указывать на B, а элемент F станет недоступным, так как на него никто не ссылается.

Пусть Previous - переменная ссылочного типа, содержащая ссылку на элемент, предшествующий удаляемому. Тогда, чтобы осуществить операцию исключения, достаточно выполнить следующий оператор присваивания:

Previous^.C := Previous^.C^.C

Правая часть оператора присваивания читается так: идти по ссылке, хранящейся в Previous, взять ссылку из поля C удаляемого элемента. Полученное ссылочное значение записывается в поле C элемента, предшествующего удаляемому.

Чтобы вставить элемент в произвольное место связанного списка, кроме начала, также необходимо изменить только значения указателей. Сами элементы списка при этом не перемещаются. На рис.5. показано включение в связанный список элемента X.

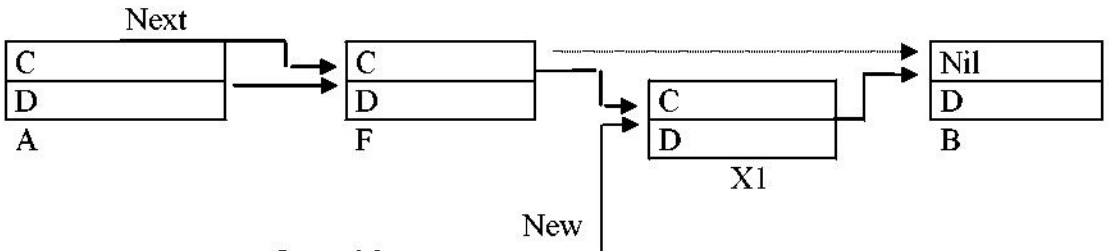


Рис. 4. Механизм вставки нового элемента в списке.

Пусть ссылочная переменная Next указывает на элемент, после которого необходимо вставить в список элемент X. На элемент X указывает ссылочная переменная New. Операция вставки реализуется двумя операторами присваивания:

```
New^.C  := Next^.C;
Next^.C := New;
```

Первый оператор заносит в поле ссылки вставляемого элемента ссылочное значение, указывающее на элемент B и содержащееся ранее в F. Второй оператор записывает в поле ссылки элемента F ссылку на вставляемый элемент.

Для добавления в конец списка элемента X достаточно найти последний элемент списка (поле ссылки имеет значение NIL) и переслать в его поле ссылки указатель на добавляемый элемент. При этом поле ссылки добавляемого элемента должно содержать NIL. Эти действия реализуются операторами

```
New^.C  := Nil;
Next^.C := New;
```

Обычно при построении связанных списков вводят ссылочную переменную, указывающую на начало списка, на его первый элемент. Поэтому при удалении первого элемента или при вставке нового элемента в начало списка необходимо переопределять не указатель

предыдущего элемента, как в рассмотренных выше примерах, а значение начального указателя.

Операция удаления реализуется одним оператором

```
First := First^.C;
```

а операция вставки - двумя операторами:

```
New^.C := First;
First := New;
```

Пример:

```
Program spisok;
Type
    name   = string[10];
    ukaz   = ^element;
    element = record
                uk : ukaz;
                dan: name
              end;
Var
    kniga          : name;
    nach,tek,ob,obpr : ukaz;
    i              : integer;
Begin  { формирование 1-го элемента списка }
    New(nach);
    ReadLn;
    WriteLn('введите первое название');
    Read(kniga);
    nach^.uk := nil;
    nach^.dan := kniga;
    While kniga <> 'AAAAA' do  { формирование списка }
     Begin
      tek := nach;
      New(ob);
      WriteLn(' Введите очередное название ');
      Read(kniga);
      While (tek<>nil) and (tek^.dan < kniga) do
           { Поиск подходящего места }
       Begin
        obpr := tek;   { ссылка на предыдущий элемент }
        tek := tek^.uk  { переход к следующему элементу }
       End;
     { вставка нового элемента }
       ob^.uk := tek;
       ob^.dan := kniga;
       If tek = nach then
        nach := ob
       Else
        obpr^.uk := ob
    End;  { конец формирования списка }
    WriteLn(' каталог учебников:');  { вывод на экран дисплея
упорядоченного каталога книг }
    tek := nach;
    While tek^.uk <> nil do
     Begin
```

```
        WriteLn (tek^.dan);
        tek := tek^.uk
      End
End.
```

Пояснения к программе:

Программа SPISOK вводит с клавиатуры названия учебников и строит из них связанный список, упорядоченный по алфавиту. По окончании формирования каталог книг выводится на экран дисплея.

Выводы:

Использование списковых структур при решении подобных задач имеет определенные преимущества: исключает предварительное резервирование памяти, дает возможность сортировать (упорядочивать) список одновременно с вводом, удалять и вставлять новые элементы. Такой упорядоченный список можно хранить во внешнем файле.

Контрольные вопросы

1. Каково назначение переменных ссылочного типа?
2. Как распределяется память под переменные ссылочного типа?
3. Каково назначение процедур NEW и DISPOSE, MARK и RELEASE?
4. В чем состоит отличие механизмов работы этих процедур?
5. Дайте определение динамической переменной?
6. Чем отличается динамическая переменная от статической?
7. Что понимается в языке Паскаль под кучей?
8. Какие операции выполняются над переменными ссылочного типа?
9. Как организуется однонаправленный список ?
10. Каким образом можно исключить элемент из списка?
11. Как организуется вставка элемента в список?
12. Каким образом можно добавить элемент в конец списка?

Задание к работе

Выполнить индивидуальное задание.

Методические указания

1. Необходимо ознакомиться с примером программы SPISOK.
2. При решении задачи использовать ссылочный тип данных.
3. Разработать алгоритм решения задачи.
4. Написать программу.
5. Отладить программу.
6. Разработать тестовые наборы данных на проверку полноты функционирования программы.
7. Оформить отчет и написать выводы по эффективности использования ссылочных типов данных.

Содержание отчета

1. Титульный лист.
2. Словесная постановка задачи.
3. Графический или текстуальный алгоритм решения задачи.
4. Листинг программы.
5. Контрольный тест и результаты тестирования программы.
6. Ответы на контрольные вопросы.

Варианты индивидуальных заданий

1. Организовать однонаправленный список всех простых чисел, меньших n. Удалить из списка элементы, значения которых лежат в диапазоне от m1 до m2. Результаты обработки списка вывести на экран.

2. Дано натуральное число n (n≥2). Найти все меньшие n простые числа, используя решето Эратосфена. Решетом Эратосфена называют следующий способ. Выпишем подряд все целые числа от 2 до n. Первое простое число 2. Подчеркнем его, а все большие числа, кратные 2, зачеркнем. Первое из оставшихся чисел 3. Подчеркнем его как простое, а все большие числа, кратные, 3, зачеркнем. Первое число из оставшихся теперь 5, так как 4 уже зачеркнуто. Подчеркнем его как простое, а все большие числа, кратные 5 зачеркнем и т.д.: 2, 3, 4, 5, 6, 7, 8, 9, 10, ... Исходную последовательность чисел организовать в виде однонаправленного списка. Удаление производить внутри этого списка, не используя дополнительные списки.
3. Дана действительная матрица размера n×m; организовать однонаправленный список строк матрицы, упорядоченных:
    a) по неубыванию значений первых элементов строк.
    b) по невозрастанию значений наибольших элементов строк.
4. Даны действительные числа a_1, ..., a_n, p, натуральное число k, так что ($a_1 \le a_2 \le ... \le a_n$, $k \le n$). Удалить из последовательности a_1, ..., a_n элемент с номером k (то есть a_k) и вставить элемент, равный p, так чтобы не нарушилась упорядоченность. Последовательность a_1, ..., a_n представить в виде однонаправленного списка.
5. Дана действительная матрица размера n×m; организовать однонаправленный список строк матрицы, упорядоченных:
    a) по невозрастанию сумм элементов строк.
    b) по неубыванию значений наименьших элементов строк.
6. Таблица выигрышей денежной лотереи представлена массивом выигрышных номеров a_1, ..., a_n и массивом выигрышей в рублях p_1, ..., p_n (p_i - это выигрыши, выпавший на номер a_i). Разместите эту таблицу в динамической области памяти в виде однонаправленного списка. Организовать удаление элементов списка по мере получения выигрышей. Результаты обработки выводить на экран.
7. Дано натуральное число n. Среди чисел 1, ..., n найти все такие, запись которых совпадает с последними цифрами записи их квадрата, например $6^2=36$, $25^2=625$ и т.д. Представить их в виде однонаправленного списка, элементами которого являются само число и его квадрат.
8. Пусть дан массив a_1, ..., a_n. Требуется переставить a_1, ..., a_n , так чтобы в начале в массиве шла группа элементов больших того элемента, который в исходном массиве располагался на первом месте, затем - сам этот элемент, потом - группа элементов, меньших или равных ему. Использовать однонаправленный список.
9. Назовем натуральное число палиндромом, если его запись читается одинаково с начала и с конца, например 4884, 393, 1. Найти все меньшие 100 натуральные числа, которые при возведении в квадрат дают палиндром. При решении задачи используйте двунаправленный список.
10. Натуральное число из n цифр является числом Армстронга, если сумма его цифр, возведенных в n-ую степень равна самому числу, например, $153=1^3+5^3+3^3$. Получить все числа Армстронга, состоящие из двух, трех и четырех цифр, организовать соответственно 3 однонаправленных списка.

