МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ»

Кафедра "Интеллектуальных информационных технологий"

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по выполнению лабораторных работ по курсу

«Основы алгоритмизации и программирования в традиционных и интеллектуальных компьютерах»

для студентов специальности I - 40 03 01 «Искусственный интеллект»

ЧАСТЬ 1

БРЕСТ 2008

УДК 681.3

Методические указания предназначены для выполнения лабораторных работ по дисциплине «Основы алгоритмизации и программирования в традиционных и интеллектуальных компьютерах» и содержат описания 9 лабораторных работ по изучению основ алгоритмизации и программирования на языке высокого уровня Турбо Паскаль, являются первой частью лабораторного курса по изучению стандартных алгоритмов, методов реализации их на ПЭВМ.

Методические указания предназначены для использования студентами специальности I-40 03 01 «Искусственный интеллект», издаются в 2 частях. Часть 1.

Составитель: Хацкевич М.В., ст. преподаватель

СОДЕРЖАНИЕ

1. ЛАБОРАТОРНАЯ РАБОТА №1
ЗНАКОМСТВО СО СРЕДОЙ ТУРБО ПАСКАЛЯ

Цель работы: изучить функциональные клавиши, текстовый редактор, диалоговую среду Турбо Паскаля.

1.1. Функциональные клавиши

Функциональные клавиши используются для управления средой Турбо Паскаля, обозначаются Fl, F2,..., F12.С каждой из функциональных клавиш связывается команда меню. Изменить действия функциональных клавиш можно тремя особыми клавишами: Alt , Ctr и Shift . Для этого необходимо нажать на одну из них и затем, не отпуская ее, нажать функциональную клавишу.

Fl - обратиться за справкой к встроенной справочной службе (Help-помощь);

F2 - записать редактируемый текст в дисковый файл;

F3 - прочитать текст из дискового файла в окно редактора;

F4 - используется в отладочном режиме: начать или продолжить исполнение программы и остановиться перед исполнением той ее строки, на которой стоит курсор;

F5 - распахнуть активное окно на весь экран;

F6 - сделать активным следующее окно;

F7 - используется в отладочном режиме: выполнить следующую строку программы; если в строке есть обращение к процедуре (функции), войти в эту процедуру и остановиться перед исполнением первого ее оператора;

F8 - используется в отладочном режиме: выполнить следующую строку программы; если в строке есть обращение к процедуре (функции), исполнить ее и не прослеживать ее работу;

F9 - компилировать программу, но не выполнять ее;

F10 - перейти к диалоговому выбору режима работы с помощью главного меню;

Ctrl-F9 - выполнить прогон программы: компилировать программу, находящуюся в редакторе, загрузить ее в оперативную память и выполнить, после чего вернуться в среду Турбо Паскаля.

Alt-F5 - сменить окно редактора на окно вывода результатов работы (прогона) программы.

1.2. Текстовый редактор среды Турбо Паскаля

Текстовый редактор среды Турбо Паскаля предоставляет пользователю удобные средства создания и редактирования текстов программ. Режим редактирования автоматически устанавливается после загрузки Турбо Паскаля. Из режима редактирования можно перейти к любому другому режиму работы Турбо Паскаля с помощью функциональных клавиш или выбора нужного режима из главного меню. Окно редактирования можно смещать относительно листа с помощью следующих клавиш:

Page Up - на страницу вверх;

Page Down - на страницу вниз;

Home - в начало текущей строки;

End - в конец текущей строки;

Ctrl- Page Up - в начало текста;

Ctrl- Page Down - в конец текста;

Delete - стирает символ, на который в данный момент указывает курсор;

Ctrl-Y - всю строку, на которой расположен курсор.

В режиме автоотступа каждая новая строка начинается в той же позиции на экране, что и предыдущая. Режим автоотступа поддерживает хороший стиль оформления текста программы: отступы от левого края выделяют тело условного или составного оператора и делают программу более наглядной. Отказаться от автоотступа можно командой

Ctrl-O+I (при нажатой Ctrl нажимается сначала клавиша с латинской буквой O, а затем O отпускается и нажимается I), повторная команда Ctrl-O I восстановит режим автоотступа.

Ниже перечислены наиболее часто используемые команды редактирования Турбо Паскаля.

Backspace - стирает символ слева от курсора;
Delete - стирает символ, на который показывает курсор;
Ctrl-Y- стирает строку с курсором;
Enter - вставляет новую строку, разрезает старую;
Ctrl-Q L - восстанавливает измененную строку;
Ctrl-K B - начинает выделение блока;
Ctrl-K K- заканчивает выделение блока;
Ctrl-K Y - уничтожает выделенный блок;
Ctrl-K C - копирует блок;
Ctrl-K V - перемещает блок на новое место;
Ctrl-K W- записывает блок в файл;
Ctrl-K R - читает блок из файла;
Ctrl-K P - печатает блок.

1.3. Диалоговая среда Турбо Паскаля

Управление средой Турбо Паскаля осуществляется с помощью системы последовательно разворачивающихся меню. Главное меню - постоянно присутствует на экране, остальные разворачиваются по мере выбора продолжений. Главное меню содержит оглавление дополнительных меню.

File (файл) - действия с файлами и выход из системы;
Edit (редактировать)- восстановление испорченной строки и операции с временным буфером;
Search (искать) - поиск текста, процедуры, функции или места ошибки;
Run (работа) - прогон программы;
Compile (компилировать) - компиляция программы;
Debug (отладка) - отладка программы;
Tools (инструменты) - вызов вспомогательных программ (утилит);
Options (варианты) - установка параметров среды;
Window (окно) - работа с окнами;
Help (помощь) - обращение к справочной службе.

Меню опции FILE

NEW. Создает и открывает новое окно редактора с именем NONAMExx.PAS.
OPEN. Открывает новое окно редактора и помещает в него указанный дисковый файл.
SAVE. Записывает содержимое активного окна редактора в дисковый файл.
SAVE AS. Записывает содержимое активного окна редактора в дисковый файл под другим именем.
SAVE ALL. Записывает содержимое всех окон редактора в соответствующие дисковые файлы.
CHANGE DIR. Позволяет изменить текущий каталог пользователя.
PRINT. Печатает содержимое активного окна редактора на принтере или выводит его в файл.
PRINTER SETUP. Настраивает среду на печать текущего файла.
DOS SHELL. Обеспечивает временный выход в ДОС.
EXIT. Завершает работу с Турбо Паскалем.

Меню опции EDIT

UNDO. В активном окне редактора восстанавливает только что уничтоженную командой Ctrl-Y или измененную строку.
REDO. Отменяет действие предыдущей команды UNDO.
CUT. Удаляет выделенный блок из окна редактора и переносит его в буфер обмена Clipboard.

COPY. Копирует выделенный блок из окна редактора в буфер обмена Clipboard.
PASTE. Копирует содержимое буфера обмена Clipboard в окно редактора. Содержимое буфера остается без изменений и может использоваться повторно.
CLEAR. Удаляет из окна редактора выделенный блок, но не помещает его в буфер.
SHOW CLIPBOARD. Показывает содержимое буфера обмена.

Меню опции SEARCH

FIND. Обеспечивает поиск нужного фрагмента текста в активном окне редактора.
REPLACE. Отыскивает в окне редактора нужный текстовый фрагмент и заменяет его на новый.
SEARCH AGAIN. Повторяет поиск или поиск и замену фрагмента текста для ранее установленных параметров.
GO TO LINE NUMBER. Осуществляет позиционирование курсора в окне редактора на строку с указанным номером.
SHOW LAST COMPILER ERROR. Показывает строку текста программы, в которой была обнаружена синтаксическая ошибка при последнем прогоне компилятора.
FIND ERROR. Отыскивает в тексте программы строку, вызвавшую ошибку периода исполнения программы

Меню опции RUN

RUN. Осуществляет компиляцию, компоновку и исполнение (прогон) программы из файла редактора. Компиляция проходит в режиме MAKE (опция COMPILE/MAKE). Если программа уже откомпилирована к этому моменту, то среда сразу начнет ее прогон.
GO TO CURSOR. Начинает или продолжает режим отладки исполняемой программы под управлением встроенного отладчика до положения курсора
TRACE INTO. Начинает или продолжает режим отладки исполняемой программы под управлением встроенного отладчика. Если к моменту обращения к этой опции режим отладки не был запущен, он запускается точно так, как если бы была вызвана опция GO TO CURSOR, однако программа останавливается перед первым исполняемым оператором, т.е. указатель будет указывать на слово BEGIN, открывающее раздел операторов основной программы. Если режим отладки уже был запущен, вызов этой опции приведет к выполнению всех действий, запрограммированных в текущей строке, и указатель сместится к следующей строке программы. Если текущая строка содержит обращение к процедуре или функции, управление будет передано внутрь этой процедуры (функции) и программа остановится перед исполнением ее первого оператора.
STEP OVER. Начинает или продолжает пошаговое прослеживание работы программы, но не прослеживается работа вызываемых процедур и функций.
PROGRAM RESET. Сбрасывает все ранее задействованные отладочные средства и прекращает отладку программы. Удаляет исполнявшуюся программу из памяти и закрывает все открытые в ней в этот момент файлы.
PARAMETERS. Позволяет задать текстовую строку параметров, которые ДОС передает вызываемой программе.

Меню опции COMPILE

COMPILE. Компилирует программу или модуль, который загружен в данный момент в активное окно редактора. Если в этой программе (модуле) содержатся обращения к нестандартным модулям пользователя, последние уже должны быть откомпилированы и храниться на диске в виде TPU-файлов.
MAKE. Создает программу, которая, возможно, содержит включаемые файлы и/или обращения к нестандартным модулям. Компилируется начальный файл, если он определен опцией COMPILE /PRIMARY FILE. Если начальный файл не задан, компилируется файл из активного окна редактора.
BUILD. Эта опция полностью подобна опции MAKE за одним исключением: для всех

TPU-файлов отыскивается соответствующий PAS-файл и осуществляется его перекомпиляция независимо от того, были ли сделаны в нем изменения или нет.
DESTINATION. Эта опция управляет выходом компилятора: если справа от нее стоит кодовое слово Memory (память), выходной файл компилятора будет сохранен в оперативной памяти и может затем сразу же запускаться из Турбо Паскаля без его загрузки с диска; если справа стоит кодовое слово Disk (диск), файл с кодом программы будет сохранен на диске в виде файла с расширением .EXE. Если объявлен начальный файл, его имя будет присвоено имени вновь создаваемого EXE-файла, в противном случае EXE-файл получит имя файла из того окна редактора, которое содержит текст основной программы.
PRIMARY FILE. Задает имя начального файла. Если это имя задано, то вне зависимости от того, какая часть программы загружена в данный момент в окна редактора, ее компиляция в режимах RUN, MAKE и BUILD будет начинаться с этого файла.
CLEAR PRIMARY FILE. Очищает имя начального файла, заданное опцией PRIMARY FILE.
INFORMATION. Показывает статистику программы.

Меню опции DEBUG

BREAKPOINTS. Опция позволяет просмотреть все контрольные точки и при необходимости удалить, переместить любую контрольную точку или задать условия ее работы.
CALL STACK. Делает активным окно программного стека. В этом окне отображаются все вызовы процедур и функций. Внизу стека находится PROGRAM, т.е. имя Вашей программы, в вершине стека – текущая процедура (функция). Каждое новое обращение к процедуре (функции) отображается в этом окне в виде имени подпрограммы и списка параметров вызова.
REGISTER. Делает активным окно регистров.
WATCH. Делает активным окно отладки.
OUTPUT. Делает активным окно программы.
USER SCREEN. Делает активным окно программы и распахивает его на весь экран.
EVALUATE/MODIFY. Эта опция дает возможность в процессе отладки просмотреть содержимое любой переменной или найти значение любого выражения.
ADD WATCH. С помощью этой опции указывают отладчику те переменные и/или выражения, за изменением значений которых необходимо наблюдать при отладке программы. Указанные переменные и выражения вместе с их текущими значениями будут постоянно содержаться в окне наблюдения, доступ к которому возможен с помощью клавиши F6.
ADD BREAKPOINT. С помощью этой опции можно установить в текущей строке контрольную точку.

