Учреждение образования

«Белорусский государственный университет культуры и искусств»

Факультет культурологии и социокультурной деятельности

Кафедра информационных технологий в культуре

СОГЛАСОВАНО
Заведующий кафедрой

_____ П.В. Гляков
« 29 »мая 2017 г.

СОГЛАСОВАНО
Декан факультета

_____ Н.Н. Королев
« 30 » мая 2017 г.

УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС
ПО УЧЕБНОЙ ДИСЦИПЛИНЕ

ПРОГРАММНО-ТЕХНИЧЕСКИЕ СРЕДСТВА: АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ

для специальности 1–21 04 01 Культурология,
направление специальности 1–21 04 01–02 Культурология (прикладная),
специализации 1–21 04 01–02 04 Информационные системы в культуре

2 семестр обучения

Составитель:

П.В. Гляков, заведующий кафедрой информационных технологий в культуре учреждения образования “Белорусский государственный университет культуры и искусств”

Рассмотрен и утвержден
на заседании Совета университета 20.06. 2017 г., протокол № 10.

Составитель:

Гляков Петр Владимирович, заведующий кафедрой информационных технологий в культуре учреждения образования «Белорусский государственный университет культуры и искусств», кандидат физико-математических наук, доцент

Рецензенты:

*кафедра*веб-технологий и компьютерного моделирования Белорусского государственного университета;

В.В. Нешитой, профессор кафедры информационныхресурсов Белорусского государственного университетакультуры и искусств, доктор технических наук,профессор

Рассмотрен и рекомендован к утверждению:

Кафедрой информационных технологий в культуре
(протокол от 29.05.2017 г.,№ 9);

Советом факультета культурологии и социокультурной деятельности
(протокол от 30.05.2017 г., № 9)

Оглавление

1 ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Дисциплина «Алгоритмизация и программирование» является составной частью интегрированной дисциплины «Программно-технические средства», в которую кроме нее входят такие дисциплины, как «Компьютерная техника» и «Системное программное обеспечение». В соответствии с учебными планами она предназначена для изучения студентами высших учебных заведений по специальности 1-2104 01 Культурология (по направлениям) направления специальности 1-21 04 01-02 Культурология (прикладная)специализации 1-21 04 01-02 04 Информационные системы в культуре.

Эта дисциплина связана с такими дисциплинами, как «Основы теории информации и криптологии», «Прикладная математика».

Цель изучения дисциплины «Алгоритмизация и программирование» – формирование знаний и умений в области алгоритмизации, технологий проектирования алгоритмов, методов построения и разработки программ.

Основными *задачами* дисциплины являются:

– знакомство с основами теории сложности алгоритмов;

– изучение технологий проектирования и разработки программ;

– изучение системы программирования Pascal ABC;

– приобретение умений разрабатывать стандартные алгоритмы и программы.

В качестве базового языка для изучения основ программирования выбрана система Pascal ABC, поскольку она призвана осуществить постепенный переход от простейших программ к модульному, объектно-ориентированному, событийному и компонентному программированию. Некоторые языковые конструкции в системе Pascal ABC допускают упрощенное использование, что является очень важным на ранних этапах обучения. Ряд модулей системы программирования Pascal ABC создавался специально для учебных целей.

Учебный материал излагается на основе современных методических требований с учетом педагогических целей на уровнях представления, понимания, знания, применения и творчества. При чтении лекций особое внимание уделяется рассмотрениюпримеров, иллюстрирующих то или иное понятие, приводятся различные способы интерпретации понятий.

При обучении применяется такой метод, когда сначала выделяются классы задач и внутри этих классов задач рассматриваются типичные методы их решения. При этом сначала строятся схемы решения таких классов задач на основе классических структур управления, а потом обсуждается их

программирование, когда свойства объявляются при помощи разнообразных структур данных.

Обучение ведется на основе:

– выделения элементарных операций при построении типичныхалгоритмов обработки простых данных, структурированных статичных идинамических данных;

– одинаковой формы записи алгоритма для решения задач содинаковой структурой исходных данных;

– выделения вспомогательных алгоритмов, которые потом оформляютсяподпрограммами языка и могут объединяться в модули.

Лабораторные занятия направлены на формирование умений практического использования полученных знаний при разработке алгоритмов и программ для решения конкретных задач. Методика их проведения содействует развитию творческих способностей каждого студента и приобретению навыков самостоятельной работы. Используются такие новые формы активизации учебного процесса, как игры, викторины и т. п.

Самостоятельная работа студентов ориентирована на изучение отдельных вспомогательных тем дисциплины, решение дополнительных рекомендованных задач и подбор практических примеров, иллюстрирующих теоретические основы алгоритмизации и программирования. Результаты самостоятельной работы выявляются как при ответах на теоретические вопросы, так и при разработке алгоритмов и программ для решения задач.

Текущий контроль осуществляется при выполнении и сдаче лабораторных работ.Форма контроля – экзамен.

2 ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ

2.1 Учебные издания

Расолька, Г.А.Паскаль :тэорыя і практыка праграміравання вучэб.-метад. дапам. / Г.А. Расолька, Ю.А. Крэмень. – Мінск : БДУ, 2008. –392 с.

2.2 Конспект лекций

Лекция 1

Алгоритм и его свойства

Основные вопросы

1. Понятие алгоритма.
2. Эмпирические свойства алгоритма.
3. Способы представления и разработки алгоритмов.
4. Типы структур в алгоритмах.
5. Основная теорема структурного программирования.

*Цель.*Ознакомление с понятием алгоритма и его свойствами. Изучение способов представления алгоритмов, алгоритмических структур и основной теоремы структурного программирования.

Понятие алгоритма

Важным этапом при решении задачи на компьютере является этап разработки алгоритма. В нашем курсе мы будем знакомиться только с понятием алгоритма, потому чтосуществующее точное математическое определение алгоритма требует глубоких математических знаний в области дискретной математики. Начнем с формулировки понятия алгоритма, которую дал академик А.П. Ершов в учебнике «Основы информатики и вычислительной технике: «Под алгоритмом понимают понятное и точное предписание (указание) исполнителю осуществить последовательность действий, направленных на достижение указанной цели или на решение предлагаемой задачи.

Приведем еще две формулировки понятия алгоритма, которые тоже будут рабочими в нашем курсе:

1. Алгоритм – это точное предписание, определяющее вычислительный процесс, ведущий от варьируемых начальных данных к искомому результату.

2. Под алгоритмом будем понимать конечную совокупность указаний, руководствуясь которыми можно выполнить любое сложное задание или решить задачу.

Свойства алгоритма

Алгоритм должен обладать следующими основными свойствами:

1. Понятность– исполнитель алгоритма должен понимать, как его выполнять. Иными словами, имея алгоритм и произвольный вариант исходных данных, исполнитель должен знать, как надо действовать для выполнения этого алгоритма.

2. Детерминированность (определенность) –каждое правило алгоритма должно быть четким, однозначным и не оставлять места для произвола. Благодаря этому свойству выполнение алгоритма носит механический характер и не требует никаких дополнительных указаний или сведений о решаемой задаче. При заданных исходных данных обеспечивается однозначность искомого результата;

3. Дискретность (прерывность, раздельность) – алгоритм должен представлять процесс решения задачи как последовательное выполнение простых (или ранее определенных) шагов (этапов, действий).

4. Результативность (или конечность) состоит в том, что за конечное число шагов алгоритм либо должен приводить к решению задачи, либо после конечного числа шагов останавливаться из-за невозможности получить решение с выдачей соответствующего сообщения, либо продолжаться в течение времени, отведенного для исполнения алгоритма, с выдачей промежуточных результатов.

5. Массовость означает, что алгоритм решения задачи разрабатывается в общем виде, т.е. он должен быть применим для некоторого класса задач, различающихся лишь исходными данными. При этом исходные данные могут выбираться из некоторой области, которая называется областью применимости алгоритма.

Формы представления алгоритмов

Одним из самых трудоемких этапов решения задачи на компьютере является разработка алгоритма. Человечество разработало эффективный алгоритм завязывания шнурков на ботинках. Многие дети с пятилетнего возраста могут это делать. Но дать чисто словесное описание этого алгоритма без картинок и демонстрации – очень трудно.