ЛАБОРАТОРНАЯ РАБОТА N 14

Тема: "Введение в машинную графику"

Цель работы

Овладение начальными навыками работы с графическим режимом экрана, используя модуль GRAPH.

Краткие сведения из теории

Стандартный модуль GRAPH содержит библиотеку из более чем 50 графических программ. В нем поддерживается несколько видов закрашивания, типов линий и шрифтов, размер которых можно изменять.

Для переключения экрана в графический режим необходимо выполнить процедуру:

InitGraph (var Graphdriver : integer; var GraphMode : integer; PathToDriver : string);

По этой процедуре выполняются:

- поиск в каталоге PathToDriver файла драйвера графической карты с номером GraphDriver;
- загрузка драйвера в оперативную память;
- установка указанного графического режима GraphMode.

Если имя каталога не указано (параметр PathToDriver представляет собой пустую строку ''), то файл ищется в текущем каталоге.

Если GraphDriver = 0, то происходит автоматическое определение графической среды, а переменным GraphDriver и GraphMode присваиваются величины, определяющие номер драйвера и номер режима графического экрана.

Номера карт (графических драйверов) и режимов могут быть выражены с помощью следующих обозначений:

Detect=0 -автоматическое распознавание графической карты;
CGA=1 - карта CGA;
MCGA=2 - карта MCGA;
EGA=3 - карта EGA;
EGA64=4 - карта EGA64;
EGAMono=5 - карта EGAMono;
Reserved=6 - зарезервировано (не используется);
HercMono=7 - карта Hercules;
ATT400=8 - карта ATT400;
VGA=9 - карта VGA;
PC3270=10 - карта PC3270,

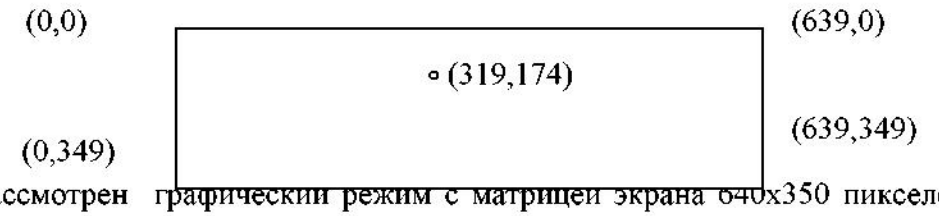
а также используются некоторые символы режима:

EGALo=0 - 620x200 пикселей, 16 цветов, 4 страницы;
EGAHi=1 - 640x350 пикселей, 16 цветов, 2 страницы.

Пиксель - это единичный элемент экрана. Количеством пикселей по горизонтали и вертикали определяется размер графического экрана. Синонимом пикселя является понятие "точка".

Для перехода из графического режима в текстовый используется процедура CloseGraph без параметров, которая устанавливает текстовый режим и очищает память, автоматически выделенную программе. Это выполняется подсистемой управления графикой.

Для того, чтобы с помощью операторов, строящих изображение, можно было обращаться к заданным точкам экрана, необходимо иметь возможность однозначно их идентифицировать. Положение точки (пикселя) задается с помощью двух координат X, Y, где X - номер столбца, Y - номер строки. По соглашению верхний левый угол экрана имеет координаты (0,0). Координата X увеличивается при перемещении вправо, а координата Y - при перемещении вниз. Таким образом, координаты каждого из четырех углов и конкретной точки, например, середины экрана, могут выглядеть следующим образом:



Здесь рассмотрен графический режим с матрицей экрана 640x350 пикселей. В некоторых других режимах эта матрица может быть иной, например, 320x200 или 640x200 и так далее.
Определение координат правого нижнего угла экрана выполняется по функциям: GetMaxX, GetMaxY.

Синтаксис:

GetMaxX : integer;
GetMaxY : integer.

Графические объекты могут быть черно-белыми или цветными. Черно-белые рисунки выполняются обычно в системах с графическими картами, не учитывающими цвет, например, с картой Hercules. Если возможен вывод в цвете (например, карта EGA или CGA), то при установке графического режима можно определить его параметры и тем самым доступную цветовую гамму. Цвета гаммы (она отсутствует для карты Hercules) пронумерованы от 0 до GetMaxColor. Цвета могут быть фиксированными (для карты CGA), либо произвольно изменяемыми (для карты EGA и VGA). Если цвета могут изменяться, то присвоение выбранного цвета позиции цветовой гаммы производится с помощью процедуры SetPalette.

Синтаксис:

SetPalette (ColorNum, Color : word);

Идентификатор цвета Color присваивается позиции ColorNum цветовой гаммы. Изменение цвета на экране будет обнаружено после выполнения процедуры SetPalette. Идентификаторы цветов могут быть выражены следующим образом:

Black=0	(черный);	DarkGray=8	(темно-серый);
Blue=1	(синий);	LightBlue=9	(светло-синий);
Green=2	(зеленый);	LightGreen=10	(светло-зелен.);
Cyan=3	(голубой);	LightCyan=11	(светло-голуб.);
Red=4	(красный);	LightRed=12	(светло-красн.);
Magenta=5	(лиловый);	LightMagenta=13	(светло-лилов.);
Brown=6	(коричневый);	Yellow=14	(желтый);
LightGray=7	(светло-серый);	White=15	(белый).

Выбор номера цвета для изображения объектов обеспечивает процедура SetColor.

Синтаксис:

SetColor(N : word);

При выполнении этой процедуры объекты будут рисоваться цветом, связанным аппаратно или программно с позицией N цветовой гаммы. Программное присвоение этой позиции другого цвета немедленно изменяет цвет рассматриваемых объектов.

Процедура SetBkColor изменяет цвет фона на такой цвет, который соответствует позиции N текущей цветовой гаммы.

Синтаксис:

SetBkColor(N : word).

14.1. Рисование графических примитивов

Рисование точки (пикселя) с координатами (X, Y) цветом с номером Color выполняется процедурой

PutPixel(X, Y: integer; Color: word).

Рисование отрезков прямых линий можно выполнить одной из следующих трех процедур:

Line(X1, Y1: integer; X2, Y2: integer) - рисование отрезка прямой линии, соединяющего точки с координатами (X1,Y1) и (X2,Y2);

LineRel(dX, dY: integer) - рисование отрезка прямой линии от текущего положения графического курсора на расстояние dX по горизонтали и dY по вертикали;

LineTo(X, Y: integer) - рисование отрезка прямой линии от текущего положения графического курсора к точке, имеющей координаты (X, Y).

После установления графического режима по умолчанию графическим окном будет весь экран. Левый верхний угол графического экрана имеет координаты (0,0), а правый нижний (GetMaxX, GetMaxY).

Графическое окно можно установить с помощью процедуры:

SetViewPort(X1, Y1: integer; X2, Y2: integer; Clip :boolean);

Процедура задает графическое окно в виде прямоугольника с координатами противоположных углов (X1, Y1) и (X2, Y2). CLIP определяет, должны ли обрезаться выходящие за пределы окна части рисунка: если этот параметр задается константой ClipOff (или False), то не обрезать, если же константой ClipOn (или True), то обрезать.

Все графические операции выполняются в текущем графическом окне, а координаты графического курсора отсчитываются всегда относительно левого верхнего угла окна. Графический курсор невидим, но его текущие координаты могут быть определены с помощью процедур GetX и GetY.

Синтаксис:

GetX : integer; GetY : integer.

Для очистки графического экрана используется процедура без параметров ClearDevice.

После очистки экрана графический курсор будет установлен в верхнем левом углу текущего графического окна.

Процедура без параметров ClearViewPort очищает текущее графическое окно и заполняет его цветом первой позиции цветовой гаммы.

Пример графического построения:

Пусть требуется на желтом фоне экрана размером 640x350 построить графическое окно симметрично относительно центра экрана с размерами в два раза меньше размеров экрана, и в этом окне построить синюю диагональную прямую и красную точку в левом нижнем углу окна.

Программа:

```
Program Gra1;
Uses Graph;   {вызов модуля Graph}
Var
        Driver, Mode : integer;
Begin
    Driver := Detect;              {автоматическое распознавание карты}
    InitGraph(Driver,Mode,");      {установка графического режима}
    SetBkColor(14);                {установка желтого цвета фона}
    SetViePort(160,88,479,262,ClipOn);{выделение графического окна размером 320x175}
    SetColor(1);                   {выбор синего цвета}
    LineRel(319,174);              {рисование диагонали}
    PutPixel(0,174,4);             { отображение красной точки }
    Readln;
    CloseGraph                     { возврат в текстовый режим }
end.
```

Здесь предложение READLN позволяет задержать на экране окно вывода с графическими изображениями до тех пор, пока не будет нажата клавиша <Enter> .