Меню опции TOOLS

MESSAGES. Активизирует окно сообщений. Окно сообщений содержит вывод инструментальных программ типа GREP и позволяет использовать эти сообщения для поиска нужных фрагментов в текстах программ.
GO TO NEXT. Ищет фрагмент, заданный следующим сообщением в окне Messages.
GO TO PREVIOUS. Ищет фрагмент, заданный предыдущим сообщением в окне Messages.
GREP. Инициирует работу утилиты GREP. В строке Enter program arguments диалогового окна опции необходимо перечислить аргументы вызова GREP: имена процедур, функций, переменных, которые необходимо отыскать в текстовых файлах, а также имена этих файлов.

Меню опции OPTIONS

COMPILER. Эта опция задает несколько параметров, с помощью которых можно управлять генерацией машинного кода программы.
MEMORY SIZES. В диалоговом окне опции OPTIONS/MEMORY SIZES используются три

поля ввода. С их помощью можно регулировать размеры памяти, которую занимает работающая программа:

Stack size - размер программного стека; по умолчанию 16384 байта, максимум -65535 байт;
Low heap limit - минимальный размер кучи; по умолчанию 0;
High heap limit - максимальный размер кучи; по умолчанию 655360 байт; этот параметр не может быть меньше параметра Low heap limit.

LINKER. В диалоговом окне регулируется режим работы компоновщика Турбо Паскаля.
DEBUGGER. Эта опция определяет используемый отладчик и режим обновления экрана дисплея в процессе отладки.

Меню опции WINDOW

TILE. Располагает окна так, чтобы каждое было видно на экране, и все они имели бы приблизительно одинаковые размеры.
CASCADE. Располагает на экране окна редактора таким образом, чтобы были видны рамки каждого из них.
CLOSE ALL. Закрывает все открытые окна.
REFRESH DISPLAY. Удаляет следы вывода программы, работавшей в режиме отладки с установленной опцией Options/Debugger/Display swapping/None.
SIZE/MOVE. Эта опция обеспечивает перемещение окна по экрану и/или изменение его размеров.
ZOOM. Распахивает активное окно на весь экран или возвращает ему прежний вид. Вызов из редактора клавишей F5.
NEXT. Активизирует очередное окно.
PREVIOUS. Активизирует предыдущее активное окно.
CLOSE. Закрывает активное окно.
LIST. Выводит на экран список всех открытых окон среды.

Меню опции HELP

CONTENTS. Выводит на экран содержание справочной службы.
INDEX. Выводит на экран алфавитный список всех ссылок справочной службы.
TOPIC SEARCH. Осуществляет поиск в окрестности курсора зарезервированного слова или имени стандартной процедуры (функции) и дает соответствующую справку.
PREVIOUS TOPIC. Выводит на экран предыдущее справочное сообщение.
HELP ON HELP. Справка о том, как пользоваться справочной службой.
FILES. С помощью этой опции Вы можете установить нужные файлы справочной службы.
COMPILER RIRECTIVES. Показывает справку о директивах компилятора.
RESERVED WORDS. Показывает справку о зарезервированных словах.
STANDARD UNITS. Показывает справку о стандартных модулях.
TURBO PASCAL LANGUAGE. Показывает справку о языке Турбо Паскаль.
ERROR MESSAGES. Показывает справку о сообщении об ошибках.
ABOUT. Выводит информацию об авторских правах и версии Турбо Паскаля.

1.4. Порядок проведения работы

1. Изучить теоретические сведения.
2. Применить на практике полученные сведения.

1.5. Содержание отчета

Отчет по лабораторной работе оформляется в соответствии с общими требованиями и включает:

– Титульный лист.
– Тему и цель работы.
– Задание (согласно варианту).
– Описание и назначение основных пунктов меню.
– Выводы по работе.

1.6. Контрольные вопросы

Ответить на следующие контрольные вопросы:

1. Для чего предназначены функциональные клавиши?
2. Какие возможности предоставляет текстовый редактор среды Турбо Паскаля?
3. Что представляет собой диалоговая среда Турбо Паскаля?
4. Какие основные пункты меню? Их назначение?

2. ЛАБОРАТОРНАЯ РАБОТА №2
ЛИНЕЙНЫЕ ПРОГРАММЫ

Цель работы: изучить простые типы данных, стандартные математические функции, бинарные операции, элементарный ввод\ вывод, освоить реализацию простейшего линейного алгоритма.

2.1. Простейшая линейная программа

Линейная программа, как правило, составляется для вычисления значения некоторого выражения. Операции вычисления заданного выражения и записи в память полученного значения выполняются с помощью оператора присваивания.

«идентификатор»:= «выражение» ; ,

где **«идентификатор» -** имя переменной или функции; знак **:=** - знак операции присваивания, его не следует путать с операцией отношения(=).

При использовании оператора присваивания v:=e следует непременно учитывать, что переменная v и выражение е должны иметь одинаковый тип. Имеется лишь одно исключение из этого правила: переменная может иметь тип real, а выражение значение типа **integer**.

Идентификатор не может начинаться с цифры. Идентификатор не может совпадать ни с одним из зарезервированных слов. Длина идентификатора может быть произвольной, но значащими считаются первые 63 символа.

Описать идентификатор - указать тип связанного с ним объекта программы (константы или переменной). Тип данных определяет длину внутреннего представления соответствующих переменных.

Выражения строятся из операндов (констант, переменных, функций), знаков операций и круглых скобок. Константы, переменные и функции должны быть либо описаны в программе, либо иметь стандартные имена.

Тогда общая структура программы выглядит так:

```
Program lab;
  Label;      {описание меток}
  Const;      {описание констант}
  Type        {описание типов}
Var           {описание переменных}
  Procedure;  {описание процедур}
  Function;   {описание функций}
Begin

End.
```

2.2. Типы данных

При описании переменной необходимо указать ее тип. Тип переменной описывает набор значений, которые она может принимать, и действия, которые могут быть над ней выполнены. Описание типа определяет идентификатор, который обозначает этот тип.

Указание идентификатора в левой части описания типа означает, что он определен как идентификатор типа для блока, в котором указано это описание типа. Область дей-

ствия идентификатора типа не включает его самого, исключение составляют типы "указатель" (которые называют также ссылочными типами).

Простые типы

Простые типы определяют упорядоченные множества значений.

простой тип:

– порядковый тип;

– вещественный тип.

Идентификатор вещественного типа относится к числу стандартных идентификаторов, которые могут быть вещественными, с одинарной точностью, с двойной точностью, с повышенной точностью и сложными.

Порядковые типы

Порядковые типы представляют собой подмножество простых типов. Все простые типы, отличные от вещественных типов, являются порядковыми и выделяются по следующим четырем характеристикам:

- Все возможные значения данного порядкового типа представляют собой упорядоченное множество, и каждое возможное значение связано с порядковым номером, который представляет собой целочисленное значение. За исключением значений целочисленного типа, первое значение любого порядкового типа имеет порядковый номер 0, следующее значение имеет порядковый номер 1 и так далее для каждого значения в этом порядковом типе. Порядковым номером значения целочисленного типа является само это значение. В любом порядковом типе каждому значению, кроме первого, предшествует другое значение, и после каждого значения, кроме последнего, следует другое значение в соответствии с упорядоченностью типа.
- К любому значению порядкового типа можно применить стандартную функцию **Ord**, возвращающую порядковый номер этого значения.
- К любому значению порядкового типа можно применить стандартную функцию **Pred**, возвращающую предшествующее этому значению значение. Если эта функция применяется к первому значению в этом порядковом типе, то выдается сообщение об ошибке.
- К любому значению порядкового типа можно применить стандартную функцию Succ, возвращающую следующее за этим значением значение. Если эта функция применяется к последнему значению в этом порядковом типе, то выдается сообщение об ошибке.
- К любому значению порядкового типа и к ссылке на переменную порядкового типа можно применить стандартную функцию **Low**, возвращающую наименьшее значение в диапазоне данного порядкового типа.
- К любому значению порядкового типа и к ссылке на переменную порядкового типа можно применить стандартную функцию **High**, возвращающую наибольшее значение в диапазоне данного порядкового типа.

Pascal имеет 10 встроенных порядковых типов: **Integer**(целое), **Shortint** (короткое целое), **Longint** (длинное целое), **Byte** (длиной в байт), **Word** (длиной в слово), **Boolean** (булевское), **ByteBool** (булевское размером в байт), **WordBool** (булевское размером в слово), **LongBool** (длинный булевский тип) и **Char** (символьный тип). Кроме того, имеется два других класса определяемых пользователем порядковых типов: перечислимые типы и отрезки типов (поддиапазоны).

Целочисленные типы

В Pascal имеется пять целочисленных типов: **Shortint** (короткое целое), **Integer** (целое), **Longint** (длинное целое), **Byte** (длиной в байт) и **Word** (длиной в слово). Каждый тип обозначает определенное подмножество целых чисел, как это показано в Таблице 2.1

Таблица 2.1 Предопределенные целочисленные типы

Тип	Диапазон	Формат
короткое целое (Shortint)	-128 .. 127	8 бит со знаком
Целое (Integer)	-32768 .. 32767	16 бит со знаком
длинное целое (Longint)	-2147483648 2147483647	32 бита со знаком
длиной в байт (Byte)	0 .. 255	8 бит без знака
длиной в слово (Word)	0 .. 65535	16 бит без знака

Арифметические действия над операндами целочисленного типа предполагают 8-битовую, 16-битовую и 32-битовую точность в соответствии со следующими правилами:

- Тип целой константы представляет собой встроенный целочисленный тип с наименьшим диапазоном, включающим значение этой целой константы.
- В случае бинарной операции (операции, использующей два операнда), оба операнда преобразуются к их общему типу перед тем, как над ними совершается действие. Общим типом является встроенный целочисленный тип с наименьшим диапазоном, включающим все возможные значения обоих типов. Например, общим типом для целого и целого длиной в байт является целое, а общим типом для целого и целого длиной в слово является длинное целое. Действие выполняется в соответствии с точностью общего типа и типом результата является общий тип.
- Выражение справа в операторе присваивания вычисляется, независимо от размера или типа переменной слева.
- Любые операнды размером в байт преобразуются к промежуточному операнду размером в слово, который совместим перед выполнением арифметической операции с типами **Integer** и **Word**.

Значение одного целочисленного типа может быть явным образом преобразовано к другому целочисленному типу с помощью приведения типов.

Булевские типы (Логический тип)

Существует 4 булевских типа:**Boolean**, **ByteBool**, **WordBool** и **LongBool**. Значения булевского типа обозначаются встроенными идентификаторами констант **False** и **True**. Поскольку булевский тип является перечислимым, между этими значениями имеют место следующие отношения:

1. False < True;
2. **Ord**(False) = 0;
3. **Ord**(True) = 1;
4. **Succ**(False) = True;
5. **Pred**(True) = False.

Переменные типа **Boolean** и **ByteBool** занимают 1 байт, переменная **WordBool** занимает два байта (слово), а переменная **LongBool** занимает четыре байта (два слова). **Boolean** - это наиболее предпочтительный тип, использующий меньше памяти; тип **ByteBool**, **WordBool** и **LongBool** обеспечивают совместимость с другими языками и средой Windows.