При разработке алгоритмов чаще всего используют следующие способы их описания: словесный, графический и с помощью алгоритмического языка (псевдокода).

На практике наиболее распространены следующие формы представления алгоритмов:

- словесная (запись на естественном языке);
- графическая (изображения из графических символов);

– псевдокоды (полуформализованные описания алгоритмов на условном алгоритмическом языке, включающие в себя как элементы языка программирования, так и фразы естественного языка, общепринятые математические обозначения и др. Русскоязычным представителем псевдокодов является алгоритмический язык);

– программная (тексты на языках программирования).

Словесный способ записи алгоритмов

Словесный способ записи алгоритмов представляет собой описание последовательных этапов обработки данных. Алгоритм задается в произвольном изложении на естественном языке.

Например. Записать алгоритм нахождения наибольшего общего делителя двух натуральных чисел (алгоритм Эвклида).

Алгоритм может быть следующим:

1. Задать два числа;
2. Если числа равны, то взять любое из них в качестве ответа и остановиться, в противном случае продолжить выполнение алгоритма;
3. Определить большее из чисел;
4. Заменить большее из чисел разностью большего и меньшего из чисел;
5. Повторить алгоритм с шага 2.

Описанный алгоритм применим к любым натуральным числам и должен приводить к решению поставленной задачи. Убедитесь в этом самостоятельно, определив с помощью этого алгоритма наибольший общий делитель чисел 102 и 24.

Словесныеописания алгоритмов не имеет широкого распространения, так как они обладают следующими существенными недостатками:

– строго не формализуемы;
– страдают многословностью записей;
– допускают неоднозначность толкования отдельных предписаний.

Графический способ записи алгоритмов

Графический способ представления алгоритмов является более компактным и наглядным по сравнению со словесным.

При графическом представлении алгоритм изображается в виде последовательностисвязанных между собой функциональных блоков, каждый из которых соответствуетвыполнению одного или нескольких действий.

Такое графическое представление называется схемой алгоритма илиблок-схемой. В блок-схеме каждому типу действий (вводу исходных данных, вычислению значений выражений, проверке условий, управлению повторением действий, окончанию обработки и т.п.) соответствует геометрическая фигура, представленная в видеблочного символа. Блочные символы

соединяютсялиниями переходов, определяющими очередность выполнения действий.

Графический способ записи алгоритмов – наиболее наглядный и распространенный. Он основан на использовании геометрических фигур(блоков), каждая из которых отображает конкретный этап процесса обработки данных, соединяемых между собой прямыми линиями, называемыми линиями потока.

Псевдокоды

Псевдокод представляет собой систему обозначений и правил, предназначенную для единообразной записи алгоритмов.

Псевдокод занимает промежуточное место между естественным и формальным языками. С одной стороны, он близок к обычному естественному языку, поэтому алгоритмы могут на нем записываться и читаться как обычный текст. С другой строны, в псевдокоде используются некоторые формальные конструкции и математическая символика, что приближает запись алгоритма к общепринятой математической записи.

В псевдокоде не приняты строгие синтаксические правила для записи команд, присущие формальным языкам, что облегчает запись алгоритма на стадии его проектирования и дает возможность использовать более широкий набор команд, рассчитанный на абстрактного исполнителя.

Однако в псевдокоде обычно имеются некоторые конструкции, присущие формальным языкам, что облегчает переход от записи на псевдокоде к записи алгоритма на формальном языке. В частности, в псевдокоде, так же, как и в формальных языках, есть служебные слова, смысл которых определен раз и навсегда. Они выделяются в печатном тексте жирным шрифтом, а в рукописном тексте подчеркиваются.

Единого или формального определения псевдокода не существует, поэтому возможны различные псевдокоды, отличающиеся набором служебных слов и основных (базовых) конструкций.

Примером псевдокода является школьный алгоритмический язык в русской нотации, описанный в учебнике А.Г. Кушниренко и др. "Основы информатики и вычислительной техники", 1991. Этот язык в дальнейшем мы будем называть просто "алгоритмический язык".

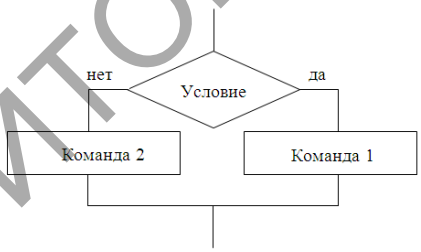
Основные типы структур в структурном программировании

В структурном программировании для записи алгоритмов используют три базовых структуры: следование, разветвление и цикл. Представим эти структуры в виде блок-схем.

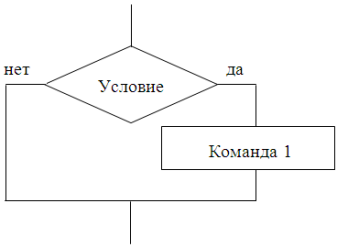
Характерной особенностью базовых структур является наличие в них одного входа и одного выхода. Стрелка сверху базовой структуры указывает на вход, а стрелка снизу – на выход.

1. Базовая структура "следование".Образуется последовательностью действий, следующих одно за другим.Она отображает естественный (сверху-вниз) порядок выполнения действий». Поэтому нет необходимости указывать в ней стрелки.

2. Базовая структура “разветвление”. В этой алгоритмической конструкции в зависимости от условия выполняется та или иная последовательность действий. Когда условие является истинным, выполняется команда 1, в противном случае выполняется команда 2.

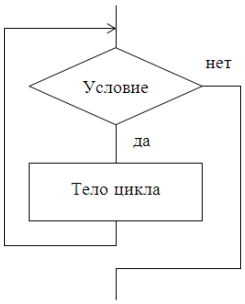


В сокращенной форме эта конструкция может быть представлена следующим образом.



Команда 1 выполняется тогда, когда условие является истинным.Если при выполнении или невыполнении условия требуется выполнить не одно, анесколько действий, то следует применить составной оператор.

3. Базовая структура «цикл».Она используется для представления последовательности действий, выполняемых многократно. Последовательность действий, повторяющаяся в процессе выполнения цикла, называется телом цикла.



При выполнении команды цикла сначала проверяется условие.Если условие истинно, то выполняются действия тела цикла и снова осуществляетсяпереход к проверке условия. Если условие окажется ложным, то осуществится выход из структуры цикла.

Основная теорема структурного программирования

Основная теорема структурного программирования утверждает, что логическая структура любого алгоритма может быть представлена комбинацией трех базовых структур: следования, разветвления и цикла.

Лекция 2

Алгоритмический язык

Основные вопросы

1. Общий вид алгоритма на алгоритмическом языке.
2. Запись списка параметров.
3. Описание типов данных.
4. Запись строк с аргументами и результатами.
5. Описание вспомогательных переменных.
6. Оператор присваивания.
7. Оператор разветвления.
8. Оператор цикла с предусловием.

Цель. Изучение способа записи алгоритма на алгоритмическом языке, описания параметров, вспомогательных переменных. Формирование умения использовать операторов присваивания, разветвления, выбора и цикла при записи алгоритма.

Алгоритмический язык– формальный язык, используемый для записи, реализации или изученияалгоритмов.В отличие от большинства языков программирования он не привязан к архитектурекомпьютера, не содержит деталей, связанных с устройством машины. Алголоподобный алгоритмический язык с русским синтаксисом был введён в употребление академикомА.П. Ершовымв середине 1980-х годов в качестве основы для «безмашинного» курса информатики. Впервые он был опубликован в учебнике «Основы информатики и вычислительной техники» в 1985 г.

Алгоритм на русском алгоритмическом языке в общем виде записывается следующим образом:

```
алг название алгоритма (список параметров с описанием типа)
аргсписок аргументов
резсписок результатов
начсписок вспомогательных переменных с описанием типа
последовательность команд
кон
```

Часть алгоритма от словаалгдо слованачназываетсязаголовком, а часть, заключенная между словаминачикон, – телом алгоритма.