Рисование окружности с центром в точке (X, Y) и радиусом Radius выполняется процедурой Circle(X, Y: integer; Radius: word);

Рисование дуги окружности с центром в точке (X, Y), радиусом Radius, углом начала StAngle и углом конца EndAngle, выполняется процедурой:

Arc(X, Y : integer; StAngle, EndAngle : word; Radius : word).

Углы StAngle и EndAngle выражаются в радианах. Рисование происходит против движения часовой стрелки.

Рисование дуги эллипса с центром в точке (X, Y), полуосями XRadius и YRadius, углами начала StAngle и конца EndAngle выполняется процедурой:

Ellipse(X, Y: integer; StAngle, EndAngle : word; Radius, YRadius : word).

Углы выражаются в радианах. Рисование осуществляется против часовой стрелки. Для StAngle = 0 и EndAngle = 2 будет нарисован полный эллипс.

Изображение закрашенного сектора круга выполняется процедурой:

PieSlice(X, Y : integer; StAngle, EndAngle : word; Radius : word).

Параметры этой процедуры имеют тот же смысл, что и в процедуре Arc, рассмотренной выше. Шаблон закраски задается процедурой SetFillStyle (см. ниже).

Пусть, например, требуется построить сектор круга с центром в точке (319, 174), радиусом в 100 точек и представляющим собой левую нижнюю четверть круга. Для этого достаточно выполнить процедуру PieSlice со следующими параметрами:

PieSlice (319,174,180,270,100).

Процедурой Rectangle(X1, Y1: integer; X2, Y2: integer) будет построен прямоугольник, противоположные вершины которого имеют координаты (X1, Y1) и (X2, Y2).

Процедура Bar(X1, Y1: integer; X2, Y2: integer) в отличие от предыдущей процедуры строит закрашенный прямоугольник.

Допустим, необходимо построить закрашенный в синий цвет прямоугольник с горизонтальной стороной в 200 точек, с вертикальной в 100 точек и с центром в точке (319, 174). Два следующих предложения решают эту задачу:

```
SetColor (1);              { выбор синего цвета }
Bar(219,124, 419,224);     {вычерчивание закрашенного прямоугольника}
```

Выбор типа линии, которой будет нарисован объект, можно выполнить процедурой:

SetLineStyle(LineStyle: word; Pattern: word; ThickNess:word);

Выбор типа линий выполняется на основании LineStyle и толщины линии по ThickNess. Если LineStyle = UserBitLn, то тип линии выбирается по системе битов Pattern. Этот параметр рассмотрен при описании процедуры "заливки". Тип линии может быть выражен следующим образом:

```
SolidLn  = 0      - сплошная линия;
DottedLn  = 1     - пунктирная линия (из точек);
CenterLn  = 2     - осевая линия (из точек и тире);
DashedLn  = 3     - штриховая линия;
UserBitLn = 4     - линия, определяемая пользователем.
```

Толщина линий может быть выражена следующим образом:

```
NormWidth  = 1    - тонкая линия;
ThickWidth = 3    - толстая линия .
```

Пример.

Построить прямоугольник точечной тонкой линией зеленого цвета, противоположные углы которого имеют координаты (10,10) и (319,174).

Программа:

```
Program Rect;
Uses Graph;
Var
    dr, mode : integer;
Begin
    { инициализация граф. экрана }
    dr := detect;
    InitGraph (dr, mode,'');
    { Установка тонкой (1) линии точечного типа }
```

```
SetLineStyle(DottedLn, 0, 1);
SetColor(2);
{ вычерчивание прямоугольника }
Rectangle(10,10, 319,174);
Readln
End.
```

Заполнять области стандартными шаблонами позволяет процедура:

SetFillStyle(Pattern : word; Color : word),

где Pattern задает номер шаблона, а Color - номер цвета шаблона.

Номера заполняющих шаблонов могут быть выражены следующим образом:

EmptyFill	= 0	- заполнение цветом фона;
SolidFill	= 1	- сплошное заполнение;
LineFill	= 2	- заполнение толстыми горизонтальными линиями;
LtSlashFill	= 3	- заполнение наклонными линиями (правый наклон);
SlashFill	= 4	- заполнение толстыми наклонными линиями;
BkSlashFill	= 5	- заполнение толстыми косыми линиями (левый наклон);
LtBkSlashFill	= 6	- заполнение косыми линиями;
HatchFill	= 7	- заполнение вертикальной сеткой;
HatchFill	= 8	- заполнение наклонной сеткой;
InterleaveFill	= 9	- заполнение переплетенными линиями;
WideDotFill	= 10	- заполнение точками;
CloseDotFill	= 11	- плотное заполнение точками.

Например, чтобы заполнить вертикальной сеткой красного цвета прямоугольник, рассмотренный в предыдущем примере, необходимо использовать следующие предложения:

```
SetFillStyle(7, 4);
Bar(10,10, 319,174);
```

Заполнение заданным шаблоном области, охватывающей точку с координатами (X, Y), ограниченной линией, номер цвета которой определен Border, выполняется процедурой

FloodFill(X, Y: integer; Border: word).

Шаблон и цвет заполнения области могут быть определены с помощью процедуры SetFillStyle .

Например, чтобы заполнить наклонными линиями коричневого цвета круг желтого цвета с центром в точке (319, 174) и радиусом 100, необходимо использовать операторы:

```
SetFillStyle(3, 6);
SetColor(14);
Circle(319, 174, 100);
FloodFill (310, 170, 14);
```

14.2. Контроль за выполнением графических операций

При выполнении графических процедур возможны следующие ошибочные ситуации:

1. Параметры процедуры не нарушают требований синтаксиса, но подобраны неправильно при этом выполнение процедуры не вызовет никаких изменений.
2. Параметры подобраны правильно, а процедура выполняется неправильно.
3. Определение причин неправильного выполнения графических операций остается за разработчиком программы. Эту задачу упрощает функция GraphResult, позволяющая определить результат завершения графической операции.

Синтаксис:

GraphResult : integer;

Если операция закончилась успешно, функция возвращает 0, в противном случае отрицательное значение, идентифицирующее причину неудачи. Коды ошибок:

grOk -0 -нормальное выполнение графической операции;

grNoInitGraph	-1	-графический режим не установлен;
grNotDetected	-2	-нет графической карты;
grFileNotFound	-3	-файл драйвера устройства не найден;
grInvalidDriver	-4	-неподходящий файл драйвера устройства;
grNoloadMem	-5	-нет памяти для загрузки драйвера;
grNoScanMem	-6	-нет памяти для заполнения области методом сканирования;
grNoFloodMem	-7	-нет памяти для заполнения области методом заливки;
grFontNotFound	-8	-файл со шрифтами не найден;
grNoFontMem	-9	-нет памяти для загрузки шрифта;
grInvalidMode	-10	-неправильный графический режим;
grError	-11	-другие ошибки;
grIOError	-12	-ошибки операции ввода - вывода;
grInvalidFont	-13	-ошибочный стиль шрифта;
grInvalidFontNum	-14	-ошибочный номер стиля шрифта;
grInvalidDeviceNum	-15	-ошибочный номер устройства.

Поскольку двукратный вызов функции GraphResult (без выполнения между вызовами графической операции) приводит к тому, что вторым результатом всегда будет 0, рекомендуется назначить первый результат промежуточной переменной.

Для обработки "неуспешных" графических операций следует создавать процедуры анализа и выхода из ситуаций, вызвавших неуспех.

Контрольные вопросы

1. Какой модуль обеспечивает графику на ЭВМ?
2. Как инициализировать графический режим, если неизвестно, какой графический драйвер используется на данной машине?
3. Всегда ли необходимо явно задавать все параметры процедуры InitGraph?
4. Что такое пиксел и каковы его характеристики?
5. С помощью каких функций можно узнать размеры экрана и количество цветов в цветовой гамме (палитре) данного графического режима?
6. Могут ли на экране высвечиваться одновременно отрезки прямых линий разных цветов?
7. Где будет графический курсор после выполнения одной из процедур PutPixel, Line, LineRel, LineTo и как это проверить практически?
8. Всегда ли высвечиваются рисуемые точки после определения графического окна?
9. Какими процедурами можно нарисовать окружность?
10. Как отсчитываются углы в тех процедурах, где они используются в качестве параметров?
11. Почему в процедурах Rectangle и Bar указываются координаты лишь двух вершин прямоугольника?
12. Какие из рассмотренных процедур рисуют линии, а какие - области?
13. Влияет ли процедура SetLineStyle на результаты работы процедур, которые рисуют области?
14. Влияет ли процедура SetFillStyle на результаты работы процедур, рисующих линии?
15. К каким результатам приводит обращение к процедуре FloodFill, когда первые два ее параметра задают точку, лежащую вне области, охватываемой линией?
16. Что произойдет, если в процедуре FloodFill линия , охватывающая заданную точку, окажется не замкнутой?
17. Можно ли результаты процедур Bar, PieSlice получать с использованием других процедур?