Предполагается, что переменная типа **Boolean** имеет порядковые значения 0 и 1, но переменные типа **ByteBool**, **WordBool** и **LongBool** могут иметь другие порядковые значения. Когда выражение типа **ByteBool**, **WordBool** или **LongBool** равна 1, то подразумевается, что она имеет значение **True**, а если оно равно 0 - то **False**. Когда значение типа **ByteBool**, **WordBool** или **LongBool** используется в контексте, где ожидается значение **Boolean**, компилятор будет автоматически генерировать код, преобразующий любое ненулевое значение в значение **True**. Переменные и функции типа **boolean** могут связываться через операции, представленные в Таблице 2.2

Таблица 2.2 Операции над типом boolean

	Операция	Тип операнда	Тип результата	Значение
0	Not	boolean	boolean	Отрицание
1	And	boolean	boolean	Конъюнкция И
2	Or Xor	boolean boolean	boolean boolean	Дизъюнкция ИЛИ Исключающее ИЛИ
3	= <> < <= > >=	простой тип	boolean	Равно Неравно Меньше Меньше или равно Больше Больше или равно

Результаты логических операций and, or, xor представлены в Таблице 2.3

Таблица 2.3 Таблица истинности логических операций **and**, **or**, **xor**

A	B	A and B	A or B	A xor B
False	False	False	False	False
False	True	False	True	True
True	False	False	True	True
True	True	True	True	False

Заметим, что функция **ORD** преобразует к целому не только символы, но и логические величины.

Ord(false)=0;

Ord(true)=1;

Символьный тип (char)

Тип данных char обозначает множество символов кода **ASCII**. Один символ требует для своего внутреннего представления 8 бит = 1 байту. Константы такого типа обрамляются апострофами ('это строка '). Первые 32 символа в таблице **ASCII** являются управляющими. Для того чтобы включить в последовательность символов такие символы **ASCII** , введено понятие управляющего символа. Предусмотрено два способа записи:

#13#10 -возврат каретки, перевод строки;

^M^K -возврат каретки, перевод строки;

Для данных типа char существуют следующие стандартные функции (Таблица 2.4):

Таблица 2.4 Стандартные функции для данных типа char

Вызов	Параметр	Значение	Действие
Ord(x)	X:char	integer	Номер ASCII символа x
Chr(x)	X:integer	char	ASCII символ с номером x
Upcase(x)	X:char	char	Значением является большая буква x, если она существует (в противном случае x остается неизменным)

Вещественные типы

К вещественному типу относится подмножество вещественных чисел, которые могут быть представлены в формате с плавающей точкой с фиксированным числом цифр.

Имеется пять видов вещественных типов: вещественное (**Real**), с одинарной точностью (**Single**), с двойной точностью (**Double**), с повышенной точностью (**Extended**) и сложное (**Comp**). Действия над типами с одинарной точностью, с двойной точностью и с повышенной точностью и над сложным типом могут выполняться только при наличии числового сопроцессора 8087. Вещественные типы различаются диапазоном и точно-

стью связанных с ними значений Диапазон представления и десятичные цифры для вещественных типов представлены в Таблице 2.5

Таблица 2.5 Вещественные числа

Тип	Диапазон	Цифры
вещественное (**Real**)	2.9x10^-39 .. 1.7x10^38	от 11 до 12
с одинарной точностью (**Singl**)	1.5x10^-45 .. 3.4x10^38	от 7 до 8
с двойной точностью (**Double**)	5.0x10^-324 .. 1.7x10^308	от 15 до 16
с повышенной точностью (**Extended**)	1.9x10^-4951 .. 1.1x10^4932	от 19 до 20
Сложный тип (**Comp**)	-2^63 + 1 .. 2^63 - 1	

Примечание: Сложный тип содержит только целочисленные значения в диапазоне от -2^63+1 до 2^63-1, что приблизительно равно -9.2x10^18 и 9.2x10^18.

Pascal поддерживает две модели генерации кода для выполнения действий над вещественными типами: программную – для чисел с плавающей точкой и аппаратную-для чисел с плавающей точкой.

2.3. Тождественные и совместимые типы

Два типа могут быть тождественными, и эта тождественность (идентичность) является обязательной в некоторых контекстах. В других случаях два типа должны быть только совместимы или совместимы по присваиванию. Два типа являются тождественными, если они описаны с одним идентификатором типа, или если их определения используют один и тот же идентификатор типа.

Совместимость типов

Совместимость типов имеет место, если выполняется, по крайней мере, одно из следующих условий:

- Оба типа являются одинаковыми.
- Оба типа являются вещественными типами.
- Оба типа являются целочисленными.
- Один тип является поддиапазоном другого типа.
- Оба типа являются отрезками одного и того же основного типа.

Совместимость по присваиванию

Совместимость по присваиванию необходима, если имеет место присваивание значения, например, в операторе присваивания или при передаче значений параметров. Значение типа T1 является совместимым по присваиванию с типом T2 (допустим оператор T1:=T2), если выполняется одно из следующих условий:

- T1 и T2 имеют тождественные типы, и ни один из них не является файловым типом или структурным типом, содержащим компонент с файловым типом на одном из своих уровней.
- T1 и T2 являются совместимыми порядковыми типами, и значения типа T2 попадают в диапазон возможных значений T1.
- T1 и T2 являются вещественными типами, и значения типа T2 попадают в диапазон возможных значений T1.
- T1 является вещественным типом, а T2 является целочисленным типом.
- T1 и T2 являются строковыми типами.
- T1 является строковым типом, а T2 является символьным типом (**Char**).

2.4. Выражения

Выражения состоят из операций и операндов. Большинство операций в языке Паскаль являются бинарными, то есть содержат два операнда. Приоритет операций приведен в Таблице 2.6

Таблица 2.6 Старшинство операций (приоритет)

Операция	Приоритет	Вид операции
@, not	первый (высший)	унарная операция
*, /, div, mod, and, shl, shr	второй	операция умножения, деления, сдвига
+, -, or, xor	третий	операция сложения
=, <>, <, >, <=, >=, in	четвертый (низший)	операция отношения

Операции подразделяются на арифметические операции, логические операции, операции со строками, операции над множествами, операции отношения и операцию @ (операция получения адреса).

Бинарные арифметические операции приведены в Таблице 2.7

Таблица 2.7 Бинарные арифметические операции

Операция	Действие	Типы операндов	Тип результата
+	Сложение	Целый Вещественный	Целый Вещественный
-	Вычитание	Целый Вещественный	Целый Вещественный
*	Умножение	Целый Вещественный	Целый Вещественный
/	Деление	Целый Вещественный	Вещественный Вещественный
Div	Целочисленное деление	Целый	Целый
Mod	Остаток	Целый	Целый

Значение выражение i div j представляет собой математическое частное от i/j, округленное в меньшую сторону до значения целого типа. Если j равно 0, результат будет ошибочным.

Операция mod возвращает остаток, полученный путем деления двух ее операндов, то есть:

i mod j = i - (i **div** j) * j

Знак результата операции **mod** будет тем же, что и знак i. Если j равно нулю, то результатом будет ошибка.

В Паскале реализованы стандартные математические функции (см. Таблица 2.8).

Таблица 2.8 Стандартные математические функции

Обращение	Тип параметра	Тип результата	Примечание
Abs(x)	Real, integer	Тип аргумента	Модуль аргумента
ArcTan (x)	Real	Real	Арктангенс (значение в радианах)
Cos(x)	Real	Real	Косинус угла в радианах
Exp(x)	Real	Real	Экспонента
Frac(x)	Real	Real	Дробная часть числа
Int(x)	Real	Real	Целая часть числа
Ln(x)	Real	Real	Логарифм натуральный
Pi	-	Real	π=3,14159
Random(x)	Integer	Integer	Псевдослучайное число, равномерно распределенное в диапазоне (0..x-1)
Randomize	-	-	Инициация генератора псевдослучайных чисел
Sin(x)	Real	Real	Синус, угол в радианах
Sqr(x)	Real	Real	Квадрат в радианах
Sqrt(x)	Real	Real	Корень квадратный

2.5. Элементарный ввод и вывод

Для ввода с клавиатуры (**input**) существует процедура **read**. В отличие от других процедур **read** может иметь переменное число параметров, а **readln** может не иметь их вовсе.

Вывод на экран осуществляется с помощью операторов **write** и **writeln**.

Стандартный формат (стандартный вывод):

Integer – сколько разрядов, столько требуется для записи числа
Real – всего 18 разрядов
Boolean – false или true
Char - символ или символы
String - символ или символы

Формат отличный от стандартного (форматный вывод):

Pi:d – d-выражение типа integer, задающее ширину поля данных, в которое должно быть записано значение pi с выравниванием по правому краю.

Pi:d:s – d -используется, как только что описано выше, s-выражение типа integer , задающее число знаков после десятичной точки.

Если задаваемая ширина поля данных d будет выдана слишком маленькой, d расширяется до нужного числа позиций.

Пример,
Write(pi:3);
Writeln(pi:1:2);

Параметрами write (writeln) могут являться выражения типа write(2+3*4/(X+2), chr(123))

2.6. Порядок проведения работы

1. Изучить теоретические сведения.
2. Разработать блок-схему алгоритма.
3. Реализовать алгоритм на Турбо Паскале.
4. Протестировать программу.
5. Ответить на контрольные вопросы.
6. Оформить отчет.

2.7. Содержание отчета

Оформить отчет, содержащий:

1. Титульный лист.
2. Тему и цель работы.
3. Задание (согласно варианту).
4. Блок – схему алгоритма.
5. Листинг программы.
6. Тестовые примеры.
7. Выводы по работе.

2.8. Контрольные вопросы

1. Что такое линейный алгоритм?
2. Какие стандартные типы данных реализованы в Турбо Паскале?
3. Какие простые типы данных изучили?
4. Какие основные виды операций знаете?
5. Какие стандартные математические функции реализованы в Турбо Паскале?
6. Как осуществляется ввод\вывод данных в программе?

3. ЛАБОРАТОРНАЯ №3
УСЛОВНЫЙ ОПЕРАТОР

Цель работы: изучить один из способов ветвления алгоритма - условный оператор if.

3.1. Условный оператор if

Условный оператор позволяет проверить некоторое условие и в зависимости от результатов проверки выполнить то или иное действие. Условный оператор - это средство ветвления вычислительного процесса.

Структура условного оператора имеет следующий вид:

```
IF <условие>
   THEN  <оператор1>
         ELSE <оператор2>   ,
```

где IF, THEN, ELSE - зарезервированные слова (если, то, иначе);

<условие> - произвольное выражение логического типа;

<оператор1>, <оператор2> - любые операторы языка Турбо Паскаль.

Вначале вычисляется условное выражение <условие>. Если результат есть **TRUE** (истина), то выполняется <оператор1>, а <оператор2> пропускается; если результат есть **FALSE** (ложь), наоборот, <оператор1> пропускается, а выполняется <оператор2>.

Например:

```
var
  x, y, min: integer;
  begin
   if x > min
      then
         y:= min
      else
         y:= x;
```

При выполнении этого фрагмента переменная Y получит значение переменной x, если только это значение не превышает min, иначе y станет равно min.

Часть **ELSE** <оператор2> условного оператора может быть опущена. Тогда при значении TRUE условного выражения выполняется <оператор1>, в противном случае этот оператор пропускается:

```
var
x, y, min: integer;
begin
 if x < min then max := x;
 y := x;
.....
end;
```

В этом примере переменная y после выполнения программы будет иметь значение переменной y, а в min запоминается минимальное значение x.

Любая встретившаяся часть **ELSE** соответствует ближайшей к ней «сверху»части THEN условного оператора.