В предложенииалгпосле названия алгоритма в круглых скобках перечисляются параметры алгоритма, являющиеся аргументами и результатами, и тип значения (цел,вещ,сим,литилилог) всех входных (аргументы) и выходных (результаты) переменных. При описании массивов (таблиц) используется служебное словотаб, дополненное граничными парами по каждому индексу элементов массива.

В записи алгоритма ключевые слова обычно подчёркиваются либо выделяются полужирным шрифтом. Для выделения логических блоков применяются отступы, а парные слова начала и конца находятся на одной вертикали.

Рассмотрим на примере вычисления функции n! (n!=1*2*3* … *n), как выглядит алгоритм на алгоритмическом языке. В этом примере предполагается, что n является натуральным числом.

```
алгФакториал (натn, вещf)
аргn
резf
начцелi
f:=1, i:=2
```

```
нцпокаi≤ n
f:=f*I, i:=i+1
кц
кон
```

В таблице приведены основные служебные слова алгоритмического языка.

Описание алгоритма	Типы данных	Обозначение условий	Обозначение цикла	Логические функции
алг– алгоритм	цел – целый	если	нц – начало цикла	и
арг –аргумент	вещ – вещественный	то	кц– конец цикла	или
рез – результат	сим – символьный	иначе	пока	не
нач – началоалгоритма	лит – строка	все	для	
кон– конец алгоритма	лог – логический	выбор	от	
	таб – массив	при	до	
	длин – длина		шаг	

Основные команды алгоритмического языка

Команда присваивания служит для вычисления выражений и присваивания их значений переменным. Общий вид команды присваивания:

A := B,

где знак ":=" означает команду, которая будетменять прежнее значение переменной, стоящей в левой части,на вычисленное значение выражения, стоящего в правой части.

Пример 1. Разработать алгоритм, который по заданному радиусу rбудет находить длину окружности C.

На алгоритмическом языке алгоритм будет выглядеть следующим образом

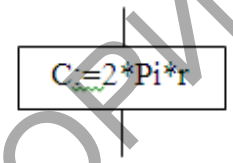
```
алгДлина окружности(вещr, C)
   аргr
   резC
нач
```

```
   C:=2
   C:=C*Pi
   C:=C*r
кон
```

В этом примере мы специально подробно расписали все действия для вычисления длины окружности. Далее мы это делать не будем. Мы должны понимать, что алгоритм – это не программа, а разработка алгоритма – это лишь один из этапов решения задачи на компьютере (хотя и очень важный). Поэтому алгоритм решения этой задачи можно было сразу записать так

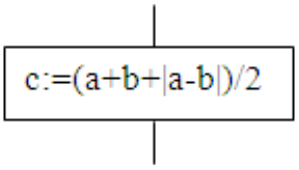
```
алг Длина окружности(вещr, C)
   аргr
   резC
нач
   C:=2*Pi*r
кон
```

Блок-схема данного алгоритма выглядит следующим образом



Пример 2. Разработать алгоритм, который найдет большее значение переменных *a* и *b* и присвоит это значение переменной *c*.

Внимательно рассмотрите приведенный ниже алгоритм и выясните, будет ли он решать поставленную задачу. Использованная в нем запись |*a*-*b*| означает модуль числа *a*-*b*.

Алгоритмический язык	Блок-схема
алгМаксимум(веща,b,c) аргa,b резc нач c:=(a+b+|a-b|)/2 кон	

Оператор разветвления

Оператор разветвления если в алгоритмическом языке имеет две формы: полную и сокращенную.

Полная форма	Краткая форма
если условие то операторы 1 иначе операторы 2 все	если условие то операторы все

При выполнении оператора разветвленияеслив полной форме сначала проверяется условие**.** Если оно истинно, то выполняются операторы 1 и завершается выполнение оператора если. В противном случае выполняются операторы 2и на этом прекращается выполнениеоператора если.

В случае сокращенной формы оператора еслиоператоры, после служебного слова то, выполняются только тогда, еслиусловие истинно.

Пример. Разработать алгоритм, который найдет большее значение трех переменных *a*,*b* и *c*и присвоит это значение переменной *d*.

Для решения этой задачи рассмотрим два способа. Один способ содержит вложенные друг в друга операторыесли; другой – использует ранее полученные знания для нахождения максимума из двух переменных.

1 способ	2 способ
алгМаксимум 1(веща,b,c,d) арга,b,c резd нач еслиа>b тоеслиа>c tod:=a иначеd:=c все иначееслиb>c tod:=b иначеd:=c все все кон	алгМаксимум 2(веща,b,c,d) арга,b,c резd нач еслиа>b tod:=a иначеd:=b все еслиc>d tod:=c все кон

Выясните преимущества и недостатки каждого способа.

Оператор цикла с предусловием

Оператор цикла с предусловиемиспользуется, когда число повторений операторов тела цикла заранее неизвестно. В алгоритмическом языке он имеет следующий вид:

```
нцпокаусловие
  операторы
кц
```

При выполнении оператора цикла с предусловиемвначале проверяется условие. Если условие является истинным, то выполняются операторы тела цикла и передача управления происходит на проверку условия. Так будет происходить до тех пор, пока условие не станет ложным. Когда условие станет ложным, завершится выполнение оператора цикла.

Пример. Разработать алгоритм, который будет находить n-е число Фибоначчи. Числа Фибоначчи определяются следующим образом:

$F_1=0$, $F_2=1$, $F_n=F_{n-1}+F_{n-2}$для $n>2$, где n – натуральное число.

```
алг Число Фибоначчи(целn,F)
   аргn
   резF
начцелF1,F2,i
   еслиn=1
      тоF:=0
      иначееслиn=2
               тоF:=1
               иначеF1:=0; F2:=1; i:=2
                  нцпокаi<n
                     F:=F1+F2; F1:=F2; F2:=F; i:=i+1
                  кц
            все
   все
кон
```

В этом алгоритме для организации цикла использован оператор цикла с предусловием пока. Если аргумент n равен 3, то условие цикла $i<n$обеспечивает однократное выполнение операторов тела циклаF:=F1+F2; F1:=F2; F2:=F; $i:=i+1$. При их выполнении переменнаяiстанетравной3, что сделаетусловие цикла ложным. Поэтому завершится выполнение цикла и самого алгоритма.На выходе из цикла мы получим значение переменнойF, равное 2, что на самом деле соответствует истине: третье число Фибоначчи равно 2. Самостоятельно исполните алгоритм приn, равном 5.

Лекция 3

Типы алгоритмов

Основные вопросы

1. Линейные, разветвляющиеся и циклические алгоритмы.
2. Исполнение алгоритма.

3. Правильность алгоритмов.

4. Подбор тестов для проверки алгоритма.

5. Тестирования и отладка алгоритмов.

6. Разработка смешанных алгоритмов с использованием разветвлений и циклов.

Цель. Изучение основных типов алгоритмов, формирование умений исполнять алгоритмы, выполнять тестирование и отладку алгоритмов, разрабатывать алгоритмы с использованием разветвлений и циклов.

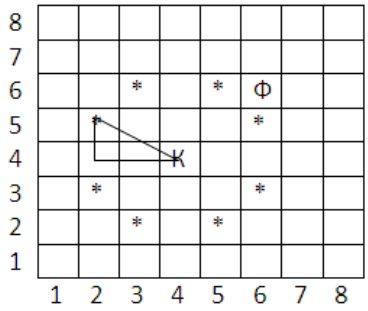
В соответствии с тремя базовыми структурами раличают три типа алгоритмов: линейные, разветвляющиеся и циклические. При выполнении линейного алгоритма исполнитель выполняет одну команду за другой в порядке их следования. В разветвляющихся алгоритмах действия исполнителя определяются результатами проверки некоторых условий. Структура раветвления может быть в полной и неполной форме. При исполнении циклического алгоритма отдельные команды или группы команд повторяются многократно.

При решении задач на компьютере, которые требуют только последовательные действия, отсутствует необходимость использовать этап алгоритмизации. Для их решения сразу записывают команды на языках программирования. Поэтому мы сразу перейдем к рассмотрению алгоритмов с разветвлениями, циклами и их комбинациями.