Задание к работе

Построить круговую диаграмму, отображающую процентное соотношение отличников, хорошистов и прочих. Для заливки секторов использовать различные шаблоны и цвета.

Построить столбиковую диаграмму, отображающую рост цен на бензин.

Построить график функции, заданной в индивидуальном задании.

Методические указания

1. Разработать алгоритмы и программы для решения задач заданий.
2. Скомпилировать программы.
3. Составить контрольные тесты и протестировать программы.
4. Составить отчет и представить его к защите.

Для выполнения первого задания необходимо придерживаться следующей схемы:

а) подсчитать общее количество элементов, входящих в систему, например:

$$К = КО + КХ + КП,$$

где К - количество элементов в системе (количество всех студентов);

КО - количество отличников;

КХ - количество хорошистов;

КП - количество прочих;

б) вычислить процентное отношение группы элементов в системе, например, процентное отношение отличников в системе равно:

$$\%О = \frac{КО * 100\%}{К}$$

в) вычислить, сколько радиан составляет та или иная группа элементов, если вся система составляет 2π радиан, например:

$$Кол.рад.отл. = \frac{2 * \pi * \%О}{100\%}$$

г) используя графические процедуры и функции вывести на экране круговую диаграмму.

Для выполнения второго задания необходимо:

а) определить максимальное значение элементов, входящих в систему, в нашем примере - максимальную стоимость (Max) бензина за рассматриваемый период времени;

б) рассчитать высоту HMax в пикселах самого высокого столбика Max, используя функцию GetMaxY;

в) вычислить количество пикселей KPixH, приходящихся на единицу стоимости;

г) вычислить высоту в пикселях каждого элемента системы;

д) рассчитать ширину одного столбика, учитывая при этом межстолбиковое расстояние и используя функцию GetMaxX;

е) используя графические процедуры и функции вывести столбиковую диаграмму на экран.

Для выполнения третьего задания используйте пример программы, приведенный ниже.

Пример. Построение графика функции Y = Sqr(X).

При отображении графика функции на экране необходимо выполнить переход от локальной системы координат в систему координат экрана, а также во избежание помех рассчитать граничные значения X, при которых значения Y начинают выходить за пределы экрана.

Программа:

```
Program GraficFunction;
Uses Graph;
Var
        grDriver : Integer;
        grMode   : Integer;
        ErrCode  : Integer;
        X, Y, X1, Y1, CX, CY, XG: Integer;
Begin
{*** инициализация графического режима экрана ***}
grDriver := Detect;
```

```
InitGraph(grDriver, grMode,'');
ErrCode := GraphResult;
If ErrCode=grOk then {инициализация графического экрана прошла
                      успешно}
      begin  {*** Построение осей координат ***}
             CX := Round(GetMaxX / 2); {для настройки на координаты любого}
             CY := Round(GetMaxY / 2); {экрана используем функции GetMaxX и GetMaxY}
                                       {(CX, CY) - центр сист. коорд-т}
             Line(0, CY, GetMaxX, CY); {вычерчивание оси ординат}
             Line(CX, 0, CX, GetMaxY); {        - " -          абцисс}
             {*** Построение графика функции Y = Sqr(X)***}
             XG:=Round(Sqrt(20*(GetMaxY-CY)));{определение ширины параболы}
             for X:= -XG  to XG do
                    begin
                           X1 := X + CX; {определение текущей координаты X экрана}
                           Y  := Sqr(X); {определение ординаты функции}
                           Y1 := GetMaxY - Round(( Y / 20 + CY)); {преобразование
                           текущей ординаты функции в текущую координату Y экрана}
                           Circle(X1, Y1, 2) { точка, инцидентная параболе, вычерчивается
                           в виде окружности радиусом в два пикселя}
                    end;
             {*** Вывод на экран пояснительного сообщения ***}
             SetTextStyle(0, 0, 2);
             OutTextXY(180, 350, 'График функции Y = Sqr(X)');
             ReadLn;
             CloseGraph;
      end
else  { возникла ошибка при инициации графического экрана}
      WriteLn('Graphics    error:',GraphErrorMsg(ErrCode));    {вывод    диагностического
      сообщения}
End.
```

Содержание отчета

1. Титульный лист.
2. Словесная постановка задачи.
3. Текстуальный или графический алгоритм работы программы.
4. Листинг программы.
5. Анализ допущенных ошибок.
6. Сведения об использованных цветах и шаблонах заполнения областей.
7. Ответы на контрольные вопросы по согласованию с преподавателем.

Варианты заданий

1. $Y = A \cdot \lg(X)$;
2. $Y = A \cdot \sin(X)$;
3. $Y = A \cdot \exp(X)$;
4. $Y = A \cdot X^2 + B \cdot X + C$;
5. $Y = A \cdot \mathrm{tg}(X)$;
6. $Y = A \cdot \mathrm{arctg}(X)$;
7. $Y = A \cdot X^n + C$, $n=1$;
8. $Y = A \cdot X^n + C$, $n=2$;
9. $Y = A \cdot X^n + C$, $n=3$;
10. $Y = A \cdot X^n + C$, $n=4$,

где A, B, C - произвольные параметры.

ЛАБОРАТОРНАЯ РАБОТА N 15

Тема: "Введение в построение человеко-машинного интерфейса"

Цель работы

Научиться корректно вводить исходную информацию и отображать выходную.

Краткие сведения из теории

В практическом программировании сложились определенные принципы организации ввода и вывода данных. Эти принципы позволяют разработать систему ввода и вывода в виде дружественного диалога пользователя с компьютером.

Для создания дружественного человеко-машинного интерфейса удобно использовать стандартный модуль CRT интегрированной Среды Турбо-Паскаля.

Данный модуль предоставляет готовые к использованию процедуры и функции для работы с клавиатурой, экраном в текстовом режиме и динамиком.

15.1. Функции CRT для клавиатуры

В модуле CRT существуют две функции ReadKey и KeyPressed, предназначенные для работы с клавиатурой.

Функция ReadKey возвращает символ, введенный с клавиатуры.

Синтаксис:

```
ReadKey : Char;
```

Пример.

```
Uses Crt;
Var
    Ch : Char;
begin
    WriteLn('Нажмите клавишу');
    Ch := ReadKey;  {считали код клавиши}
    if Ch = #0 then  {нажата функциональная клавиша?}
        begin {да}
            Ch := ReadKey; {чтобы получить код функциональной клавиши,
                            необходимо повторить чтение}
            WriteLn('Вы нажали функциональную клавишу с кодом = ', Ord(Ch));
        end
    else  {нет}
        WriteLn('Вы нажали клавишу с ASCII кодом = ', Ord(Ch));
end.
```

Примечание. Все модули подключаются к программе с помощью предложения USES.

Функция KeyPressed отвечает на вопрос: "Была ли нажата какая-либо клавиша?".

Синтаксис:

```
KeyPressed: Boolean.
```

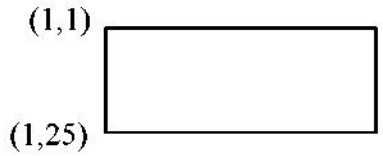
Функция возвращает значение TRUE, если клавиша была нажата, FALSE - если нет.

Пример.

```
Uses Crt;
Begin
    repeat
        {Экран заполняется до тех пор, пока не нажата клавиша}
        Write('Xx');
    until KeyPressed;
End.
```

15.2. Процедуры CRT для текстового экрана

Модуль Crt позволяет создавать окна на экране монитора. При записи в такое окно оно ведет себя точно также, как целый экран. При этом остальная часть экрана остается нетронутой. Внутри окна можно удалять и добавлять строки, при этом курсор возвращается к правому краю и при достижении курсором нижней строки текст продвигается вверх.
По умолчанию окном считается весь экран.
С каждым окном связаны координаты:



Полный экран имеет координаты:
1 <= X <= 80; 1 <= Y <= 25.