Например:

```
var
a,b,c,d : integer;
begin
a : = 1; b : = 2; C : = 3 ; d : =4 ;
if a > b then
```

```
    if c < d then
      if c < 0 then c := 0
      else
      a := b; {a равно 1}
  if a > b then
      if c<d then
        if c<b then c := 0
        else
          else
            else
  a := b; {a равно 2}
  end.
```

3.2. Порядок проведения работы

1. Изучить теоретические сведения.
2. Разработать блок-схему алгоритма.
3. Реализовать алгоритм на Турбо Паскале.
4. Протестировать программу.
5. Ответить на контрольные вопросы.
6. Оформить отчет.

3.3. Содержание отчета

Оформить отчет, содержащий:

1. Титульный лист.
2. Тему и цель работы.
3. Задание (согласно варианту).
4. Блок – схему алгоритма.
5. Листинг программы.
6. Тестовые примеры.
7. Выводы по работе.

3.4. Контрольные вопросы

1. Что такое оператор?
2. Какой формат записи условного оператора if?
3. Когда необходимо использовать условный оператор if?
4. Расширенный формат записи условного оператора if?

4. ЛАБОРАТОРНАЯ РАБОТА №4 ОПЕРАТОРЫ ЦИКЛА.

Цель работы: изучить операторы цикла **for**, **repeat**, **while**.

4.1. Оператор цикла FOR

В языке Турбо Паскаль имеются три различных оператора, с помощью которых можно запрограммировать повторяющиеся фрагменты программ.

Счетный оператор цикла FOR имеет такую структуру:
FOR <пар_цик> := <нач_знач> **TO** <кон_знач> **DO** <оператор>.

Здесь **FOR**, **TO**, **DO** - зарезервированные слова (для, до, выполнить);
<пар_цик> - параметр цикла - переменная типа любого порядкового типа;
<нач_знач> - начальное значение - выражение того же типа;
<кон_знач> - конечное значение - выражение того же типа;
<оператор> - произвольный оператор Турбо Паскаля.

При выполнении оператора **FOR** вначале вычисляется выражение <нач_знач> и осуществляется присваивание <пар_цик> : = <нач_знач>. После этого циклически повторяется:

1. проверка условия <пар__цик> меньше либо равно <кон_знач>;
2. если условие не выполнено, оператор **FOR** завершает свою работу;
3. если условие верно, то затем выполняется оператор <оператор>;
4. увеличивается значение переменной <пар_цик> на единицу.

Пример

```
Program Summa;
{ подсчитывает сумму всех целых чисел от 1 до N}
var
i, n, s : Integer;
begin
 Write('N = ');
 ReadLn(n); {Вводим N}
 s:=0; {Начальное значение суммы}
 for i : = 1 to n do {Цикл подсчета суммы}
    s := s + i;
 writeln('Сумма = ',s) {Выводим результат}
end.
```

Условие, управляющее работой оператора **FOR**, проверяется перед выполнением оператора <оператор>: если условие не выполняется в самом начале работы оператора **FOR**, исполняемый оператор не будет выполнен ни разу. Шаг наращивания параметра цикла строго постоянен и равен (+1).

Существует другая форма оператора:

FOR <пар__цик>: = <нач_знач> **DOWNTO** <кон_знач> **DO** <оператор>;

DOWNTO означает, что шаг наращивания параметра цикла равен (-1), а управляющее условие приобретает вид <пар_цик> = <кон_знач>.

Пример,

```
Program Summa;
{ подсчитывает сумму всех целых чисел от 1 до N}
var
i, n, s : Integer;
begin
 Write('N = ');
 ReadLn(n); {Вводим N}
 s := 0;
 if n >= 0 then
       for i := 1 to n do  s := s + i
          else
             for i := -1 downto n do
                S := S + i
End.
```

4.2. Оператор цикла WHILE

Оператор цикла **WHILE** с предпроверкой условия:

Формат записи:
WHILE <условие> **DO** <оператор>.

Здесь **WHILE**, **DO** - зарезервированные слова (пока [выполняется условие], делать);

<условие> - выражение логического типа;
<оператор> - произвольный оператор Турбо Паскаля.

Если выражение <условие> имеет значение **TRUE**, то выполняется <оператор>, после чего вычисление выражения <условие> и его проверка повторяются. Если <условие> имеет значение **FALSE** , оператор **WHILE** прекращает свою работу.

Пример,

```
Program Summa;
var
i, n, s : integer;
begin
 Write('N = ');
 ReadLn(n); {Вводим N}
 s := 0;
 while n<>s do
       begin
          s:=s+1;
          writeln(s);
          end;
end.
```

4.3. Оператор цикла REPEAT

Оператор цикла **REPEAT... UNTIL** с постпроверкой условия:

REPEAT <тело_цикла> **UNTIL** <условие>.

Здесь **REPEAT**, **UNTIL**- зарезервированные слова (повторять до тех пор, пока не будет выполнено условие);

<тело_цикла> - произвольная последовательность операторов Турбо Паскаля;

<условие> - выражение логического типа.

Операторы <тело_цикла> выполняются хотя бы один раз, после чего вычисляется выражение <условие>: если его значение есть **FALSE**, операторы <тело_цикла> повторяются, в противном случае оператор **REPEAT... UNTIL** завершает свою работу.

Для иллюстрации применения оператора **REPEAT... UNTIL** приведем пример программы, которая все время повторять цикл ввода символа и печати его кода до тех пор, пока очередным символом не будет символ **CR** (вводится клавишей Enter).

Пример,

```
Program Codes;
{Программа вводит символ и выводит на экран его код. Для за-
завершения работы программы нужно дважды нажать Enter}
var
ch : Char; {Вводимый символ}
const
CR = 13; {Код символа CR}
begin
 repeat
    ReadLn(ch);
    WriteLn(ch, ' = ',ord(ch));
 until ord(ch) = CR;
end.
```

Обратите внимание: пара **REPEAT... UNTIL** подобна операторным скобкам **begin... end**, поэтому перед **UNTIL** ставить точку с запятой необязательно.

Для гибкого управления циклическими операторами **FOR**, **WHILE** и **REPEAT** в состав Турбо Паскаля включены две процедуры:

BREAK - реализует немедленный выход из цикла; действие процедуры заключается в передаче управления оператору, стоящему сразу за концом циклического оператора;

CONTINUE - обеспечивает досрочное завершение очередного прохода цикла; эквивалент передачи управления в самый конец циклического оператора.

Введение в язык этих процедур практически исключает необходимость использования операторов безусловного перехода **GOTO**.

4.4. Порядок проведения работы

1. Изучить теоретические сведения.
2. Разработать блок-схему алгоритма.
3. Реализовать алгоритм на Турбо Паскале.
4. Протестировать программу.
5. Ответить на контрольные вопросы.
6. Оформить отчет.

4.5. Содержание отчета

Оформить отчет, содержащий:

1. Титульный лист.
2. Тему и цель работы.
3. Задание (согласно варианту).
4. Блок – схему алгоритма.
5. Листинг программы.
6. Тестовые примеры.
7. Выводы по работе.

4.6. Контрольные вопросы

1. Для чего предназначены операторы повтора?
2. Формат записи оператора **FOR**?
3. Формат записи оператора **WHILE**?
4. Формат записи оператора **REPEAT**?

5. ЛАБОРАТОРНАЯ РАБОТА№5
ОПЕРАТОР ВЫБОРА CASE. МЕТКИ И ОПЕРАТОРЫ ПЕРЕХОДА

Цель работы: изучить оператор выбора **case**. Изучить метки и оператор перехода.

5.1. Оператор выбора CASE

Оператор выбора позволяет выбрать одно из нескольких возможных продолжений программы. Выбор осуществляется по параметру :ключ выбора -выражение любого порядкового типа.

Структура оператора выбора такова:

CASE <ключ_выбора> **OF** <список_выбора> [**ELSE** <операторы>] **END**

Здесь **CASE**, **OF**, **ELSE**, **END** - зарезервированные слова (случай, из, иначе, конец);
<ключ_выбора> - ключ выбора;
<список_выбора> - одна или более конструкций вида:
<константа_выбора> : <оператор>;
<константа_выбора> - константа того же типа, что и выражение
<ключ_выбора>;
<операторы> - произвольные операторы Турбо Паскаля.

Оператор выбора работает следующим образом. Вначале вычисляется значение выражения <ключ_выбора>, а затем в последовательности операторов <список_выбора> отыскивается такой, которому предшествует константа, равная вычисленному значению. Найденный оператор выполняется, после чего оператор выбора завершает свою работу. Если в списке выбора не будет найдена константа, соответствующая вычисленному значению ключа выбора, управление передается операторам, стоящим за словом **ELSE**. Часть **ELSE** <оператор> можно опускать. Тогда при отсутствии в списке выбора нужной константы ничего не произойдет, и оператор выбора завершит свою работу.

Пример,

```
Program Calculator;
{Программа вводит два числа и знак операции  +, -, *, / , и выводит на экран результат  арифметического действия}
var
operation : Char; {Знак операции}
y, x, z : Real; {Операнды и результат}
ch:char;
flag:boolean;
begin
 repeat
    FLAG:=TRUE;
    ReadLn(x);
    ReadLn(operation);
    ReadIn(y);
    case operation of
       '+':   Z := X + y;
       '-':   Z := X - y;
       '*':   Z := X * y;
       '/':   Z := X / y
    else flag:=FALSE;
    end;
    if flag then writeln('otvet z=',z)
       else
       writeln('povtor!!!')
until flag
end.
```

Пример,

```
var
ch : Char;-
begin
 ReadLn(c);
 case c of
    'n','N': WriteLn('Нет');
    'y', 'У : WriteLn ('Да')
 end
end.
```

5.2. Метки и операторы перехода

Использование операторов перехода затрудняет понимание программы, делает ее запутанной и сложной в отладке. В некоторых случаях при затруднении в реализации алгоритма, использование операторов перехода разрешается.

Оператор перехода имеет вид:

GOTO <метка>.

Здесь **GOTO** - зарезервированное слово {перейти [на метку]};

<метка> - метка.

Метка - это произвольный идентификатор, позволяющий именовать некоторый оператор программы и таким образом ссылаться на него. Перед тем как появиться в программе, метка должна быть описана. В языке Турбо Паскаль допускается в качестве меток использование также целых чисел без знака. Метка располагается перед помечаемым оператором и отделяется от него двоеточием. Оператор можно помечать несколькими метками, которые в этом случае отделяются друг от друга двоеточием.

Описание меток:

```
LABEL
Label1, loop;
begin
    goto loop;    {передается управление на строку с меткой loop:}
    label1:
    writeln('метка label1');
    loop:
    writeln('метка loop');
    goto label1;  {передается управление на строку с меткой label1:}
end.
```

Метка, на которую ссылается оператор **GOTO**, должна быть описана в разделе описаний, и она обязательно должна встретиться где-нибудь в теле программы. Метки, описанные в процедуре (функции), действуют только в рамках процедуры (функции), т.е. действие таких меток локально.

5.3. Порядок проведения работы

1. Изучить теоретические сведения.
2. Разработать блок-схему алгоритма.
3. Реализовать алгоритм на Турбо Паскале.
4. Протестировать программу.
5. Ответить на контрольные вопросы.
6. Оформить отчет.

5.4. Содержание отчета

Оформить отчет, содержащий:

1. Титульный лист.
2. Тему и цель работы.
3. Задание (согласно варианту).
4. Блок – схему алгоритма.
5. Листинг программы.
6. Тестовые примеры.
7. Выводы по работе.