Алгоритмы с разветвлениями

Пример. На шахматной доске дан конь с координатами (x_k;$y_{k)}$и фигура с координатами ($x_ф$;$y_{ф)}$. Требуется разработать алгоритм, который определяет, находится ли фигура под боем коня. Если фигура находится под боем коня, то переменной *c* присвоить значение «бьет», в противном случае присвоить значение «не бьет».

Представим шахматную доску как координатную плоскость. Вместо буквенной нижней нумерации будем использовать цифровую. Обозначим коня буквой k, а фигуру – буквой Ф. Отметим на координатной плоскости звездочкой * те клетки, которые находятся под боем коня.



Если внимательно посмотреть на полученный рисунок, то можно заметить, что все клетки со звездочками находятся на одинаковом расстоянии от коня. Доказать это можно следующим образом. Соединим клетки с координатами (2;5) и (4;4), (2;4) и (4;4) и, наконец, (2;5) и (2,4). Мы получили прямоугольный треугольник, у которого длина одного катета равна 1, а другого равна 2. Построим таких же еще 7 треугольников, у которых гипотенузасоединяет клетку со звездочкой и клетку с конем. Полученные треугольники будут равны, так как у них равны две стороны и угол. Поэтому гипотенузы у них также будут равны. Геометрическое место точек, равноудаленных от одной точки, называемой центром, является окружностью. В нашем случае центром окружности является клетка с конем, а клетки со звездочками – точками окружности. Вычислим радиус окружности.Он определяется следующим образом:

$$r = \sqrt{1^2 + 2^2} = \sqrt{5}$$

Таким образом, чтобы фигура былапод боем коня, она должна стоять на клетке, находящейся на расстоянии $\sqrt{5}$ от клетки коня.

```
алгБой конем(целXk,Yk,Xf,Yf, лит с)
   аргXk,Yk,Xf,Yf
   резс
нач
   если(Xk-Xf)*(Xk-Xf)+(Yk-Yf)*(Yk-Yf)=5
      тос:='бьет'
      иначес:='не бьет'
   все
кон
```

Алгоритмы с циклами и разветвлениями

Пример 1. Требуется разработать алгоритм для вычисления суммы следующего ряда:

$$S = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \ldots + \frac{x^n}{n!}.$$

Для вычисления каждого члена ряда нет необходимости вычислять отдельно числитель и знаменатель, а потом делить числитель на знаменатель по следующим причинам. Во-первых, значения числителя и заменателя очень быстро возратают и при делении больших чисел теряется точность вычислений.

Во-вторых, при таком подходе время вычисления будет неоправданно возрастать, потому что придется использовать две структуры цикла, вложенные во внешний цикл.Мы же организуем такой вычислительный процесс, когда каждый последующий член ряда будет получаться из предыдущего по следующей формуле:

$$a_{i+1}=a_i \cdot x / i, i=1, 2, \ldots, n.$$

Такой вычислительный процесс называется итерационным, а циклы – итерациоными.

Теперь наш алгоритм на алгоритмическом языке можно записать следующим образом:

```
алгСумма ряда (целn,вещx,S)
   аргx,n
   резS
начцелi,вещa
   s:=1,a:=x,i:=1
   нцпокаi<=n
      s:=s+a,i:=i+1,a:=a*x/i
   кц
кон
```

Пример 2. Дана строка *a*. Требуется разработать алгоритм, который запишет буквы строки *a* в обратном порядке.

Алгоритм решения задачи можно представить так:

```
алгОбратный порядок(литa)
   аргa
   резa
начцелn, i, m,симb
   n:=длин(a)/2,i:=1
```

```
    нцпокаi<=n
       m:=длин(a),b:=a[i:i],a[i:i]:=a[m:m]
       a[m:m]:=b,i:=i+1
    кц
кон
```

В приведенном алгоритме показан способ обработки строковых данных. Служебное слово лит использовано для описания строки *a*. Функция длин(a) задает длину строки *a*. Для символьной переменной *b*использован описатель сим. Подстрока на алгоритмическом языке записывается следующим образом:

a[n:m],

где *a* –имя строки, *n*– номер первого символа подстроки, *m*– номер последнего символа подстроки. Если мы обращаемся к *i*-ому символу строки *a*, то пишема[i:i].

Исполнение алгоритма

Для того чтобы убедиться в правдоподобности приведенного алгоритмаОбратный порядок, исполним его. Результат исполнения будем заносить в таблицу. Строка заголовка таблицы будет содержать имена параметров, имена используемых переменных и прверяемые в алгоритме условия.В качестве теста для проверки алгоритма возьмем строку a='сорт'. Нам надо выяснить, получится ли после исполнения алгоритма ожидаемый результат: 'трос'.

a	*i*	*n*	*m*	*b*	*i<=n*
сорт	1	2			'да'
			4	с	
торт					
торс	2				'да'
			3	о	
ттрс					
трос	3				'нет'
Завершение работы цикла					

После того как переменная *i* стала равной 3, условие цикла *i<=n*оказалось ложным и алгоритм Обратный порядок завершил свою работу. Мы, действительно, видим, что строка *a*приняла ожидаемое значение –'трос'.

Процедура исполнения алгоритма никоим образом не является доказательством правильности работы алгоритма. Она лишь говорит в данном случае о том, что для выбранных исходных данных алгоритм выполнился с

требуемым результатом. Но и это вселяет в нас некоторую уверенность в том, что алгоритм работает верно.

Пример 3. Дан массив *a* размерности *n*, содержащий целые числа.Требуется разработать алгоритм, который переставит элементы массива *a* так, что вначале будут размещаться неположительные числа, а затем – положительные.

Алгоритм решения данной задачи на алгоритмическом языке можно представить так:

```
алгПерестановка(целтаба[1:n], целn)
   аргa,n
   резa
начцелf,l,x
   f:=1, l:=n
   нцпокаf<l
      еслиa[f]<=0
      тоf:=f+1
      иначеx:=a[f],a[f]:=a[l],a[l]:=x, l:=l-1
      все
   кц
кон
```

В этом алгоритме для описания массива *a* размерности *n*, содержащего целые числа, мы использовали служебные слова цел и таб, размерность массива указали в квадратных скобках a[1:n]. Исполните данный алгоритм самостоятельно для следующего массива: a={0;1;2;-2;0;-6;-4}.

Правильность алгоритмов

После построения алгоритма решения задачи, точнее, класса однотипных задач, всегда возникают вопросы: правильно ли построен алгоритм, верно ли решает он задачу. Для их решения следует проверить правильность алгоритма. Естественная и наиболее простая проверка – тестирование, которое и применяется на практике. Суть тестирования заключается в том, что, имея некоторую совокупность правильно решенных задач, решают их затем с помощью алгоритма и убеждаются в его истинности. Если имеются расхождения, то алгоритм соответственно исправляют.

Лекция 4

Характеристики времени и памяти

Основные вопросы

1. Размерность задачи.
2. Характеристики времени и памяти алгоритмов.
3. Единицы измерения времени и памяти алгоритма.
4. Оценка алгоритма в лучшем, среднем и худшем случаях.

*Цель.*Изучение характеристик времени и памяти алгоритмов. Формирование умений оценивать алгоритм в лучшем, среднем и худшем случаях.

Вычислительная сложность– понятие винформатикеитеории алгоритмов, обозначающее функцию зависимости объёма работы, которая выполняется некоторым алгоритмом, от размера входных данных. Раздел, изучающий вычислительную сложность, называетсятеорией сложности вычислений. Объём работы обычно измеряетсяпонятиями времени и пространства, называемыми вычислительными ресурсами. Время определяется количеством элементарных шагов, необходимых для решения задачи, тогда как пространство определяется объёмом памяти или места наносителе данных. Таким образом, в этой области предпринимается попытка ответить на центральный вопрос разработки алгоритмов: «как изменится время исполнения и объём занятой памяти в зависимости от размера входа?». Здесь под размером входа понимается длина описания данных задачи вбитах(например, взадаче поиска элемента в массиве, содержащего n элементов, длина входа почти пропорциональна количеству элементов). Поэтому в качестве размера этой задачи часто берут величину n).