Определить окно можно с помощью процедуры Window.
Синтаксис:

```
Window(X1,Y1, X2,Y2 : word).
```

Параметры X1, Y1 - координаты верхнего левого угла окна;
X2, Y2 - координаты нижнего правого угла окна.
Если параметры заданы некорректно, то вызов Window игнорируется.
Когда определено окно, то все координаты экрана, используемые в процедурах, становятся относительными координатами этого окна.
С каждым окном связаны две переменные, объявленные в CRT:

```
WindMin, WindMax : Word.
```

Переменная WindMin определяет левый верхний угол экрана, а переменная WindMax - правый нижний. В младшем байте хранится координата X, а в старшем - координата Y. Чтобы извлечь значения этих координат, нужно использовать функции Lo и Hi. Например, Lo(WindMin) - дает координату X левого края, а Hi(WindMax) - дает координату Y нижнего края.
С координатами связаны процедуры GotoXY, WhereX, WhereY.
Процедура GotoXY - установка курсора в указанные координаты.
Синтаксис:

```
GotoXY(X, Y : byte);
```

Верхний левый угол задается координатами (1, 1). Если X, Y заданы неверно, то переход не выполняется.
Например:

```
Window(1,10, 60,20);
GotoXY(1, 1);
```

В результате выполнения этих предложений произойдет перемещение курсора в верхний левый угол созданного окна.
WhereX, WhereY - возвращают соответственно координату X или Y текущей позиции курсора относительно текущего окна.
Например:

```
Uses Crt;
begin
    Writeln('Курсор находится в позиции ', '(' ,WhereX,' , ',WhereY,' )' );
end.
```

Пример для Window.

```
Uses Crt;
Var
    x, y: Byte;
Begin
    TextBackground(Black);          { Установка черного фона }
```

```
        ClrScr;                          { очистка экрана цветом фона }
        repeat
        { Рисуются случайные окна }
                x := Succ( Random(80));  { выбор случайной  координаты X }
                y := Succ( Random(25));  { выбор случайной  координаты Y }
                Window(x, y, x + Random(10),
                y + Random(8));
                TextBackground( Random(16));  { установка случайного цвета окна}
                ClrScr;                  { очистка окна текущим цветом фона }
        until KeyPressed;
End.
```

15.3. Основные принципы ввода информации

Во-первых, при вводе информации рекомендуется выполнять ее контроль (диагностику). Это необходимо для обнаружения ошибок, допущенных при вводе. Если не разработана ПП обработки таких ошибок, то произойдет программное прерывание по вводу-выводу, и выполнение программы будет прервано.

Во-вторых, ввод любого данного должен сопровождаться комментарием. Комментарий должен быть кратким, однозначным и недвусмысленным.

В-третьих, рекомендуется на экране дисплея создавать "окна ввода" (например, в виде таблицы при вводе массива).

Рассмотрим один из видов контроля данных.

Самый простой способ проверки ввода заключается в использовании оператора IF ... THEN. Если при вводе обнаружены ошибки, выводится диагностическое сообщение, и пользователю предлагается повторить ввод.

Пример.

```
Program IOTest;
Uses Crt;
Var
        A   : array[1..10] of integer;
        i   : byte;
        Key : char;
begin
        {$I-}
        WriteLn('По окончанию ввода нажмите клавишу <Esc>');
        WriteLn('Для ввода нового значения - клавишу <Enter>');
        i:= 1;
        { цикл по вводу значений массива A }
        Repeat
                Write('Введите значение переменной A: ');
                Readln(A[i]);
                { контроль ошибки ввода-вывода }
                if IOResult <> 0 then
                        WriteLn('Введена не цифра. Повторите ввод значения.')
                else begin
                        WriteLn('A[',i:2,'] = ', A[i]);
                        i:= i + 1;
                        if i > 10 then WriteLn('Массив заполнен. Нажмите <Esc>');
                end;
                Key := Readkey
        Until Key = #27; { #27 - код клавиши <Esc> }
        {$I+}
End.
```

Примечание. {$I-} и {$I+} - директивы компилятора для отключения и включения стандартной проверки завершения операций ввода-вывода.

При создании удобного интерфейса используются "окна" CRT. Табулирование внутри "окна" осуществляется процедурой GOTOXY.

Пример применения этих процедур при вводе данных.

Словесная постановка задачи.

Ввести данные о студенте: фамилию, имя, отчество, год рождения и номер группы. Выполнить контроль правильности ввода значения данного "год рождения" на не цифру. Ввод данных организовать с помощью окон.

Текстуальная запись алгоритма.

1. Рассчитать ширину "окна". Ширина "окна" (рис.1) состоит из:
   - количества символов в "подсказке" - наименовании данного;
   - количества символов значения данного;
   - количества символов для пробелов (информация в окне должна располагаться свободно, поэтому необходимо вставлять пробелы).
2. Рассчитать высоту "окна". Высота "окна" состоит из:
   - количества вводимых данных (одно данное записывается в одной строке);
   - строки для заголовка "окна";
   - строки для диагностических сообщений (сообщение удобнее помещать внизу "окна" или в отдельном выпадающем "окне");
   - количества пустых строк для повышения визуальности восприятия "окна", например, необходима пустая строка между верхней границей окна и заголовком окна, между заголовком и первой строкой данных и т.д.
3. Для лучшего визуального восприятия создать три вложенных друг в друга "окна" с помощью процедуры WINDOW. Для каждого "окна" установить цвет фона и цвет изображаемых в нем символов.
4. Написать в первом "окне" вспомогательную информацию о работе с программой (HELP-помощь); во втором "окне" название третьего "окна"; и в третьем - тексты "подсказок " по каждому вводимому данному.
5. Процедурой GOTOXY установить курсор на позицию экрана, с которой нужно вводить значение первого данного.
6. Оператором ReadLn ввести значение данного.
7. Организовать диагностику числовых данных.
8. Если при вводе числового значения выявлена ошибка, то
   a) войти в первое "окно";
   b) установить курсор процедурой GOTOXY на строку "окна" для диагностических сообщений;
   c) предложением WRITE напечатать сообщение "Введена не цифра";
   d) предложением DELAY выполнить задержку сообщения на экране;
   e) удалить с экрана диагностическое сообщение, используя при этом процедуру GOTOXY и пустые пробелы в предложении WRITE;
   f) процедурой GOTOXY снова установить курсор на позицию экрана, с которой нужно вновь ввести значение данного;
   g) удалить с экрана неправильно введенное значение. Для этого предложением WRITE вывести пробелы, количество которых равно максимальной длине вводимого данного;
   h) вернуться в третье "окно" с помощью процедуры WINDOW;
   i) процедурой GOTOXY снова установить курсор на позицию экрана , с которой нужно вводить значение первого данного;
   j) предложением READLN ввести уточненное значение данного;
   k) передать управление на пункт 7.
9. Процедурой GOTOXY установить курсор на позицию экрана, с которой нужно вводить значение следующего данного.

10. Пункты с 6 по 9 повторяются для всех остальных данных.

Операторная запись алгоритма.

```
{ ************************************************************ }
{ *        Программа ввода и контроля данных                 * }
{ ************************************************************ }
Program Interf;
Uses Crt;
Const
 MaxNumberStud = 30;
 { массив имен вводимых данных }
 NameDat : array[1..5] of string =
        ('Фамилия','Имя','Отчество','Год рождения', 'N группы');
Type
  RecStud = record   { запись данных по студенту }
                Fam, Name, Otch : string[15];
                GodR          : word;
                NGrup         : string[10];
            end;
Var
   i      : integer;
   Key    : char;
   Flag   : boolean; { флаг корректности ввода }
   Stud   : array[1..MaxNumberStud] of RecStud;

{ ************************************************************ }
{ *        Процедура проверки значений на не цифру           * }
{ ************************************************************ }
Procedure TestData(name: string; var Flag: boolean);
begin
 if IOResult <> 0 then
 begin
  Window(10,3, 70,22); { возврат в первое окно }
  GotoXY(14,19);
  Write('Не цифра при вводе ', name, ' !') ;
  Delay(3000);        { задержка }
  GotoXY(14,19);
  { удаление диагностического сообщения с экрана }
  Write('                          ') ;
  Window(22,8, 58,18); { переход в окно ввода }
  GotoXY(23,8);
  Write('        '); { удаление с экрана старого значения}
 end
 else Flag := True;
end;
{ ************************************************************ }
{ *                  т е л о   п р о г р а м м ы           * }
{ ************************************************************ }
Begin
 { установка цвета фона и цвета символов, очистка экрана }
 TextBackGround(White);   { процедура установки цвета фона }
 TextColor(White);        { процедура установки цвета символов }
```

```
ClrScr;                          { процедура очистки экрана цветом фона}
{ ..................................................................... }
{ организация внутреннего окна и вывод в нем вспомогательной
  информации }
Window(10,3, 70,22);          { координаты левого  угла окна - (10,3)
                                координаты правого угла окна - (70,22)}
TextBackGround(Cyan);  { установка цвета фона окна - бирюзового}
ClrScr;
GotoXY(10,2);
Write('По окончанию ввода нажмите клавишу <Esc>');
GotoXY(10,3);
Write('Для ввода нового значения - клавишу <Enter>');
{ ..................................................................... }
{ организация  вложенного  окна,  используемого  для образования
  рамки и вывода названия окна }
Window(20,7, 60,19);
TextBackGround(15);          { установка белого цвета рамки }
ClrScr;
TextColor(Red);       { установка красного цвета символов }
GotoXY(15,1);
Write('Окно ввода');

{ ..................................................................... }
{ организация второго вложенного окна, предназначенного для
  ввода информации }
Window(22,8, 58,18);
TextBackGround(Cyan);
ClrScr;
TextColor(White);
{ вывод в окне наименований вводимых данных }
For i:= 1 to 5 do
begin
  GotoXY(5,2*i);     Write(i:1,'.',NameDat[i]);
  GotoXY(21,2*i);    Write(': ');
end;
{ непосредственный ввод данных }
i:= 1;
Repeat
  GotoXY(23, 2);   Readln(Stud[i].Fam);
  GotoXY(23, 4);   Readln(Stud[i].Name);
  GotoXY(23, 6);   Readln(Stud[i].Otch);
  { ввод значения года рождения и проверка его на не цифру }
  {$I-}
  Flag:= False;
  repeat
    GotoXY(23, 8); Readln(Stud[i].GodR);
    TestData(NameDat[i], Flag);              { вызов процедуры диагностики }
  until Flag;
  {$I-}
  GotoXY(23, 10); Readln(Stud[i].NGrup);
  i:= i + 1;
  { Диагностика:  исчерпано ли место в ОП, отведенное под массив  Stud }
```

```
  if i > MaxNumberStud then
  begin
    Window(10,3, 70,22);              { возврат в первое окно }
    GotoXY(12,19);
    Write('Массив заполнен. Нажмите клавишу <Esc> !') ;
    Delay(3000);        { задержка }
    GotoXY(12,19);
    { удаление с экрана диагностического сообщения }
    Write('                                   ');
  end;
  Key := Readkey
 Until Key = #27;
 { ..... }
 { очистка всего экрана }
 Window(1,1, 80,25);
 ClrScr;
End.
```