5.5. Контрольные вопросы

1. Какой формат записи оператора выбора case?
2. Когда необходимо использовать метки и оператор перехода?
3. Почему не рекомендуют использовать оператор безусловного перехода?

6. ЛАБОРАТОРНАЯ РАБОТА№6
СТРУКТУРИРОВАННЫЕ ТИПЫ ДАННЫХ

Цель работы: изучить основные типы структурированных данных. Тип массив.

6.1. Одномерные массивы

Простые типы данных позволяют использовать в программе одиночные объекты -

числа, символы, строки и т.п. Массив - формальное объединение нескольких однотипных объектов (чисел, символов, строк и т.п.), рассматриваемое как единое целое. Применение массивов необходимо, когда требуется связать и использовать целый ряд однотипных величин.

Типы структурированных данных:

1. Тип **string**.
2. Тип **array**.
3. Тип **record**.
4. Тип **set**.
5. Тип **file**.

При описании типа массив служебное слово, **packed**, должно обеспечить наиболее эффективное размещение в памяти постоянных этого типа. Поскольку в Турбо Паскале это и так имеет место всегда, использование данного служебного слова никакого действия не оказывает и лишь предусматривается на случай использования соответствующей программы в стандартном Паскале без внесения дополнительных изменений.

Массив – это совокупность фиксированного и упорядоченного числа одинаковых компонент. Каждую компоненту можно снабдить индексом. Для описания массива требуется задать: тип компонент, тип индекса. Тип индекса может быть только порядковым (кроме **longint**). Чаще всего используется интервальный тип (диапазон).

Массив описывается в разделе описания переменных посредством указания типа его элементов и максимального их количества. Тип элементов может быть любым. При задании максимального числа элементов массива (числа ячеек таблицы) обычно употребляют диапазон целых чисел. Общий вид описания массива следующий:

TYPE «Имя массива» =**ARRAY**[размерность] **OF** тип элементов;
VAR «идентификатор»: «имя массива»;

Пример,

```
Const n=5;
Type mas=array [1..n] of integer;
Var m:mas;
   i:integer;
Begin
   For i:=1 to n do readln(m[i]);
   Writeln;
   For i:=1 to n do writeln(m[i]);
     end.
```

Определить переменную как массив можно и непосредственно при ее описании, без предварительного описания типа массива, например:

VAR Имя массива : **ARRAY**[размерность] **OF** тип элементов ;

Пример,

```
Const n=5;
Var mas:array [1..n] of integer;
   i:integer;
Begin
   For i:=1 to n do readln(mas[i]);
   Writeln;
   For i:=1 to n do writeln(mas[i]);
     end.
```

К имени массива предъявляются те же требования, что и к имени переменной.

Размерность - это диапазон целых чисел, каждое из которых является порядковым

номером (индексом) одной из ячеек массива. Обычно диапазон индексов задают, начиная от единицы: [1..100]. Это удобно, но не обязательно. Важно лишь, чтобы правая граница диапазона была больше или равна левой границе.

При описании массивов, задавая числовые пределы изменения индексов, обычно указывают максимально возможное число элементов, которое может быть востребовано в данной программе; при этом допускается, что фактическое количество элементов может оказаться меньше, чем затребовано. Но если значение индекса элемента массива не входит в указанный при описании диапазон, это приведет к ошибке. В математике такие структуры называются векторами.

Для ссылки на отдельные элементы массива используется переменная с индексом: имя массива [индекс]. Чтобы получить доступ к конкретному элементу массива, в качестве индекса можно использовать не только целое число, соответствующее порядковому номеру этого элемента в массиве, но и выражение, значение которого равно упомянутому целому числу. Например, при обращении к элементам некоторого массива

A: **Array**[1..100] **of** Real ,

в качестве индекса можно использовать любое арифметическое выражение, значением которого будет целое число из диапазона 1..100:

A[56]; A[i+7]; A[i div j] .

Вводить и выводить значения из массивов целесообразно поэлементно, используя в цикле операторы **Readln, Write**, **Writeln** и оператор присваивания.

Например,

```
Program Primer;
Const Max=100;
Var A:Array[1..50] of Real;{массив "A" состоит из 50-ти вещественных чисел}
B:Array[1..Max] of integer;{массив "B" состоит из ста целых чисел}
Mas:Array[1999..2000] of integer;{массив "Mas" состоит из 2-х целых чисел}
Mas2:Array[-700..-1] of Real;{массив "Mas2" состоит из 700 вещественных чисел}
```

Одномерный массив соответствует понятию вектора (линейной таблицы, строки), двумерный массив соответствует понятию – матрицы (набора векторов, прямоугольной таблицы), трехмерный массив – понятию массива матриц (набору матриц, комплекту матриц).

Вводить и выводить массивы можно только поэлементно.

Пример, Ввод и вывод одномерного массива

```
const
  n = 5;
type
      mas = array[1..n] of integer;
var
      a: mas;
      i: byte;
begin
      writeln('введите элементы массива');
      for i:=1 to n do readln(a[i]);
      writeln('вывод элементов массива:');
      for i:=1 to n do write(a[i]:5);
end.
```

6.2. Многомерный массив

Так как тип, идущий за ключевым словом of в описании массива, - любой тип Турбо Паскаль, то он может быть и другим массивом. Например,

```
Type
      mas = array[1..5] of array[1..10] of integer;
```

Такую запись можно заменить более компактной:

```
Type
    mas = array[1..5, 1..10] of integer;
```

Таким образом, возникает понятие многомерного массива. Глубина вложенности массивов произвольная, поэтому количество элементов в списке индексных типов (размерность массива) не ограничена, однако не может быть более 65520 байт. Многомерный массив представляет собой чистую абстракцию, т.к. в памяти они хранятся в виде линейных последовательностей значений. Работа с многомерными массивами почти всегда связана с организацией вложенных циклов. Так, чтобы заполнить двумерный массив (матрицу) случайными числами, используют конструкцию вида:

```
for i:=1 to m do
    for j:=1 to n do a[i,j]:=random(10);
```

Для "красивого" вывода матрицы на экран используйте такой цикл:

```
for i:=1 to m do begin
    for j:=1 to n do write(a[i,j]:5);
    writeln;
end;
```

Действия над массивами

Для работы с массивом как единым целым используется идентификатор массива без указания индекса в квадратных скобках. Массив может участвовать только в операциях отношения (равно, не равно) и операторе присваивания. Массивы, участвующие в этих действиях, должны быть идентичны по структуре, т.е. иметь одинаковые типы индексов и одинаковые типы компонентов.

Операция присваивания допустима только при данном описании:

Пример,

```
const n=5;
Type mas=array [1..n] of integer;
var m:mas;i:integer;
a:mas;
begin
   for i:=1 to n do readln(m[i]);
   writeln;
   a:=m;
   for i:=1 to n do writeln(a[i]);
end.
```

Так же, будет правильным описание:

```
...
var m,a:mas;i:integer;
...
```

, либо следующим образом:

```
...
var m,a: array [1..n] of integer; i:integer;
...
```

При другом способе описания операция присваивания вызовет ошибку.

Допустимые действия над массивами представленны в Таблице 6.1

Таблица 6.1 Действия над массивами

Выражение	Результат
A:=B	Все значения элементов массива B присваиваются соответствующим элементам массива A. Значения элементов массива B остаются неизменны.

6.3. Порядок проведения работы

1. Изучить теоретические сведения.
2. Разработать блок-схему алгоритма.
3. Реализовать алгоритм на Турбо Паскале.
4. Протестировать программу.
5. Ответить на контрольные вопросы.
6. Оформить отчет.

6.4. Содержание отчета

Оформить отчет, содержащий:

1. Титульный лист.
2. Тему и цель работы.
3. Задание (согласно варианту).
4. Блок – схему алгоритма.
5. Листинг программы.
6. Тестовые примеры.
7. Выводы по работе.

6.5. Контрольные вопросы

1. Формат записи типа array?
2. Когда необходимо использовать одномерный массив?
3. Когда необходимо использовать многомерный массив,
4. Какие действия можно производить над массивом?

7. ЛАБОРАТОРНАЯ РАБОТА№7 ТИП ДАННЫХ CHAR. ТИП ДАННЫХ STRING.

Цель работы: изучить типы данных char и string. Изучить стандартные процедуры и функции работы со строками.

7.1. Тип данных CHAR

Тип данных символ (**char**). Операции над символами.

Множеством значений этого типа являются символы, упорядоченные в соответствии с расширенным набором символов кода **ASCII**.

Например,

```
Var
    Alpha : char;
Begin
    Alpha :='p';
    Alpha :='+';
    Alpha :='3';
    Alpha :=' ';
    Alpha :='';
    .....
End.
```

Первый оператор присваивания записывает в переменную Alpha символ p. Второй делает Alpha равной литере плюса (+). Третий делает Alpha равной символу 3. Заметим, что число 3 отличается от целого числа 3 тем, что оно не может быть использовано в арифметических операциях. Четвертый оператор присваивания делает Alpha равной литере пробела. Хотя литера пробела при печати не изображается, она является обыкновенным значением типа **char**. Последний оператор присваивания делает Alpha равной литере апострофа, это специальный случай, так как знак апострофа используется для ограничения значения типа **char**.

Мы будем пользоваться множеством литер, находящимся в таблице кодов, называемой **ASCII** – американский стандартный код обмена информацией. Все символы упорядочены, т.к. имеют свой личный номер. Важно, что соблюдаются следующие отношения:

'A' < 'B' < 'C' < ... < 'X' < 'Y' < 'Z' '0' < '1' < '2' < ... < '7' < '8' < '9'

Для проверки равенства или неравенства переменных типа **char** могут использоваться операторы булевого (логического) сравнения.

Например, программа, которая считывает две литеры и печатает больше, равна или меньше первая литера второй.

```
Program Sravnenie;
Var
  First, Second : char;
Begin
  write ('Введите две литеры через пробел ');
  readln (First, Second);
  write ('Первая литера ');
  if First > Second
    then
      write ('больше второй. ')
    else
      if First = Second
        then
          write ('равна второй. ')
        else
          write ('меньше второй. ')
End.
```

Так как **char** – порядковый тип, то к его значениям применимы следующие функции.

Succ – возвращает следующий символ литерного множества;

Pred – возвращает предыдущий символ литерного множества;

Ord – возвращает значение кода литеры;

Chr – возвращает значение литеры, является обратной по отношению к функции Ord.

Например,

Succ('0')='1' – символ, следующий за символом 0, равен символу 1.

Pred('3')='2' – символ, предшествующий символу 3, равен 2;

Chr(65)='A' – символ, соответствующий коду 65, равен A;

Ord('A')=65 – код символа A равен 65

7.2. Тип данных STRING

Строка (**string**) – это последовательность символов. Тип данных (**string**) определяет строки с максимальной длиной 255 символов. Переменная этого типа может принимать значения переменной длины. Строка символов представляет собой последовательность, содержащую нуль и более символов из расширенного набора символов кода **ASCII**, записанную в одной строке программы и заключенную в одиночные кавычки (апострофы). Строка символов, ничего не содержащая между апострофами, называется нулевой строкой. В качестве расширения стандартного Паскаля, Турбо Паскаль разрешает вставлять в строку символов управляющие символы. Символ # с целой константой без знака в диапазоне от 0 до 255 обозначает соответствующий этому значению символ в коде **ASCII**. Между символом # и целой константой не должно быт никаких разделителей. Аналогично, если несколько управляющих символов входит в строку символов, то между ними не должно быть разделителей.