Стеоретической информатикойтесно связаны такие области как алгоритмический анализитеория вычислимости. Связующим звеном между теоретической информатикой и алгоритмическим анализом является тот факт, что их формирование посвящено анализу необходимого количества ресурсов определённых алгоритмов решения задач, тогда как более общим вопросом является возможность использования алгоритмов для подобных задач. Конкретизируясь, попытаемся классифицировать проблемы, которые могут или не могут быть решены при помощи ограниченных ресурсов. Отличие теории вычислительной сложности от вычислительной теории заключается в сильном ограничении доступных ресурсов. Вычислительная теория отвечает на вопрос, какие задачи, в принципе, могут быть решены алгоритмически.

Теория сложности вычислений возникла из потребности сравнивать быстродействие алгоритмов, чётко описывать их поведение (время исполнения и объём необходимой памяти) в зависимости от размера входа.

Количество элементарных операций, затраченных алгоритмом для решения конкретного экземпляра задачи, зависит не только от размера входных данных, но и от самих данных. Например, количество операций алгоритма сортировки методом пузырькаказначительно меньше в случае, если входные данные уже отсортированы. Чтобы избежать подобных трудностей, рассматривают понятие временной сложности алгоритмав худшем случае.

Временная сложность алгоритма(в худшем случае) – это функция от размера входных данных, равная максимальному количеству элементарных операций, проделываемых алгоритмом для решения экземпляра задачи указанного размера.

Аналогично понятиювременной сложности в худшем случаеопределяется понятиевременная сложность алгоритма в наилучшем случае. Также рассматривают понятиесреднее время работы алгоритма, то есть математическое ожиданиевремени работы алгоритма. Иногда говорят просто: «Временная сложность алгоритма» или «Время работы алгоритма», имея в виду временную сложность алгоритма в худшем, наилучшем или среднем случае (в зависимости от контекста).

По аналогии с временной сложностью, определяютпространственную сложность алгоритма, только здесь говорят не о количестве элементарных операций, а об объёме используемой памяти.

Для того чтобы приступить к вычислению характеристик времени и памяти алгоритма, надо выбрать виртуальную вычислительнуюмашину или устройство, которое будет понятно исполнителю. В нашем случае проще всего в качестве такого устройства выбрать алгоритмический язык.

За единицу времени мы будем принимать время, требуемое для выполнения одного оператора присваивания и время выполнения проверки условия. За единицу памяти возьмем память, занимаемую одной переменной или одним элементом массива. Для некоторых задач можно взять память, занимаемую одним символом. Несмотря на кажущуюся грубость выбранных нами оценок, они позволяют сравнивать между собой алгоритмы решения задачи и выбирать лучший.

Оценка алгоритма

Для оценки сформулируем условие задачи, допускающей достаточную вариативность решения. Удобно для этой цели взять задачу последовательного поиска.

Условие. Дан массив целых чиселa размерности n. Требуется определить, есть ли в нем элемент a_i, равный некоторому целому числу b. Если такой элемент есть, то строковой переменной с присвоить значение “да”, в противном случае присвоить значение “нет”.

Обычно алгоритмы, которые представляют студенты, на алгоритмическом языке выглядят следующим образом. С правой стороны мы указываем, сколько времени в худшем случае затрачивают операторы в указанных строках.

Алгоритм Поиск1

```
алг Поиск1(целтаб a[1:n], целn,b, литc)
  арг a, n, b
  рез c
начцелi
  i:=1, c:= "нет"            2
  нцпокаi≤ n          n+1
     еслиa[i]=b       n
        тоc:= "да", i:=n+1    0
        иначеi:=i+1   n
     все
  кц
кон
```

Идея алгоритма Поиск1 заключается в том, что последовательно просматриваются элементы массива *a* и сравниваются с величиной *b*. Как только будет обнаружен элемент, равный *b*, цикл завершает свою работу, поскольку переменной *i*, используемой для организации цикла, присваивается значение *i*:=*n*+1, которое делает условие цикла ложным.

Алгоритм Поиск2

```
алг Поиск2(целтаб a[1:n], целb, n, литc)
    арг a, n, b
    рез c
начцелi
    i:=1, c:= "нет"            2
    нц покаi≤ nис:= "нет"      2(n+1)
        еслиa[i]=b             n
        тоc:= "да"             0
        иначеi:=i+1            n
      все
    кц
кон
```

В алгоритме Поиск2 условие цикла является составным. В этом условии проверяется не только переменная *i*, которая используется для последовательного просмотра элементов массива *a*, но и переменная *c*, которой предварительно установлено значение “нет”. Как только будет найден искомый элемент, этой переменной будет присвоено значение “да” и цикл завершит свою работу.

Алгоритм Поиск3

```
алг Поиск3(целтаб a[1:n], натn, целb, литc)
      арг a, n, b
      рез c
начцелi
      i:=1, c:= “нет”              2
      покаi≤ n  n+1
      нц
         еслиa[i]=b                n
            тос:= “да”             0
         все
         i:=i+1  n
      кц
кон
```

Алгоритм Поиск3 не завершает свою работу, как только найдет искомый элемент. Он будет осуществлять просмотр всех элементов массива *a* и каждый раз, когда будет встречаться элемент a_i, равный *b*, переменной *c*будет присваиваться значение “да”. Интуитивно становится понятным, что в среднем случае этот алгоритм будет работать дольше, чем алгоритмы Поиск1 и Поиск4.

Алгоритм Поиск4

```
      алг Поиск4(целтаб a[1:n], натn, целb, литc)
         арг a, n, b
         рез c
      начцелi
         i:=1                      1
         покаi≤ n                  n+1
         нц
            еслиa[i]=b             n
```

```
        тоi:=n+1            0
      все
      i:=i+1                n
   кц
   еслиi=n+2                1
      тос:= “да”            0
      иначе c:= “нет”       1
   все
кон
```

Цикл в алгоритме `Поиск4` завершает свою работу сразу же после обнаружения искомого элемента массива *a*. Для этой цели также, как и в алгоритме Поиск1, переменной *i*, используемой в условии цикла, присваивается значение, которое делает условие цикла ложным. Отличие от алгоритма Поиск1 состоит в том, что значение переменной *c*, в которой формируется результат выполнения алгоритма, присваивается после завершения работы цикла.

Характеристики алгоритмов

Вычислим характеристики времени $T(n)$ и памяти $M(n)$ для худшего случая, т. е. такого случая, когда алгоритм будет требовать при выполнении наибольшего количества времени. Такой случай соответствует ситуации, в которой массив *a* не содержит элемента a_i, равного числу *b*.

С правой стороны строк приведенных алгоритмов Поиск1, Поиск2, Поиск3, Поиск4 мы указали, сколько раз выполняется либо проверка условия цикла и разветвления, либо оператор присваивания. Команды это должны сделать сами. Просуммировав эти значения, мы получаем соответственно следующие характеристики времени в худшем случае для приведенных алгоритмов: $T_1(n)=3n+3$, $T_2(n)=4n+4$, $T_3(n)=3n+3$, $T_4(n)=3n+4$. Для всех четырех алгоритмов память, требуемая для выполнения алгоритма в худшем случае, будет одинаковой: $M(n)=n+4$. В эту величину входит память, требуемая для размещения исходных данных, результата и вспомогательной переменной *i*.

Из вычисленных характеристик времени выполнения алгоритма в худшем случае видно, что при достаточно большом размере задачи *n* алгоритм `Поиск2` будет выполняться дольше остальных в 4/3 раза:

$$\lim_{n \to \infty} \frac{4n+4}{3n+4} = \frac{4}{3}.$$

Как правило, командам не удается найти более быстрый алгоритм для решения задачи последовательного поиска. Поэтому преподаватель сам знакомит студентов с идеей алгоритма для выполнения быстрого

последовательного поиска (БПП) и сам записывает на доске алгоритм БПП, который приведен ниже.