15.4. Рекомендации по отображению результатов работы программы

При выводе данных рекомендуется:

- отображать их в "окнах" в виде таблицы и по возможности сопровождать графикой;
- использовать цветовые и звуковые эффекты;

Таблица состоит из следующих частей: заголовка, подзаголовка и собственно таблицы. Заголовок содержит наименование таблицы, определяющее ее смысловое содержание. Совокупность содержаний графа (столбцов) образует подзаголовок. Слова, входящие в наименование графы, могут сокращаться. Если количество строк в таблице превышает длину стандартного листа (60 строк), то необходимо организовать постраничный вывод таблицы. При этом на любой странице кроме данных необходимо также печатать номер страницы и подзаголовок.

Возможный порядок отображения таблицы:

1. Рассчитать ширину таблицы (см. пример таблицы 1). Ширина таблицы состоит из:
   - суммы максимальных длин вводимых данных;
   - разделителей граф (их количество на единицу больше числа граф. В качестве вертикального разделителя удобно использовать символ "¦" или символ псевдографики "|" (Alt-179));
   - пустых пробелов между графами для повышения визуального восприятия таблицы.

Список студентов

Таблица 1

№ п/г	Фамилия И.О.	Год рождения
519-1а	Бадмаев Баир Александрович	1975
519-1б	Иванова Людмила Алексеевна	1975
218-2а	Павлова Раиса Адамовна	1974
218-2б	Доржиева Энгельсина Баировна	1974
239-1а	Борисов Вадим Сергеевич	1974
239-3а	Ринчинова Дарима Доржиевна	1975
239-2б	Савельев Николай Петрович	1975

Рис. 2. Вид экрана с изображением таблицы.

2. Создать окно с помощью процедуры WINDOW и установить цвета для фона "окна" и символов в нем.
3. Вывести заголовок таблицы.
4. Вывести горизонтальную линию.
5. Вывести подзаголовок таблицы, например:

№ п/г	Фамилия И.О.	Год рождения

8. Организовать цикл по выводу строки значений данных, используя форматированный вывод оператора WRITE.
9. Вывести горизонтальную линию.

Контрольные вопросы

1. Чем характеризуется текстовый экран? Укажите средства позиционирования курсора на экране.
2. Что представляет собой "окно" и какими средствами оно создается?
3. Какими следует пользоваться координатами при выводе информации в "окне": локальными - "окна" или глобальными - экрана?
4. Как осуществляется переход между вложенными или параллельными "окнами"?
5. Перечислите процедуры и функции модуля CRT и укажите их назначение.
6. В чем заключается контроль данных? Обоснуйте необходимость организации корректного ввода.
7. Что такое диагностическое сообщение? Перечислите диагностические сообщения программы Interf.
8. Что понимается под человеко-машинным интерфейсом, и в чем заключается понятие "дружественности"?

Задание к работе

Разработайте программу, имеющую удобный интерфейс для ввода и вывода информации.

Методические указания

1. Структуру данных возьмите из лабораторной работы N 10. Структура - это запись, то есть любая совокупность взаимосвязанных данных. Например, характеристики компьютера, состав компьютера или структура института являются структурами.
2. Создайте программу аналогичную программе Interf.pas дополнив ее:
   - процедурой контроля на смысловое значение данных;
   - удалением в "окне ввода" значений, введенных на предыдущем шаге.
3. Отладьте программу.
4. Согласно "возможному порядку отображения таблицы", приведенному в разделе 2.4, разработайте процедуру отображения Вашей структуры данных.
5. Вставьте эту процедуру в отлаженную программу.
6. Отладьте модифицированную программу.
7. Напишите отчет по работе и представьте его к защите.

Содержание отчета

1. Титульный лист.
2. Словесная постановка задачи.
3. Текстуальное описание алгоритма решения задачи.
4. Графическая схема алгоритма решения задачи.
5. Листинг программы.
6. Результаты тестирования программы (твердая копия структуры данных).
7. Инструкция к использованию программы.
8. Ответы на контрольные вопросы по согласованию с преподавателем.

ЛАБОРАТОРНАЯ РАБОТА N 16

Тема: "Модульное программирование"

Цель работы

1. Научиться выделять из общей задачи подзадачи с целью описания их в виде отдельных модулей.
2. Получить навыки компиляции модулей и основной программы средствами TP.
3. Понять связь объектов основной программы и модулей.

Краткие сведения из теории

При проектировании программ, предназначенных для решения сложных задач, обычно прибегают к методу пошаговой детализации, или, иначе, методу программирования сверху-вниз. Процесс пошаговой детализации начинается с составления схемы алгоритма, представляющей решение задачи в самых общих чертах. Те этапы решения, которые на схеме выражены недостаточно точно и определенно, подвергаются дальнейшему разбиению с целью уточнения первоначальной схемы на последующих шагах. Процесс детализации повторяется по отношению к новому варианту схемы алгоритма до тех пор, пока не будет достигнут такой уровень ясности решения, при котором написание программы на Паскале уже не составит труда.

Таким образом, метод программирования сверху-вниз состоит в том, что на каждом этапе частные детали решения задачи (например, касающиеся конкретного способа определения делимости одного числа на другое) откладываются до того момента, пока не выявится общая структура алгоритма.

С чего начинается проектирование программы? Прежде всего с предварительного анализа задачи. Затем с определения и описания цели программы в самых общих чертах. Первоначальное описание цели подвергается последовательному уточнению, выявляя таким образом несколько четко сформулированных задач, для которых вновь строится описание цели и выявляются новые подзадачи. Продолжая процесс детализации, можно расчленить всю проблему на все более и более простые подзадачи. Таким образом, выстраивается иерархическое дерево решения проблемы (рис. 1), ствол которой реализует главная программа, а подзадачи - подпрограммы.

При этом тело главной программы будет представлять собой совокупность предложений, в котором производятся поочередно обращения к отдельным ПП.

В терминах модульного программирования ПП подразделяются на внешние и внутренние. С внутренними ПП мы познакомились в лабораторной работе N 9. Ими являются процедуры и функции. Внешние ПП называются модулями.

Модуль - это некоторая последовательность предложений, в совокупности выполняющих вполне определенную подзадачу или несколько взаимосвязанных подзадач. Такая узко специализированная группа предложений, в принципе, может быть частью текста главной программы, что не всегда удобно, так как загромождает программу. Таким образом, модуль - это некая изолированная программная единица, обладающая следующими характеристиками:

набором предложений, описывающим ход решения задачи, при этом в набор могут входит и внутренние ПП;

параметрическими значениями, обеспечивающими настройку модуля на выполнение конкретного вычисления. Располагая обобщенной группой предложений, т.е. модулем, программа может осуществить их выполнение, задав соответствующий набор параметров;

необходимостью автономной трансляции.

Основным преимуществом использования подпрограмм является возможность тестировать и отлаживать их независимо от других ПП. Отлаженная ПП может рассматриваться как не требующий проверки элементарный шаг программы. Например, если

главная программа вызывает ПП, то мы можем каждую из них тестировать и отлаживать отдельно до тех пор, пока не убедимся в том, что она правильно работает, а затем объединить их в одно целое.