Пример,

```
#13#10
'line 1'#13'line 2'
#7#7'Make up!'#7#7
```

Формат описания типа:

```
Type
<имя типа>=string[максимальная длина строки];
var
<идентификатор,.....>:<имя типа>;
```

Переменную типа **string** можно задать и без описания типа:

```
Var <идентификатор>:string[максимальная длина строки];
```

Тип **string** широко используется для обработки текстов. Строка трактуется как цепочка символов. К любому символу можно обратиться точно так же, как к элементу одномерного массива, например,

```
Var    st:string;
 Begin
.....
    If st[5]='F' then

end.
```

Например,

```
MaxLine : string;
City : string[30];
```

Строковая переменная может иметь атрибут длины, определяющий ее максимальную длину. Текущая длина строковой переменной может быть определена с помощью встроенной функции **Length**. Для заданного значения типа **string** эта функция возвращает целое значение, показывающее количество литер в строке. Выражения, в которых операндами служат строки, называются строковыми выражениями.

Над строками определены две операции:

1. Операция сцепления (+) применяется для сцепления нескольких строк в одну.

   Например, SumStr := 'Турбо'+'Паскаль'+'7.0'

Если длина сцепленной строки превысит максимально допустимую длину N, то 'лишние' символы отбрасываются.

2. Операции отношения (=, <>, >, <, >=, <=) проводят сравнение двух строк (посимвольно, с учетом внутренней кодировки символов) слева направо до первого несовпадающего символа, и та строка считается больше, в которой первый несовпадающий символ имеет больший номер в стандартной таблице обмена информацией. Результат выполнения операций отношения над строками всегда имеет булевой тип.

   Например, выражение 'MS-DOS'<'MS-Dos' имеет значение True

Если строки имеют различную длину, но в общей части символы совпадают, считается, что более короткая строка меньше, чем более длинная. Строки считаются равными, если они совпадают по длине и содержат одни и те же символы на соответствующих местах в строке.

Для присваивания строковой переменной результата строкового выражения используется оператор присваивания. Если значение переменной после выполнения оператора присваивания превышает по длине максимально допустимую при описании величину, то все лишние символы справа отбрасываются. Допускается смешение в одном выражении операндов строкового и символьного типа. К отдельным символам строки можно обратиться по номеру (индексу) данного символа в строке. Например, чтобы обратиться

к третьему символу строки SumStr, надо записать SumStr[3]. Запись SumStr[0] дает значение текущей длины строки.

Для эффективного программирования алгоритмов обработки текстов необходимо хорошо понимать внутреннюю структуру представления строк в памяти. Строки реализованы достаточно просто. Для хранения строковых переменных выделяется память, на единицу большая максимальной длины строки. Начальный байт этой памяти отводится для хранения текущей длины строки, следующие байты - для символов самой строки. Так как элементы строк стандартно нумеруются целыми числами, начиная с единицы, байт с длиной строки можно считать нулевым ее элементом. Такая структура памяти допускает прямой доступ к ее элементам.

Важно отметить, что имеется возможность динамически управлять текущей длиной строки. Следующая программа показывает автоматическое изменение длины строки после тех или иных операций с нею. Обратите внимание, что общий (определяемый с помощью стандартной функции SizeOf размер памяти, отведенной для хранения строки все время остается неизменным.

```
Program StringLength;
Var
  S : string;     {макс. длина строки = 255}
Begin
  S:='';          {пустая строка}
  writeln (S,' ',SizeOf(S),' ',Length(S));  {размер=256, длина=0}
  S:='Пример длинной строки';    {присваиваем строке некоторое значение}
  writeln (S,' ',SizeOf(S),' ',Length(S));  {размер=256, длина=21}
  Delete(S,7,8);                {удаляем из строки 8 символов, начиная с 7}
  writeln (S,' ',SizeOf(S),' ',Length(S));  {размер=256, длина=13}
  S:=S+' символов';             {добавляем к строке строку}
  writeln (S,' ',SizeOf(S),' ',Length(S));  {размер=256, длина=22}
End.
```

Внимание! При решении задач со строковыми переменными Вы можете столкнуться с распространенной трудноуловимой ошибкой, когда после присваивания некоторым элементам строки символов ни содержимое, ни длина строки не изменяются. Очень важно понимать, что при доступе к некоторому элементу строки значение ее текущей длины не проверяется. Это иллюстрирует следующая программа:

```
Program StringElements;
Var
  S : string;          {макс. длина строки = 255}
Begin
  S:='ABCD';           {инициализация строки}
  writeln (S,' ',Length(S));{вывод строки и ее длины}
  S[5] := 'E';         {присваивание элементу строки}
  writeln (S,' ',Length(S));{ни сама строка, ни ее длина не изменились}
End.
```

Присваивание пятому элементу строки некоторого значения не изменяет длину строки, что подтверждает вывод на экран ее содержимого и длины (конечно, само присваивание реально произошло, но на значение текущей длины строки в нулевом байте это никакого влияния не оказало). Работа с элементами строки без учета ее текущей длины и является ошибкой программиста. Посмотрите следующую программу:

```
Program StringElements2;
Var
  Str : string[26];     {длина строки = 26}
  i : integer;
```

```
Begin
  Str:='A';
  for i := 1 to 26 do
     Str[i] := Chr (Ord('A')+i-1);
  writeln(Str);
End.
```

Предполагается, что данная программа должна сформировать строку из 26 символов, содержимым которой является последовательность заглавных букв латинского алфавита. Однако вызов процедуры writeln показывает, что содержимым переменной Str 1 будет строка из одного символа 'A'. Природа совершенной ошибки заключается в том, что присваивание значений элементам строки не влияет не текущую длину, которая была установлена равной 1 при первом присваивании. Поэтому правильной будет следующая программа:

```
Program Elements3;
Var
  Str1 : string[26];{длина строки = 26}
  i : integer;
Begin
  S:='';
  for i := 'A' to 'Z' do
     Str1 := Str1 + i;
  writeln(Str1);
End.
```

Операция конкатенации, как и все стандартные операции, работающие со строками, в отличие от поэлементного присваивания, изменяет длину строки, что дает корректный результат. Кроме того, вторая программа работает непосредственно с символами букв. Наконец, не следует забывать инициализировать строку перед ее заполнением (первый оператор программы). В противном случае, так как начальная длина строки является неопределенной, можно получить произвольный результат; не стоит рассчитывать на то, что в нулевом байте стоит нуль.

Для обработки строковых данных можно использовать встроенные процедуры и функции:

1. **Delete** (Str1,Poz,N) – удаление N символов строки Str1, начиная с позиции Poz.
2. **Insert** (What,Where,Poz) – вставка строки What в строку Where, начиная с позиции **Poz**.
3. **Copy** (Str1,Poz,Nstr) – выделяет строку длиной Nstr, начиная с позиции Poz, из строки Str1.
4. **Concat** (Str1,Str2,...,StrN) – выполняет сцепление строк в том порядке, в каком указаны в списке параметров.
5. **Poz** (What,Where) – обнаруживает первое появление подстроки What в строке Where.
6. **UpCase** (Ch) – преобразует строчную букву в прописную.
7. **Str** (Number,Stroka) – преобразует число в строку.
8. **Val** (Stroka,Number,Code) – преобразует строку в число и выдает код правильности преобразования.

Функция Length

Встроенная функция **Length** (длина) позволяет определить фактическую длину текстовой строки, хранящейся в указанной переменной (а не величину предельного размера строки, установленную при декларации):

```
Program DemoFunctionLength;
Var
    Word : string;
Begin
    write ('Введите слово :');
```

```
    readln(Word);
    writeln('Это слово состоит из ',Length (Word),' букв');
End.
```

Примечание. При подсчете фактической длины строки учитываются все входящие в нее символы, в том числе и пробелы.

Функция Upcase

Функция **Upcase** позволяет преобразовывать символ любой литеры из строчного в прописной. Эта функция рассчитана на обработку отдельного символа. Поэтому для обработки строки символов с помощью этой функции приходится организовывать цикл.

```
Program DemoFunctionUpcase;
Var
    Word : string;
    i : Byte;
Begin
    Word := 'фирма Microsoft';
    for i := 1 to Length (Word) do
        Word[i] := UpCase (Word[i]);
    writeln(Word); {выводится текст 'фирма MICROSOFT'}
End.
```

В результате работы программы на терминал выдается строка, содержащая большие английские буквы и маленькие русские.

Примечание. Русские литеры не могут обрабатываться этой функцией.

Для того, чтобы преобразовать в заглавные строчные буквы русского алфавита, применяют оператор выбора **Case**:

```
case Word[i] of
    'а' : Word[i] := 'А';
    'б' : Word[i] := 'Б';
    'в' : Word[i] := 'В';
end.
```

Функция Copy

Функция **Copy** позволяет копировать фрагмент некоторой строки из одной переменной в другую. Вызывая эту функцию, нужно указать следующие параметры:

- имя строки, из которой должен извлекаться копируемый фрагмент,
- позицию в строке, начиная с которой будет копироваться фрагмент,
- число копируемых символов.

```
Program DemoFunctionCopy;
Var
    Word 2: string;
    Word1 : string[20];
Begin
    Word 2:= 'фирма Microsoft';
    writeln(Word); {выводится текст 'фирма MICROSOFT'}
    Word1 := Copy (Word2,1,5);
    writeln(Word1); {выводится текст 'фирма'}
End.
```

Примечание. Если начальная или конечная позиции копируемого текста находятся вне пределов исходной строки символов, то сообщение об ошибке не выдается. Результатом выполнения операции в первом случае будет строка нулевой длины, во втором - фрагмент от начальной позиции копирования до конца исходной строки.

Функция Pos

С помощью функции **Pos** можно осуществить поиск некоторого фрагмента в строке. Если заданный фрагмент в строке присутствует, то функция возвращает номер позиции, с которой он начинается. Если фрагмент не найден, то функция возвращает нуль.

```
Program DemoFunctionPos;
Var
    Word : string;
    SearchWord : string[20];
    Position : Byte;
Begin
    Word := 'фирма Microsoft';
    writeln(Word); {выводится текст 'фирма MICROSOFT'}
    writeln ('Введите искомый текст ');
    readln (SearchWord);
    Position := Pos(SearchWord, Word);
    if Position <> 0
      then
      begin
        write ('Фрагмент <',SearchWord,'> содержится в строке <',Word);
        writeln ('>, начиная с позиции ',Position );
      end
    else
      writeln('Фрагмент <',SearchWord,'> не содержится в строке <',Word);
End.
```

Примечание. Функция **Pos** требует полного совпадения искомого фрагмента и фрагмента строки, в которой производится поиск. Причем большие и маленькие буквы считаются различными символами.

Функция Concat

Функция **Concat** (Str1,Str2,...,StrN) выполняет конкатенацию (или сцепление) строк Str1,Str2,...,StrN в том порядке, в каком они указаны в списке параметров. Сумма символов всех сцепленных строк не должна превышать 255.

```
Program DemoFunctionConcat;
Var
    Word3 : string;
    Word1, Word2 : string[20];
Begin
    Word1 := 'фирмы ';
    Word2 := 'Microsoft';
    Word3 := Concat('Компьютеры ',Word1,Word2);
    writeln(Word3); {выводится текст 'Компьютеры фирмы Microsoft'}
End.
```

Процедура Insert

Процедура **Insert** вставляет в исходную строку, начиная с указанной позиции, какую-либо другую строку. Оператор **Insert** (Word1,Word2,5) указывает, строку Word1 необходимо вставить в строку Word2, начиная с 5-ой позиции.

Процедура Delete

Процедура **Delete** удаляет в исходной строке фрагмент определенной длины, начиная с указанной позиции. Так, оператор **Delete**(Word1,2,3) удаляет из указанной строки фрагмент, длиной в три символа, начиная со второго.