Алгоритм БПП

```
алг БПП(целтаб a[1:n+1], натn, целb, литc)
      арг a, n, b
      рез c
начцелi
      i:=1,    a[1:n+1]:= b          2
      пока a[i]≠ b                   n+1
      нц
        i:=i+1   n
      кц
      еслиi≤n   1
        тос:= “да”                   0
        иначе c:= “нет”              1
      все
кон
```

Характеристики времени и памяти в худшем случае для этого алгоритма будут равны соответственно следующим величинам:

$$T_{\text{БПП}}(n) = 2n+5, \qquad M_{\text{БПП}}(n) = n+5.$$

При сравнении характеристики времени алгоритма БПП с характеристикой времени алгоритма Поиск1 (который является наиболее быстрымиз приведенных) при достаточно большом значении n получаем следующее:

$$\lim_{n \to \infty} \frac{3n+3}{2n+5} = \frac{3}{2}.$$

Это означает, что алгоритм БПП выполняется в 1,5 раза быстрее, чем алгоритм Поиск1, говоря другими словами, его использование вместо алгоритма Поиск1 позволяет экономить электроэнергию в 1,5 раза.

Лекция 5

Методы обработки информации

Основные вопросы

1. Двоичный поиск.

2. Прямой доступ.

3. Алгоритм проверки правильности расстановок скобок одного типа.

4. Обработка строковых данных. Поиск различных символов в строке и определение частоты их появления.

Цель. Изучение основных методов обработки информации.

Лекция 6

Технология работы в системе Pascal ABC

Основные вопросы

1. Набор текста программы.

2. Запуск программы на выполнение и ее остановка.

3. Пошаговое выполнение программы.

4. Трассировка модулей программы.

5. Работа в окне отладки программы.

Цель. Изучение специальной терминологии и правил набора текстовой информации. Знакомство с единицами измерения и корректурными знаками.

Лекция 7

Основы построения программ на языке Pascal ABC

Основные вопросы

1. Структура программы на языке Pascal ABC.

2. Алфавит языка.

3. Стандартные идентификаторы, идентификаторы пользователя.

4. Выражения и операции.

5. Константы и переменные.

6. Типы данных: скалярные и структурированные.

7. Арифметические выражения и операции.

8. Логические выражения и операции. Приоритет операций.

9. Разделы описания меток, констант, типов данных, переменных, процедур и функций.

10. Раздел операторов, комментарии.

Цель. Изучение структуры программы на языке Pascal ABC, способов записи идентификаторов, выражений, констант, переменных, типов данных и операций.

Лекция 8

Управляющие конструкции языка Pascal ABC

Основные вопросы

1. Структурные операторы: составные и условные, операторы цикла.
2. Общие правила пунктуации.
3. Массив как структурированный тип данных.
4. Массив, описание типа, одномерные и двумерные массивы. Размещение в памяти.
5. Действия над одномерными и двумерными массивами.
6. Методы поиска. Бинарный поиск.
7. Упорядочение элементов массива.
8. Разработка программ с использованием различных методов сортировок массивов.
9. Методы сортировки: сортировка простым выбором, сортировка простым обменом.

Цель. Изучение основных управляющих конструкций языка Pascal ABC, формирование умений разрботавать программы для решения типовых задач.

Лекция 9

Процедуры и функции

Основные вопросы

1. Порядок организации пользовательских процедур и функций.
2. Глобальные и локальные, фактические и формальные параметры.
3. Прямая и косвенная рекурсия. Рекурсивные подпрограммы.
4. Разработка программных модулей с использованием функций и процедур.

Цель. Изучение методов создания пользовательских процедур и функций, формирование умений разрабатывать программные модули с использованием функций и процедур.

Лекция 10

Работа со строковыми типами данных

Основные вопросы

1. Моделирование процессов обработки символьной информации: поиск, удаление, преобразование и замена.
2. Строковые и символьные типы данных.
3. Процедуры и функции обработки строк и символов.

Цель. Изучение процессов обработки символьной информации, формирование умений разрабатывать программы для обработки строк и символов.

Лекция 11

Построение изображения на экране

Основные вопросы

1. Графические возможности языка Pascal.
2. Построение простых графических рисунков.
3. Библиотечные модули CRT, GRAPH и их функции.

Цель. Изучение графических возможностей языка PascalABCдля построений изображений на экране.

ПРАКТИЧЕСКИЙ РАЗДЕЛ

3.1 Описание лабораторных работ

Лабораторная работа № 1

Цель работы: формирование знаний и умений по работе с интегрированной средой (ИП) языка программирования(ЯП) Паскаль. Приобретение навыков работы с меню ИС ЯП Паскаль.

Краткие теоретические сведения

Интегрированная среда языка программирования ТУРБО ПАСКАЛЬ

Разработка программ на Паскале включает в себя следующие действия (этапы разработки программы): ввод и редактирование текста программы на языке программирования Паскаль, ее трансляцию, отладку.

Для выполнения каждого этапа применяются специальные средства: для ввода и редактирования текста используется редактор текстов, для трансляции программы – компилятор, для построения исполняемого компьютером программного модуля с объединением разрозненных откомпилированных модулей и библиотекой стандартных процедур Паскаля - компоновщик (linker), для отладки программ с анализом ее поведения, поиском ошибок, просмотром и изменением содержимого ячеек памяти компьютера- отладчик (debugger).

Для повышения качества и скорости разработки программ в середине 80-х гг. была создана система программирования Турбо Паскаль. Слово Турбо в названии системы программирования — это отражение торговой марки фирмы-разработчика Borland International, Inc. (США).

Систему программирования Турбо Паскаль называют интегрированной (integration - объединение отдельных элементов в единое целое) средой программирования, так как она объединяет в себе возможности ранее разрозненных средств, используемых при разработке программ: редактора текстов, компилятора, компоновщика, отладчика, и при этом обеспечивает программисту великолепные сервисные возможности. Часто ее кратко называют IDE (Integrated Development Environment - интегрированная среда разработки).

Интегрированная среда программирования Турбо Паскаль версий 6.0 и 7.0 имеет следующие возможности:

• множество накладывающихся окон;

• поддержка мыши, меню, диалоговых окон;

• многофайловый редактор, который может редактировать файлы до 1 Мбайта;

• расширенные возможности отладки;

• полное сохранение и восстановление среды разработки.

К ее существенным отличиям от среды программирования Турбо Паскаль более ранних версий относятся:

• объектно-ориентированная среда разработки прикладных программ Turbo Vision;

• полные возможности встроенного ассемблера;

• личные поля и методы в объявлении объектов;

• директива расширенного синтаксиса $X, которая позволяет вам интерпретировать функции как процедуры (и игнорировать результаты функций);

• директивы ближних и дальних процедур;

• расширенные возможности встроенной справочной системы с использованием вырезки и вставки кода примеров для каждой библиотечной процедуры и функции.

Основные файлы пакета Турбо Паскаль

Допустим, что система программирования Турбо Паскаль установлена на диске D: в каталоге D:\BORLAND\BP, то в каталоге ..\BP находятся следующие основные файлы Турбо Паскаля:

TURBO.EXE — интегрированная среда программирования;

TURBO.HLP — файл, содержащий данные для оперативной подсказки;

TURBO.TP — файл конфигурации системы;

TURBO.TPL — библиотека стандартных модулей Турбо Паскаля.

В каталоге D:\BORLAND\BP\BGI находятся файлы, необходимые для работы в графическом режиме: GRAPH.TPU — модуль с графическими

процедурами и функциями Турбо Паскаля, несколько файлов с расширением .BGI — драйверы различных типов видеосистем компьютеров, несколько файлов с расширением .CHR, содержащих векторные шрифты.

Запуск интегрированной среды программирования Турбо Паскаль

Для запуска интегрированной среды программирования нужно установить текущим каталог с Турбо Паскалем и (или) ввести команду: turbo.exe. После запуска программы экран компьютера будет иметь вид, показанный на рисунке 3.