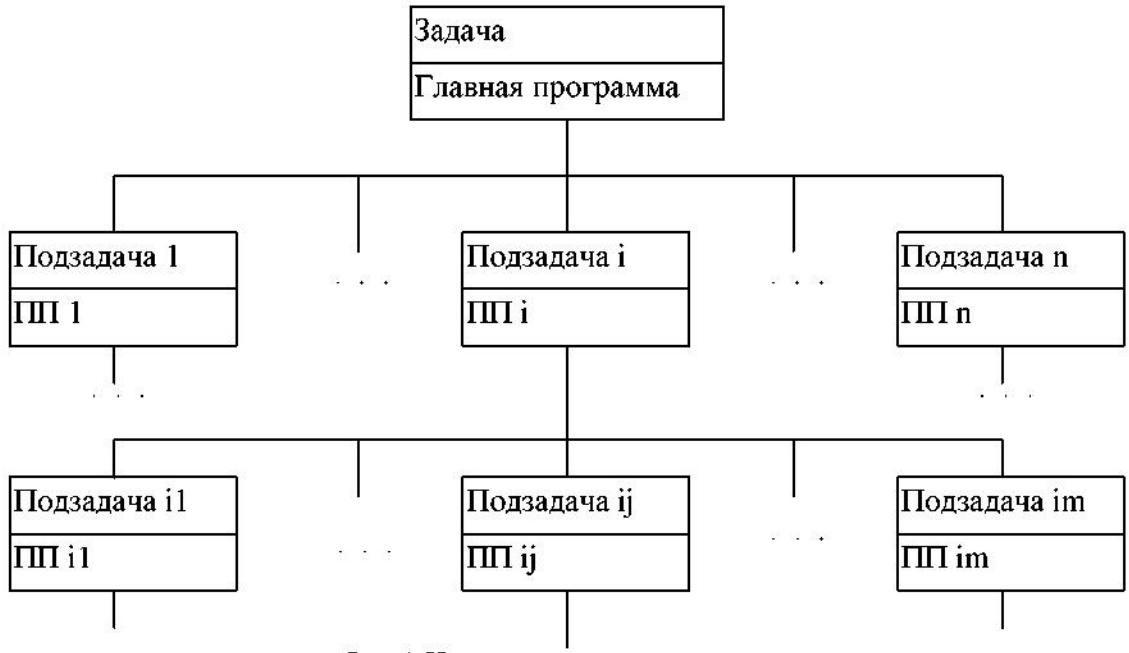


Рис. 1. Иерархическое дерево программы

После разработки общего проекта программы в схемах или псевдокодах наступает момент реализации проекта.

При реализации проекта программы в мировой практике наработано множество методов или правил проектирования программ, но наибольшее распространение получил метод структурного программирования.

16.1. Основы структурного программирования

Структурное программирование - это емкое понятие, отражающее современный общепринятый стиль программирования. Для программ, написанных в структурной форме, характерно: широкое использование комментариев, тщательный выбор осмысленных имен, аккуратная организация текста, продуманные форматы отображаемых выходных данных, наконец, программная документация в полном объеме.

Комментарии следует отнести к наиболее существенным атрибутам хорошего стиля программирования. Их назначение - помогать читать и понимать программу. При чтении тщательно продуманных комментариев пользователь может позволить себе не погружаться в детали программной части текста; с другой стороны, комментарии не должны становиться помехой В программе рекомендуется использовать не абстрактные имена переменных типа A, B и тому подобные, а смысловые или ***мнемонические имена***. Мнемонические имена наиболее информативны и повышают понимаемость программы. Например, RecStud - запись данных по студенту, MaxNumStud максимальное количество студентов и т.п.

Существует еще несколько рекомендаций по ***оформлению текста*** программы:

1. Не следует на одной строке располагать несколько предложений.

2. Рекомендуется включать пустые строки. С помощью пустых строк можно выделить ту или иную часть текста (группу операторов)

- они помогут привлечь к ней внимание. Для компилятора пустые строки безразличны, он их игнорирует.

3. Предложения программы следует располагать, имея в виду следующие правила:

а) ключевые слова, такие как ***const, type, var, begin, end***, в главной программе должны быть выравнены по левому краю текста;

б) предложения внутри цикла ***for*** (или цикла ***while***) должны быть набраны с отступом;

в) если в программе имеется конструкция ***if-then-else***, то ключевые слова ***then*** и ***else*** следует выравнить относительно друг друга;

г) предложения внутри программы нужно набирать с отступом и так далее.

Следующим важным фактором, на который необходимо обратить внимание при разработке программы, является ***наглядный вывод*** результатов. Это обусловлено тем, что основной целью разработки программы является получение результатов решения проблемы или задачи. Принципы и средства создания наглядного вывода и ввода рассмотрены в лабораторной работе N12.

При отладке программы полезно в текст вставлять ***отладочную печать***, предназначенную для вывода промежуточных результатов вычислений.

Всякая профессионально разработанная программа должна сопровождаться поясняющей ее ***документацией***. В комплект документов обычно включают:

исходное описание задачи;

схему алгоритма или описание логики программы;

листинг программы ;

тестовые наборы данных;

инструкции по эксплуатации программы;

результаты ее испытаний на некоторых тестовых наборах данных.

16.2. Средства Паскаля для разработки программ модульной структуры

Модуль - это автономно компилируемая программная единица, включающая в себя различные компоненты раздела описаний (типы, константы, переменные, процедуры и функции) и, возможно, некоторые исполняемые операторы инициирующей части. По организации модули в Турбо-Паскале (TP) характеризуются следующим:

1) явным образом выделяется некоторая "видимая" интерфейсная часть, в которой сконцентрированы описания глобальных типов, констант и переменных;
2) в них приводятся заголовки глобальных процедур и функций. Тела процедур и функций располагаются в исполняемой части модуля, которая может быть скрыта от пользователя.

Важной особенностью модулей является то обстоятельство, что компилятор TP размещает их программный код в отдельном сегменте памяти. Максимальная длина сегмента не может превышать 64 Кбайта, однако количество одновременно используемых модулей ограничивается лишь доступной памятью, что дает возможность создавать весьма крупные программы.

16.2.1. Структура модулей

Модуль имеет следующую структуру:

```
UNIT<имя>
INTERFACE                    }
<интерфейсная часть>         } секция интерфейса
IMPLEMENTATION               }
<исполняемая часть>          }
[BEGIN                       } секция реализации
<инициирующая часть>         }
END                          }
```

В этом описании:

UNIT - ключевое слово, начинающее заголовок модуля (с англ. модуль);

<имя> - имя модуля (идентификатор);

INTERFACE - ключевое слово, начинающее интерфейсную часть модуля;

IMPLEMENTATION - ключевое слово, начинающее исполняемую часть (с англ. исполнение);

BEGIN - ключевое слово, начинающее инициирующую часть. Эта часть модуля, заключенная в [ ], необязательна.

END - признак конца модуля.

Таким образом, модуль состоит из заголовка и трех составных частей, любая из которых может быть пустой.

16.2.2. Заголовок модуля и связь модулей друг с другом

Заголовок модуля состоит из служебного слова UNIT и имени модуля. Для правильной работы среды TP и возможности подключения средств, облегчающих разработку больших программ, имя модуля должно совпадать с именем дискового файла, в которой помещается исходный текст модуля. Например, если имя модуля имеет следующий заголовок:

UNIT global;

то исходный текст соответствующего модуля должен размещаться в дисковом файле global.pas. Имя модуля служит для его связи с другими модулями и основной программой. Эта связь устанавливается специальной конструкцией языка:

USES <список модулей>

где USES -ключевое слово (с англ. использует)

<список модулей> - список модулей, с которыми устанавливается связь. Элементами списка являются имена модулей, отделенные запятой.

Например:

USES crt, graph, global;

Если в программе используется предложение USES ... , то оно должно открывать раздел описаний основной программы или может следовать за служебным словом INTERFACE в модуле. Тогда структура модуля имеет вид:

UNIT<имя>;
INTERFACE
[USES <список модулей>;]
<интерфейсная часть>
IMPLEMENTATION
<исполняемая часть>
[BEGIN
<инициирующая часть>;]
END.

Примечание. Если модуль, имя которого указано в предложении USES, использует другие модули, то имена этих модулей также должны быть указаны в предложении USES, причем до того, как они будут использоваться.

16.2.3. Интерфейсная часть модуля

Интерфейсная часть модуля открывается служебным словом INTERFACE. В этой части содержатся объявления всех глобальных объектов модуля (типов, констант, переменных и блоков). Эти объекты модуля становятся видимыми для любой программы или модуля, использующих данный модуль. Любая программа, использующая этот модуль имеет доступ к "видимым" элементам. Процедуры и функции, "видимые" для любой программы, использующей данный модуль, описываются в разделе INTERFACE, а их действительные тела находятся в части IMPLEMENTATION.

Если процедура или функция является внешней, то в интерфейсной части необходимо указать служебное слово EXTERNAL, а в части IMPLEMENTATION не нужно повторно указывать описание процедуры.

Пример:

```
UNIT Cmplx;
Interface
type
        complex=record
        re,im=real;
        end;
Procedure Addc (x,y:complex; var z:complex);
```

```
Procedure Mulc (x,y:complex; var z:complex);
...
...
End.
```

Теперь, если в основной программе использовать предложение

```
USES Cmplx;
```

то в программе станут доступными тип Complex и две процедуры Addc и Mulc из описанного выше модуля.

Примечание:

1. Константы и переменные, объявленные в интерфейсной части модуля, также как и глобальные объекты основной программы, помещаются в общий сегмент данных, максимальная длина которого 65621 байт.
2. Если в интерфейсной части объявляются внешние блоки (EXTERNAL) или блоки в машинных кодах (INLINE), то их тела должны следовать сразу за их заголовками.