Процедура Str

Общий вид **Str**(Chislo,Stroka)

Процедура **Str** преобразовывает числовое значение переменной Chislo в строковую переменную Stroka. После первого параметра может указываться формат, аналогичный формату вывода.

```
Program DemoProcedureStr;
Var
    Word1 : string;
    Chislo : integer;
Begin
    Chislo := 1560;
    Str(Chislo:8, Word);
    writeln(Word1); {выводится строка '    1500'}
End.
```

Процедура Val

Общий вид **Val**(Stroka,Chislo,Code)

Процедура **Val** преобразует значение строки Stroka в величину целочисленного или вещественного типа и помещает результат в Chislo. Значение строковой переменной Stroka не должно содержать пробелов в начале и в конце. Code целочисленная переменная. Если во время операции преобразования ошибки не обнаружено, значение Code равно нулю, если же ошибка обнаружена, Code будет содержать номер позиции первого ошибочного символа, а значение Chislo будет не определено.

```
Program DemoProcedureVal;
Var
    Word1 : string;
    Chislo, Code : integer;
Begin
    writeln('Введите строку цифр ');
    readln(Word1);
    Val(Word1, Chislo, Code); {преобразование строки в число}
    if Code <> 0
       then
          writeln('Ошибка! В позиции ',Code,' не цифра!',);
End.
```

7.3. Порядок проведения работы

1. Изучить теоретические сведения.
2. Разработать блок-схему алгоритма.
3. Реализовать алгоритм на Турбо Паскале.
4. Протестировать программу.
5. Ответить на контрольные вопросы.
6. Оформить отчет.

7.4. Содержание отчета

Оформить отчет, содержащий:

1. Титульный лист.
2. Тему и цель работы.
3. Задание (согласно варианту).
4. Блок – схему алгоритма.
5. Листинг программы.
6. Тестовые примеры.
7. Выводы по работе.

7.5. Контрольные вопросы

1. В чем особенность типа данных char?
2. В чем особенность типа данных string?
3. Совместимы ли данные типа char и данные типа char?
4. Какие стандартные процедуры и функции для работы со строками реализованы в Турбо Паскале?

8. ЛАБОРАТОРНАЯ РАБОТА №8
СТРУКТУРИРОВАННЫЕ ТИПЫ ДАННЫХ. ТИП ДАННЫХ МНОЖЕСТВО

Цель работы: изучить структурированный тип set.

8.1. Тип данных множество (set)

Множественный тип данных напоминает перечислимый тип данных. Вместе с тем, множество – набор элементов, не организованных в порядке следования. В математике множество – любая совокупность элементов произвольной природы. Понятие множества в программировании значительно уже математического понятия. Под множеством в Паскале понимается конечная совокупность элементов, принадлежащих некоторому базовому типу. В качестве базовых типов могут использоваться: перечислимые типы данных, символьный и байтовый типы или диапазонные типы на их основе.

Такие ограничения связаны с формой представления множества в языке и могут быть сведены к тому, что функция **Ord** для используемого базового типа должна быть в пределах от 0 до 255.Множество имеет зарезервированное слово **set of** и вводится следующим описанием

```
Type
  < имя типа > = set of < имя базового типа >;
Var
  < идентификатор,... >:< имя типа >;
```

Либо без объявления типа:

```
Var
  < идентификатор > = set of < имя базового типа >;
```

Примеры описания множеств:

```
Type
SetByte = set of byte;     {множество 1, определённое над типом byte}
SetChisla = set of 10 ... 20; {множество 2, определённое в диапазоне от 10 до 20
Symbol = set of char; {множество, определённое на множестве символов}
Month = (January, February, March, April, May, June, July, August, September, October,
November, December);
Season: set of Month; {тип множества, определённый на базе перечислимого типа Month}
  Var
Letter, Digits, Sign : Symbol {множества, определённые над символьным типом}
Winter, Spring, Summer, Autumn, Vacation, WarmSeason : Season;
Index : SetChisla=[12, 15, 17];
Operation : set of (Plus, Minus, Mult, Divid);
Param : set of 0..9=[0, 2, 4, 6, 8];
```

Для переменных типа множества в памяти отводится по 1 биту под каждое возможное значение базового типа. Так, под переменные Letter, Digits, Sign будет отведено по 256/8=32 байта. Для переменной Winter, базовый тип которой (Month) имеет 12 элементов, необходимо 2 байта, причем второй используется только наполовину. Если множество содержит какой-то элемент, то связанный с ним бит имеет значение 1, если нет – 0.

Для того, чтобы дать переменной множества какое-то значение, используют либо конструктор множества – перечисление элементов множества через запятую в квадратных скобках

```
Sign:=['+', '–'];
Spring:=[March, April, May];
b:=[ 'k', 'l', 'd' ] ,
```

либо определение через диапазон. Тогда в множество включены все элементы диапазона

```
Digits:='0'..'9'];
WarmSeason := [May .. September];
```

Обратите внимание, что в определении множества Digits использованы символы в таблице **ASCII**-кодов, а не целые числа.

Обе формы конструирования могут сочетаться:

```
Vacation:=[January, February, June .. August];
```

В программах множества часто используются как константы, в этом случае их можно определить следующим образом:

```
{постоянное множество допустимых символов}
Const
   YesOrNo = ['Y', 'y', 'N', 'n']; {множество – типизированные константы}
Const
   Digits : set of char=['0'..'9'];
   DigitsAndLetter : set of char=['0'..'9', 'a'..'z', 'A'..'Z'];
{применение операции "+" для объявления множества-константы}
Const
   Yes = ['Y', 'y'];
   No = ['N', 'n'];
   YesOrNo = Yes+No;
```

8.2. Действия над множествами

Объединение множеств (+)

Определение. Объединением 2-х множеств называется третье множество, которое содержит элементы, которые принадлежат хотя бы одному из множеств операндов, при этом каждый элемент входит в множество только один раз.

Объединение множеств записывается как операция сложения.

```
Type
  Symbol = set of char;
Var
   SmallLatinLetter, CapitalLatinLetter, LatinLetter : Symbol;
Begin
   SmallLatinLetter :=['a'..'z'];
   CapitalLatinLetter := ['A'..'Z'];
   LatinLetter := SmallLatinLetter+CapitalLatinLetter;
End.
```

В операции объединения множеств могут участвовать и отдельные элементы множества. Например, допустима следующая запись, где два элемента и множество объединяются в новое множество:

```
WarmSeason := May+Summer+September;
```

или другая запись `B: = B+['c'],`

которую можно применить для организации множества в цикле, если заменить множество ['c'] переменной Sim того же типа, что и множество B, и считывать с клавиатуры данные в переменную Sim, а затем объединяя с множеством B.

```
B: = B+Sim,
```

Разность множеств (-)

Определение. Разностью 2-х множеств является третье множество, которое содержит элементы 1-го множества, не входящие во 2-е множество.

a: = a–['d']

Если в вычитаемом множестве есть элементы, отсутствующие в уменьшаемом, они не влияют на результат.

Summer := WarmSeason–Spring–Autumn;

Summer := WarmSeason–May–September;

Модуль System содержит процедуры для включения элемента в множество

Include(**Var** S : **set of** T; Element : T);

и исключения из множества

Exclude(Var S : **set of** T; Element : T); ,

где S – множество элементов типа T, а Element – включаемый элемент.

Эти функции отличаются от операций объединения и вычитания множеств только скоростью исполнения.

Пересечение множеств

Определение. Пересечением множеств называется множество, содержащее элементы, одновременно входящие в оба множества операндов. Операция обозначается знаком умножения.

Summer := WarmSeason*Vacation;

Сравнение множеств

Определение. Множества считаются равными, если все элементы, содержащиеся в одном множестве, присутствуют в другом, и наоборот.

В соответствии с этим правилом определяется результат логических операций "=" и "<>".

Например,

```
If WarmSeason*Vacation=Summer
   Then
     WriteIn ('Правильно');
```

Проверка включения

Определение. Одно множество считается входящим в другое, если все элементы содержатся во втором, при этом обратное в общем случае может быть несправедливо.

Логические операции проверки вхождения одного множества в другое записываются через операции больше или равно:

```
if S1<=S2
   then
      writeln ('S1 входит в S2');
if S1>=S2
   then
      writeln ('S2 входит в S1');
```

Проверка принадлежности

Логическая операция проверки принадлежности элемента множеству записывается через оператор **in**.

Например, выражение May **in** WarmSeason имеет значение **True**.

Использование множеств и операции in позволяет, в частности, сделать эффективнее проверку правильности вводимых символов.

Например, для проверки допустимости введенного символа можно использовать следующее условие:

(Reply='y') **or** (Reply='Y') **or** (Reply='n') **or** (Reply='N')

Но если ввести множество

Const

AllowSymbol : **set of char** = ['Y', 'y', 'N', 'n'];

проверяемое условие можно записать в более компактной форме:

Reply **in** AllowSymbol

Примечание. Множественный тип данных не может быть использован для определения функции.

Дополнительно к этим операциям можно использовать две процедуры.

INCLUDE- включает новый элемент во множество. Обращение к процедуре:

INCLUDE (S, I)

Здесь S- множество, состоящее из элементов базового типа TSetBase;

I-элемент типа TSetBase, который необходимо включить во множество;

EXCLUDE - исключает элемент из множества. Обращение:

EXCLUDE (S, I)

Параметр обращения – такие же, как у процедуры INCLUDE.

Нет возможности считать или записать переменные типа множество через **writeln**, **write**, **readln**, **read**. При попытке возникает ошибка: 64 Cannot Read or Write variables of this type! Так как процедуры **Read**, **Readln** могут считывать переменные символьного, целого, действительного и строкового типа и процедуры **Write**, **Writeln** могут выводить переменные символьного, целого, действительного, булевского, строкового типов.

8.3. Порядок проведения работы

1. Изучить теоретические сведения.
2. Разработать блок-схему алгоритма.
3. Реализовать алгоритм на Турбо Паскале.
4. Протестировать программу.
5. Ответить на контрольные вопросы.
6. Оформить отчет.

8.4. Содержание отчета

Оформить отчет, содержащий:

1. Титульный лист.
2. Тему и цель работы.
3. Задание (согласно варианту).
4. Блок – схему алгоритма.
5. Листинг программы.
6. Тестовые примеры.
7. Выводы по работе.

8.5. Контрольные вопросы

1. В чем особенность типа данных set?
2. Какие действия допустимы над типом set?
3. Как реализован ввод\вывод данных во множество?
4. Какие стандартные процедуры и функции для работы с множествами реализованы в Турбо Паскале?

9. ЛАБОРАТОРНАЯ РАБОТА №9
СТРУКТУРИРОВАННЫЕ ТИПЫ ДАННЫХ. ТИП ДАННЫХ ЗАПИСЬ.

Цель работы: изучить особенности структурированного типа данных record.

9.1. Тип данных запись

Довольно часто вполне оправданным является представление некоторых элементов в качестве составных частей другой, более крупной логической единицы. Представляется естественным сгруппировать информацию о номере дома, названии улицы и городе в единое целое и назвать адресом, а объединенную информацию о дне, месяце и годе

рождения – датой. В языке Паскаль для представления совокупности разнородных данных служит комбинированный тип запись.

Запись и массив схожи в том, что обе эти структуры составлены из ряда отдельных компонент. В то же время, если компоненты массива должны быть одного типа, записи могут содержать компоненты разных типов.

Структура объявления типа записи такова:

```
Type
«имя типа» = record  «список полей»
End  ,
```

где «имя типа» - идентификатор;

Record , end –зарезервированные поля;

«список полей» - последовательность разделов записей, между которыми ставится точка с запятой.