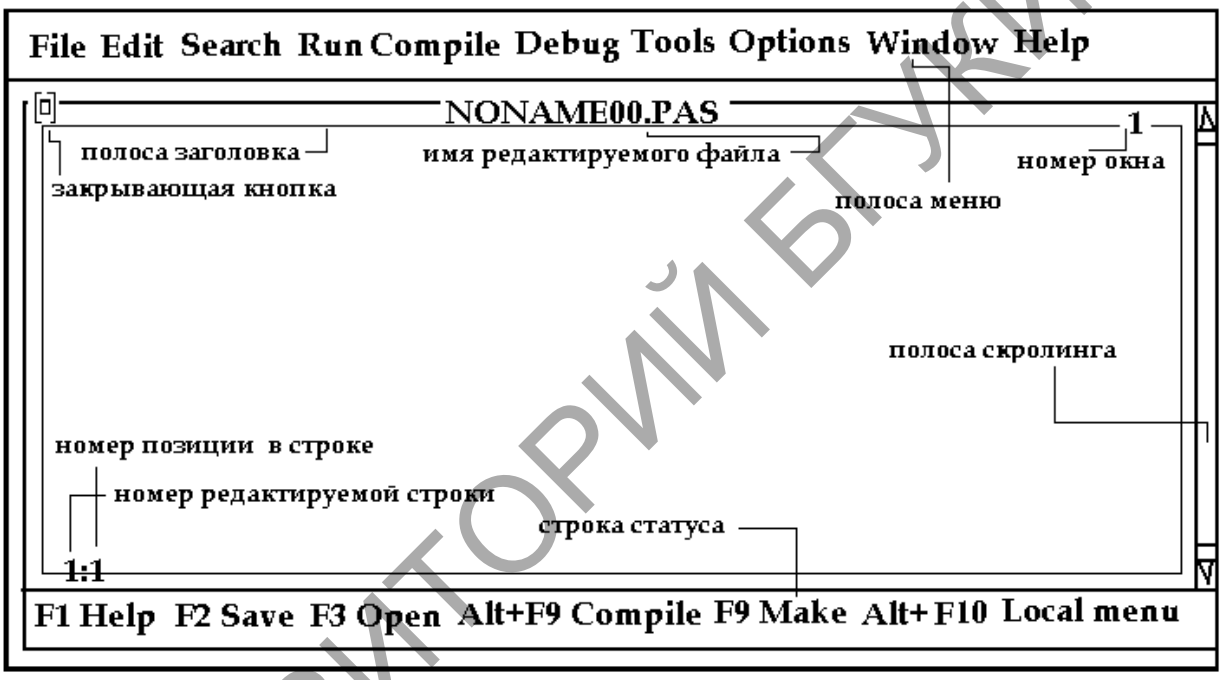


Рис. 3 Окно интегрированной среды Турбо Паскаль

На экране отображаются три видимых компонента интегрированной среды программирования: полоса меню в верхней части, область окна в центре и строка статуса внизу.

Полоса меню и подменю. Полоса меню является основным доступом ко всем командам меню. Она становится невидимой только в то время, когда вы просматриваете вывод своей программы. Если полоса меню активна, то заголовок меню будет высвечен; это текущее выбранное меню. Если за командой меню следует знак многоточия (...), выбор команды приведет к выводу диалогового окна. Если за командой следует стрелка (>), то команда ведет в другое меню. Команда без знака многоточия или без стрелки указывает, что как только вы ее выберете, произойдет какое-то действие.

Строка статуса. Строка статуса отображается в нижней строке экрана и выполняет следующие функции:

• напоминает основные строки ключей и сокращений (или горячих клавиш), допустимых в этот момент в активном окне;

• предоставляет самый быстрый вариант выполнения действий, отмечая горячие клавиши в строке статуса мышью вместо выбора команд из меню или нажатия последовательности клавишей;

• содержит информацию о том, какая функция выполняется. Например, когда сохраняется редактируемый файл, в строке статуса выводится сообщение: "Saving filename...";

• предлагает краткие советы по выбранной команде меню и элементам диалогового окна.

При смене окна или изменении характера деятельности информация в строке статуса сразу же меняется. Одна из наиболее характерных строк статуса - та, которую вы видите во время написания и редактирования программ в окне редактора.

Работа с меню ИС

После запуска среды программирования Турбо Паскаль 7.0 в верхней части экрана выводится меню:

File Edit Search Run Compile Debug Tools Options Window Help

Пункт меню File (файлы) выбирается нажатием F10 - File или Alt+F.

Меню File

Меню File позволяет открывать и создавать файлы с программами в окнах редактора, а также сохранять изменения, выполнять другие файловые функции, осуществлять временный выход в DOS и выходить совсем из среды программирования. После выбора этого пункта меню на экран выводится выпадающее меню.

Команда New (новый) открывает новое редакционное окно с именем по умолчанию NONAMEXX.PAS (вместо XX устанавливаются числа от 0 до 99) и автоматически делает его активным.

Команда Open (открыть) - F3 показывает диалоговое окно с выбором программных файлов для их открытия в окне редактора. Это диалоговое окно содержит окно ввода, список файлов, кнопки, помеченные как Open (Открыть), Replace (Заменить), Cancel (Отказ) и Help (Справочная информация), и информационную панель, описывающую выбранный файл.

Команда Save (сохранить)- F2 сохраняет файл, находящийся в активном окне редактора, на диск.

Команда Save As (сохранить как) позволяет сохранить файл, находящийся в активном окне редактора, под другим именем, в другом каталоге, на другом устройстве.

Команда Save all (сохранить все) работает точно так же, как команда Save, за исключением того, что она сохраняет содержание всех модифицированных файлов, а не только файл, находящийся в активном окне редактора. Эта команда недоступна, если ни одно окно редактора не открыто.

Команда Change dir (изменить справочник) позволяет с помощью диалогового окна Change Directory задать устройство и каталог как текущий. Текущий каталог-это каталог, который Турбо Паскаль использует для сохранения файлов и их поиска. (При использовании относительных путей в Options/Directories они относятся только к текущему каталогу).

Меню Search

Меню Search (поиск) выбирается нажатием Alt+S. Оно позволяет осуществлять поиск текста, объявления процедур и месторасположение ошибок в ваших файлах. После выбора этого пункта меню на экран выводится выпадающее меню.

Команда Find (найти) - Alt+S+F (или Ctrl+Q+F) показывает диалоговое окно Find, позволяющее набрать текст, который вы хотите найти, и установить опции, влияющие на поиск.

Диалоговое окно поиска содержит несколько кнопок:

• Options (опции);

• Case sensitive - различение прописных и строчных букв;

•Whole words only-только целые слова;

• Regular expression — регулярное выражение. Включайте эту кнопку в том случае, если хотите, чтобы Турбо Паскаль распознавал спецификаторы ^ , $,., *, +, [], \ в строке поиска.

Команда Replace (заменить)- Alt+S+R (или Ctrl+Q+A) выводит диалоговое окно, позволяющее набирать образец текста для поиска и образец текста, на который его надо заменить. Дополнительная кнопка Prompt on replace (подсказка для замены) управляет подсказкой для каждой замены.

Команда Search Again (поиск вновь) - Ctrl+L повторяет последнюю команду Find или Replace.

Команда Go to line number (идти к строке номер) выдает подсказку номера строки, которую вы хотите найти.

Команда Show last compiler error показывает последнюю ошибку компилятора в верхней части экрана и позиционирует курсор возле ошибки. Если последняя компиляция удачна, то сообщение не высвечивается.

Команда Find error (поиск ошибки)- Alt+F8 находит местоположение ошибки времени выполнения.

Команда Find procedure (поиск процедуры) выводит диалоговое окно, позволяющее ввести имя процедуры для поиска. Эта команда доступна только во время сеанса отладки.

Меню Run

Меню Run (выполнение) выбирается нажатием Alt+R. Команды меню запуска запускают вашу программу, а также начинают и заканчивают сеансы отладки. После выбора этого пункта меню на экран выводится выпадающее меню.

Команда Run (выполнение) - Ctrl+F9 запускает вашу программу, используя параметры, которые вы передали в нее с помощью команды Run/Parameters. Если со времени последней компиляции исходный код был модифицирован, товстроенный менеджер проекта автоматически перекомпилирует и отредактирует вашу программу.

Команда Program reset (сброс программы)- Ctrl+F2 прекращает текущий сеанс отладки, освобождает память, размещенную под вашу программу и закрывает все открытые файлы, используемые программой.

Команда Go to cursor (перейти на курсор) - F4 выполняет программу до строки, на которой стоит курсор в текущем окне редактора. Если курсор стоит на строке, которая не содержит выполнимое утверждение, команда выдаст предупреждение.