16.2.4. Исполняемая часть модуля

Исполняемая часть модуля начинается словом IMPLEMENTATION и содержит тела процедур и функций, объявленных в интерфейсной части. В этой части также могут объявляться локальные для модуля объекты: вспомогательные типы, константы, переменные и блоки, а также метки, если они используются в инициирующей части. Локальные объекты являются невидимыми для программ, использующих данный модуль.

Объявленные в интерфейсной части глобальные процедуры и функции должны описываться в той же последовательности, в какой появляются их заголовки в интерфейсной части. Заголовки процедур и функций в исполняемой части должны быть идентичны тем, которые указаны в интерфейсной части, или могут иметь краткую форму: ключевое слово и имя процедуры или функции.

Пример:

```
Unit Cmplx;
Interface
type
        complex = record
        re, im : real;
        end;
Procedure Addc (x, y: complex; var z: complex);
Implementation
Procedure Addc;
        begin
                z.re := x.re + y.re;
                z.im := x.im + y.im;
        end;
end.
```

Примечание: процедуры и функции, локальные для исполняемой части модуля, т.е. не описанные в интерфейсной части, должны иметь полный, несокращенный заголовок.

16.2.5. Инициирующая часть модуля

Инициирующая часть завершает модуль. Она может отсутствовать вместе с начинающим ее словом Begin или быть пустой - тогда за Begin сразу следует признак конца модуля End. Такой случай возникает при создании модуля, содержащего данные, используемые несколькими модулями.

В этой части модуля помещаются исполняемые операторы, содержащие некоторый фрагмент программы. Они выполняются до передачи управления основной программе и предназначены для следующих целей:

- инициируются переменные, которые использует программный модуль или которые он делает доступными программе;
- открываются файлы, которые будут использоваться основной программой;
- устанавливается связь с другими ПЭВМ по коммуникационным каналам.

16.2.6. Компиляция модулей

В среде TP имеются средства, управляющие способами компиляции модулей и облегчающие разработку крупных программных продуктов. В частности, определены три режима компиляции: COMPILE, MAKE, BUILD. Эти режимы отличаются способом связи компилируемого модуля или компилируемой основной программы с другими модулями , объявленными в предложении USES.

При компиляции модуля или основной программы в режиме Compile все модули из предложения USES должны быть предварительно откомпилированы и результаты их компиляции должны быть помещены в одноименные файлы с расширением tpu.

Например, если в программе или модуле имеется предложение USES global, то на диске в каталоге, объявленном опцией UNIT DIRECTORIES, уже должен находиться файл Global.tpu, который создается в результате компиляции модуля.

В режиме MAKE компилятор проверяет наличие TPU-файлов для каждого объявленного модуля. Если какой-либо из файлов не обнаружен, система пытается отыскать одноименный файл с расширением pas, т.е. файл с исходным текстом модуля. Если внесены какие-либо изменения в pas-файл, то независимо от наличия в каталоге соответствующего tpu-файла, система осуществляет его компиляцию перед компиляцией основной программы. Более того, если изменения внесены в интерфейсную часть модуля, то будут перекомпилированы и все другие модули, обращающиеся к нему. Таким образом, режим MAKE существенно облегчает процесс разработки крупных программ с множеством модулей: программист избавляется от необходимости следить за соответствием tpu-файлов и их исходного текста, поскольку система выполняет его автоматически.

В режиме BUILD существующие tpu-файлы игнорируются, и система пытается отыскать и откомпилировать существующий pas-файл для каждого объявленного в USES модуля. После компиляции в этом режиме программист может быть уверен в том, что учтены все сделанные им изменения в любом из модулей.

Подключение модулей к основной программе осуществляется в порядке их объявления в предложении USES. При переходе к очередному модулю система предварительно отыскивает все модули, на которые он ссылается. В списке модулей данный модуль должен быть указан после всех тех модулей, которые он использует.

Если в программе не указано предложение USES, TP в любом случае присоединит стандартный модуль SYSTEM, который обеспечивает выполнение стандартных программ и подпрограмм, специфических для TP.

Контрольные вопросы

1. В чем заключается метод пошаговой детализации?
2. Перечислите правила структурного программирования.
3. Чем отличаются внешние ПП от внутренних?
4. Нарисуйте схему программы модульной структуры?
5. Каковы характерные особенности организации модуля в языке Паскаль?
6. Какую структуру имеет модуль в языке Паскаль?
7. В каких случаях можно отсутствовать раздел IMPLEMENTATION?
8. Какое расширение имеет имя файла с исходным текстом модуля?
9. Охарактеризуйте три режима компиляции модулей и основной программы.
10. Для чего предназначена интерфейсная часть модуля?
11. Можно ли в программе переопределить объекты, объявленные в интерфейсной части модуля?
12. Как получить доступ к "закрытому" объекту, значение которого вычисляется в модуле, но не используется в программе?

Задание к работе

Разработайте программу модульной структуры. За основу возьмите задание из лабораторной работы N 12.

Методические указания

1. Изучите постановку главной задачи с точки зрения теории модульного программирования и выделите основные цели задачи.
2. Выделите основные задачи, соответствующие различным целям ее решения,

например:

- ввод и контроль данных;
- обработка данных;
- сервисная работа с файлами;
- отображение данных;
- сервисная работа по созданию удобного интерфейса.

3. Каждая задача должна быть реализована в виде модуля.
4. Выделите цели по каждой задаче. Например, целями задачи "сервисная работа по созданию удобного интерфейса" являются:
   - создание меню по работе с программой;
   - выбор отдельных элементов меню и их распознавание;
   - отображение HELPa при необходимости и так далее.
5. 5. В каждой задаче выделите подзадачи. Например, задача "ввод и
   - формирование окна (окон) ввода данных;
   - контроль корректности ввода данных;
   - вывод диагностических сообщений;
   - другие подзадачи.
6. Каждая подзадача должна быть реализована в виде функции (функций) или процедуры (процедур).
7. При написании программы можно пользоваться ранее созданными Вами процедурами и функциями.
8. Выполните отладку программы, используя схему "снизу-вверх": процедура, модуль, программа.
9. Напишите отчет по работе и представьте его к защите.

Содержание отчета

1. Титульный лист.
2. Словесная постановка задачи.
3. Текстуальное или графическое описание алгоритма решения задачи.
4. Листинг программы.
5. Результаты тестирования программы (твердая копия структуры данных).
6. Анализ допущенных ошибок.
7. Инструкция к использованию программы.
8. Ответы на контрольные вопросы по согласованию с преподавателем.

ЗАКЛЮЧЕНИЕ

При преподавании дисциплины "Программирование и алгоритмические языки" кафедрой "Системы информатики" в качестве базового принят язык Паскаль. Выбор языка программирования обусловлен тем, что он создан специально для обучения дисциплине программирования.

Он быстро приобрел широкую популярность во всем мире, и в настоящее время реализован практически на всех вычислительных машинах от микроЭВМ до СуперЭВМ.

Язык Паскаль обладает многими достоинствами в плане обучения, среди которых можно выделить следующие: простота и ясность конструкций; высокая типизация данных; контроль типов; возможность построения новых типов данных; возможность достаточно полного контроля правильности программы на этапе компиляции и во время ее выполнения; наличие набора структурных типов данных: массивов, записей, записей с вариантами, множеств, файлов; возможность удовлетворения требованиям структурного программирования; гибкость и надежность; возможность программирования задач различного профиля.

К положительным качествам пособия, разработанного для изучения вопросов программирования на языке Паскаль, следует отнести:

продуманную структуру;

возможность эффективно проводить лабораторные занятия;

акцентирование внимания на особенностях использования отдельных операторов и приемов программирования;

повышение возможности систематизации знаний по программированию на языке Паскаль;

обеспечение приобретения базы общих знаний по программированию, которая в будущем позволит использовать их при изучении других систем программирования.

Как и любое издание, пособие не лишено недостатков:

не все контрольные вопросы представляют собой вопросы качественного характера;

не предусмотрены сквозные индивидуальные задания, охватывающие темы всех лабораторных работ.

ЛИТЕРАТУРА

1. Грехем Р. Практический курс языка Паскаль для микроЭВМ: Пер. с англ. - М.: Радио и связь, 1986. - 268 с.
2. Грогоно П. Программирование на языке Паскаль: Пер. с англ.- М.: Мир, 1982. - 382 с.
3. Перминов О.Н. Язык программирования Паскаль. - М.: Радио и связь, 1983. - 119 с.
4. Прайс Д. Программирование на языке Паскаль: Пер. с англ. - М.: Мир, 1987. - 232 с.
5. Фаронов В.В. Программирование на персональных ЭВМ в среде Турбо-Паскаль. - М.: Изд-во МГТУ, 1990. - 580 с.
6. Зуев Е.А. Язык программирования Турбо-Паскаль 6.0. - М.:Унитех, 1992. - 304 с.

СОДЕРЖАНИЕ