Приведем пример описания переменной, имеющей структуру записи:

```
Var
 Address : Record
    HouseNumber : Integer;
    StreetName : String[20];
    CityName : String[20];
    PeopleName : String;
 End;
```

Отметим, что поля StreetName и CityName имеют одинаковый тип: String[20]. Поскольку в описании эти поля могут располагаться в любом порядке, то можно сократить описание записи с полями одинакового типа. Сокращенное описание записи Address выглядит следующим образом:

```
Var
 Address : Record
    HouseNumber : Integer;
    StreetName, CityName : String[20];
    PeopleName : String;
 End;
```

Каждая компонента записи называется полем. В переменной записи Address поле с именем HouseNumber само является переменной типа Integer, поле StreetName – двадцатисимвольной строкой и т.д.

Для того чтобы обратиться к некоторому полю записи, следует написать имя переменной и имя поля. Эти два идентификатора должны разделяться точкой.

Оператор, который присваивает полю HouseNumber значение 45, выглядит так:

```
Address.HouseNumber := 45;
```

Таким же образом присваиваются значения другим полям записи Address :

```
Address.StreetName := 'Профсоюзная';
Address.CityName := 'Сургут';
Address.PeopleName := 'Петрова Алла Ивановна';
```

Каждое поле записи Address можно рассматривать как обычную переменную, которую можно напечатать или использовать в расчетах. Вместе с тем запись может использоваться как единое целое. В этом случае надо ввести тип записи.

Предположим, имеется следующее описание:

```
Type
 Date = Record
    Day : 1..31;
    Month : (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec);
    Year : Integer;
 End;
```

```
Var
  HisBirth, MyBirth : Date;
```

После приведенного описания переменные HisBirth и MyBirth имеют тип записи Date. Помимо действий над отдельными полями записей HisBirth и MyBirth можно выполнять операции над всей записью. Следующий оператор присваивания устанавливает равенство значений записей HisBirth и MyBirth :

```
HisBirth := MyBirth;
```

Это присваивание эквивалентно следующей последовательности операторов:

```
HisBirth.Day := MyBirth.Day;
HisBirth.Month := MyBirth.Month;
HisBirth.Year := MyBirth.Year;
```

Для переменных одного типа можно проверить выполнение отношения равенства или неравенства ("=", "<>"). После выполнения приведенных выше присваиваний следующее булево выражение будет иметь значение True:

```
HisBirth = MyBirth;
```

Рассмотрите пример описания процедуры, которая получает запись в качестве параметра-значения и печатает дату в сокращенной стандартной форме, используя формат (MM-DD-YYYY).

```
Procedure WriteDate(OneDate : Date);
Begin
   Write(Ord(OneDate.Month)+1);
   Write('-');
   Write(OneDate.Day:2);
   Write('-');
   Write(OneDate.Year:4);
End;
```

Так как на тип компонент массива не накладывается ограничений, то можно использовать массив, компонентами которого являются записи. Посмотрите описание такого массива:

```
Var
  Birthdays : Array [1..Persons] of Date;
```

Чтобы обратиться к некоторому полю определенной записи массива, следует определить имя массива, индекс интересующей записи и имя необходимого поля.

Например, следующий оператор печатает содержимое поля Year записи Birthdays[3]:

```
  Write(Birthdays[3].Year);
```

Примечание. Поля записи в свою очередь тоже могут быть массивами, множествами, записями.

Соблюдение всех правил перечисления индексов и имен полей при составлении ссылок является довольно утомительным занятием, часто приводящим к ошибкам. В некоторых программах, содержащих большое количество обращений к одному и тому же полю, такое положение приводит к однообразному повторению. Чтобы облегчить выполнение многократных ссылок для описанных структур вводится оператор **With** (в переводе с английского – предлог "с").

Общая форма записи:

with <имя переменной> **do** <оператор>

В рамках оператора, определяемого внутри оператора **With**, к полям определяемой переменной можно обращаться просто по имени. Например,

```
  With c.bd do month:=9;
```

Оператор with позволяет более компактно представлять часто используемые переменные. Посмотрите это на примере фрагмента программы, печатающего адрес рабочего № 14:

```
with Payroll[14].Residence do
 begin
    writeln(HouseNumber,' ',StreetName);
    writeln(CityName,',',StateName,',',ZipCode);
 end;
```

В рамках составного оператора, следующего за with, каждое обращение к имени поля автоматически связывается с записью Payroll[14].Residence. Печать адресов всех рабочих выполняется при помощи следующего оператора цикла:

```
for i := 1 to Workers do
 with Payroll[i].Residence do
 begin
    writeln(HouseNumber,' ',StreetName);
    writeln(CityName,',',StateName,',',ZipCode);
 end;
```

Операторы **with** могут быть вложенными. Приведенные ниже три оператора эквивалентны друг другу:

```
1. Payroll[i].Residence.HouseNumber := 50;
2. with Payroll[i].Residence do  HouseNumber := 50;
3. with Payroll[i] do
     with Residence do
     HouseNumber := 50;
```

Однако недопустимым является использование вложенных операторов **With**, в которых указываются поля одного типа, поскольку возникает неоднозначность конструкции. По этой причине приведенное использование вложенных операторов **With** является неверным:

```
with Payroll[5] do
 with Payroll[17]do
    PayScale :='A';
```

Следует очень внимательно подходить к использованию вложенных операторов **With**, применение которых не только может привести к ошибкам, но также к потере наглядности структуры программы.

9.2. Записи с вариантами

Записи, рассмотренные выше – это записи с фиксированными частями. Они имеют в различных ситуациях строго определенную структуру. Соответственно записи с вариантами в различных ситуациях могут иметь различную структуру.

Предположим, что написана программа для введения списка библиографических ссылок. Если известно, что все входы в этом списке – ссылки на книги, то можно использовать следующее описание:

```
Const
   Kol = 1000;
Type
   Entry = Record
      Autor, Title, Publisher, City : String;
      Year : 1..2000;
  End;
Var
   List : Array[1..Kol] of Entry;
```

Что произойдет, если некоторые из входов не являются ссылками на книги, а содержат ссылки на журнальные статьи. Если ограничиваться только записями с фиксирован-

ными частями, то следует описать различные массивы для каждого вида записей. Использование записей с вариантами позволяет образовать структуру, каждый вход которой соответствует содержанию записи. Опишем новый тип, в котором перечислены различные входы:

```
Type
  EntryType = (Book, Magazine);
  Теперь можно привести скорректированное описание Entry
Type
  Entry = Record
     Autor, Title : String;
     Year : 1..2000;
     Case EntryType of
        Book : (Publisher, City : String);
        Magazine : (MagName : String,Volume, Issue : Integer)
     End;
```

Это описание делится на две части: фиксированную и вариантную. Поля Autor, Title, Year составляют фиксированную часть. Оставшаяся часть описания Entry образует вариантную часть, структура которой, подобно хамелеону, может меняться в пределах двух альтернативных определений.

Первая строка вариантной части представляет оператор **Case**, который отличается тем, что в качестве селектора применяется идентификатор типа. Значения EntryType используются в качестве имен двух альтернатив определения записи. Когда эта компонента имеет значение Book, можно обращаться к следующим полям:

```
Autor, Title, Year, Publisher, City
```

С другой стороны, когда она принимает значение Magazine, то можно обращаться к таким полям:

```
Autor, Title, Year, MagName, Volume, Issue
```

В такой ситуации возникает естественный вопрос: как программа может хранить информацию о текущем состоянии каждой записи? Другими словами, каким образом можно узнать, что List[3] содержит ссылку на книгу, а List[4] – ссылку на журнал?

Естественное решение этой проблемы заключается в добавлении в каждой записи нового поля, называемого полем тега. Язык Паскаль позволяет за счет совмещения задать описание поля тега в сокращенной форме:

```
Type
  Entry = Record
     Autor, Title : String;
     Year : 1..2000;
     Case TAG : EntryType of
     Book : (Publisher, City : String);
     Magazine : (MagName : String,Volume, Issue : Integer)
  End;
```

Поле, названное TAG, является переменной типа EntryType. Когда запись содержит ссылку на книгу, TAG следует присвоить значение Book. Когда запись содержит ссылку на журнал, TAG следует присвоить значение Magazine.

Рассмотрите последовательность операторов, где в RefList[12] помещается ссылка на книгу:

```
RefList[12].TAG := Book;
RefList[12].Autor := 'Thomas Hobbes';
RefList[12].Title := 'Leviathan';
RefList[12].Year := 1651;
RefList[12].Publisher := 'Andrew Crooke';
RefList[12].City := 'London';
```

Для определения состояния записи с вариантами достаточно проверить значение поля тега. Рассмотрите процедуру, выводящую на экран переданную ей запись.

```
Procedure PrintRef(Citation : Entry);
Begin
    Writeln(Citation.Autor);
    Writeln(Citation.Title);
    Writeln(Citation.Year);
    If Citation.TAG = Book
       Then
          Writeln(Citation.Publisher,', ',Citation.City)
       Else
          Begin
          Writeln(Citation.MagName);
          Writeln(Citation.Volume'–',Citation.Issue)
       End;
End;
```

Вариантная часть может содержать произвольное число альтернатив. Хотя перечисляемые типы предпочтительнее, так как они более понятны, тем не менее, для именования альтернатив записи с вариантами могут использоваться идентификаторы произвольного порядкового типа.

Очевидно, что один и тот же идентификатор поля не может дважды использоваться при описании записи, даже если он применяется в определении различных альтернатив записи с вариантами. Если же это условие не выполняется, то обращение к такому идентификатору приведет к непредсказуемому результату. Описание записи с вариантами может иметь единственный закрывающий оператор **End**. Поскольку любая запись может иметь лишь одну вариантную часть, то **End**, который является индикатором конца описания записи, служит для обозначения конца и ее вариантной части.

9.3. Порядок проведения работы

1. Изучить теоретические сведения.
2. Разработать блок-схему алгоритма.
3. Реализовать алгоритм на Турбо Паскале.
4. Протестировать программу.
5. Ответить на контрольные вопросы.
6. Оформить отчет.

9.4. Содержание отчета

Оформить отчет, содержащий:

1. Титульный лист.
2. Тему и цель работы.
3. Задание (согласно варианту).
4. Блок – схему алгоритма.
5. Листинг программы.
6. Тестовые примеры.
7. Выводы по работе.

9.5. Контрольные вопросы

1. В чем особенность типа данных record?
2. Когда рекомендуется использовать тип record?
3. Что такое вариантная часть в записи?
4. Из каких составных частей состоит вариантная часть записи?
5. Какие ограничения накладываются на вариантную часть записи?
6. Когда необходимо использовать оператор With?

ЛИТЕРАТУРА

1. Немюгин С.А. Turbo Pascal. Программирование на языке высокого уровня: Учебник для вузов. 2-е изд. - СПб.: Питер, 2006.
2. Рапаков Г.Г. Ржеуцкая С.Ю. Turbo Pascal для студентов и школьников. - СПб: БХВ - Петербург, 2005.

Учебное издание

Составитель: Хацкевич Мария Викторовна

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
по выполнению лабораторных работ по курсу
«Основы алгоритмизации и программирования в традиционных
и интеллектуальных компьютерах»
для студентов специальности I - 40 03 01 «Искусственный интеллект»

ЧАСТЬ 1

Ответственный за выпуск: ***Хацкевич М.В***
Редактор: ***Строкач Т.В.***
Компьютерная вёрстка: ***Кармаш Е.Л.***
Корректор: ***Никитчик Е.В.***

Подписано в печать 10.06.2008 г. Формат 60x84 1/16. Бумага «Снегурочка». Усл.п.л. 2,56. Уч.-изд.л. 2,75. Тираж 60 экз. Заказ №633. Отпечатано на ризографе УО "Брестский государственный технический университет". 224017, Брест, ул. Московская, 267