Команда Trace into (пошаговая трассировка) - F7 выполняет вашу программу утверждение за утверждением. Когда она достигает вызова процедуры, то выполняет каждое утверждение внутри процедуры, вместо выполнения процедуры как одного. Если утверждение не содержит вызова процедур, доступных отладчику, Trace into остановится на следующем выполнимом утверждении.

Команда Step over (шаг через) - F8 выполняет следующие утверждения в текущей процедуре. Она не выполняет трассировку внутрь вызовов процедур нижнего уровня, даже если они доступны отладчику.

Меню Compile

Меню Compile (компиляция) выбирается нажатием Alt+C. Используется для того, чтобы сделать compile, make или build программы в активном окне. После выбора этого пункта меню на экран выводится выпадающее меню.

Команда Compile (компиляция) - Alt+F9 компилирует активный файл редактора. При этом на экран выводится окно статуса, показывающее результаты компиляции. Когда компиляция завершается, нажмите любую клавишу, чтобы удалить это окно. Если происходит какая-нибудь ошибка или предупреждение, окно редактора, содержащее исходный код с ошибкой,

становится активным, появляется сообщение об ошибке, а курсор устанавливается на местоположении первой ошибки.

Команда Make (сборка)- F9 вызывает встроенный менеджер проекта для создания .EXE -файла.

Команда Build (полная сборка) перекомпилирует все файлы независимо от их даты. Эта команда подобна команде Compile/ Make за исключением того, что она не имеет условий.

Команда Destination (назначение) позволяет определить, будет ли выполняемый код храниться на диске (как файл .EXE) или он будет храниться в памяти (и, таким образом, теряться при выходе из Турбо Паскаль).

Команда Information открывает окно, в котором выдается информация о последней скомпилированной программе, текущем состоянии памяти и окружения.

Меню Debug

Меню Debug (отладка) выбирается нажатием Alt+D. Команды меню отладки управляют всеми свойствами интегрированного отладчика. После выбора этого пункта меню на экран выводится выпадающее меню.

Команда Breakpoints (точки прерывания) открывает диалоговое окно, позволяющее управлять использованием безусловных точек прерывания. Оно показывает все установленные точки прерывания, номера их строк и условия. Условие имеет архивный список, позволяющий выбрать условие точки прерывания, использованное ранее.

Команда Call stack - Ctrl+F3 открывает окно, в котором показана последовательность процедур, вызываемых исполняемой программой. В окне содержатся имена процедур и значения передаваемых им параметров.

Команда Register открывает окно, показывающее регистры CPU (центрального процессора), используемые обычно при отладке модулей на ассемблере. Верхняя половина окна показывает содержимое регистров, а нижняя- содержимое восьми флагов.

Команда Watch открывает окно, в котором содержатся выражения и их изменяющиеся значения. Элементы окна добавляются или убираются командой Add Watch.

Команда Add watch (добавить выражение для просмотра) - Ctrl+F7 вставляет выражение просмотра в окно Watch. При выборе этой команды отладчик открывает диалоговое окно и выдает подсказку для ввода выражения просмотра. Выражением по умолчанию является слово, на котором стоит курсор в текущем окне редактора. Имеется также архивный список, который можно применить для быстрого ввода выражения, использованного ранее. Если

окно Watch является активным, можно вставить новое выражение для просмотра посредством нажатия Ins.

Команда Add breakpoint открывает диалоговое окно, в котором задаются параметры новой точки прерывания. В поле Condition вводится условие, по выполнении которого происходит прерывание. В поле Pass Count устанавливается число проходов контрольной точки, после выполнения которых произойдет останов. В поле File Name записывается полное путевое имя исходного файла, содержащего текущую контрольную точку. В поле Line Number показывается номер строки, содержащей текущую точку прерывания. Можно ввести новое значение номера.

Меню Tools

Меню Tools (сервисные средства) выбирается нажатием Alt+T. Данное меню обеспечивает различные отладочные команды сообщения, следуемые за списком по умолчанию программ и любых программ, установленных пользователем с помощью команды Options/Tools/Transfer.

Команда Messages открывает окно, в котором отображается информация из программы, выдаваемая посредством фильтра DOS (типа GREP).

Команда Go to next - Alt+F8 позволяет перейти к следующему элементу списка. Список содержит имена программ, которые можно запускать, не выходя из Турбо Паскаля. Такие программы называются трансферными. По окончании такой программы выполняется возврат в среду программирования.

Команда Go to previous- Alt+F7 осуществляет переход к предыдущему элементу списка.

Меню Options

Меню Options (опции) выбирается нажатием Alt+0. Оно содержит команды, позволяющие посмотреть и изменить различные установки по умолчанию в Турбо Паскале. Большинство команд меню приводит к появлению диалогового окна. После выбора этого пункта меню на экран выводится выпадающее меню.

Команда Compiler... (компилятор) выводит меню, которое предоставляет несколько опций для установки, влияющих на компиляцию кода.

Команда Memory sizes (размеры памяти) позволяет определить потребности памяти по умолчанию для программы. Все три установки можно задать в своем исходном коде, используя директиву компилятора $M. В поле Stack Size задается размер (в байтах) сегмента стека. Размер по умолчанию 16,384, максимальный размер – 65,520.В окне Low Heap Limit задается минимальный требуемый размер кучи (в байтах). По умолчанию минимальный размер равен 0 Кбайт. В окне High Heap Limit задается максимальный требуемый размер кучи (в байтах). По умолчанию максимальный размер равен

655,360, который (в большинстве систем) распределит всю доступную память в кучу. Это значение должно быть больше или равно наименьшему размеру кучи.

Команда Linker (редактор связей) позволяет сделать несколько установок, влияющих на редактирование.

Команда Debugger (отладчик) открывает диалоговое окно, чтобы сделать несколько установок, влияющих на интегрированный отладчик.

Команда Directories (каталоги) определяет в Турбо Паскале, где искать файлы, необходимые для компиляции, редактирования связей и файлы вывода.

Назначение каждого окна ввода:

• EXE and TPU Directory задает каталог вывода для файлов .EXE или .TPU. Если ввода в этом окне не было, файлы будут храниться в каталоге, где находятся исходные файлы.

• Include Directories задает каталог, содержащий стандартные включаемые файлы.

• Unit Directories задает каталоги, содержащие ваши файлы модулей Турбо Паскаля.

• Object Directory используется для задания каталогов, содержащих файлы .OBJ (подпрограммы ассемблерного языка).

Команда Tools открывает диалоговое окно, с помощью которого можно добавлять новые или убирать программы из меню Tools. Поле Program titles содержит список для обзора, добавления или изъятия трансферных программ.

Команда Environment (среда) дает возможность сделать установки для среды. Эта команда открывает меню, позволяющее выбрать установки из опций Preferences, Editor и Mouse.

Меню Window

Меню Window(окна) выбирается нажатием Alt+W. Оно содержит команды управления окном. Большинство из окон, которые вы откроете из этого меню, имеют все стандартные элементы окна скроллинг, закрывающую кнопку и кнопки масштабирования, позволяющие посмотреть и изменить различные установки по умолчанию в Турбо Паскале. Первые девять окон пронумерованы. Для выбора окна по номеру задать Alt+N окна. После выбора этого пункта меню на экран выводится меню.

Расположение открытых окон определяют команды Tile (черепица - неперекрывающееся расположение окон) и Cascade (каскад- расположение окон одно за другим с просмотром только активного окна, а для других окон видны только имена файлов и номера окон).

Команда Close all закрывает все окна.

Команда Refresh display восстанавливает экран, если программа его случайно испортила.

Команда Size/Move - Ctrl+F5 позволяет задать размер и позицию окна на экране.

Команда Zoom - F5 раскрывает активное окно во весь экран. Если окно уже расширено, то команда восстанавливает его текущий размер.

Команда Next - F6 активизирует следующее окно.

Команда Previous- Shift+F6 активизирует предыдущее окно, т. е. окно, бывшее активным перед текущим.

Команда Close - Alt+F3 закрывает текущее окно.

Команда List - Alt+0 используется для получения списка всех открытых окон.

Меню Help

Меню Help (помощь) выбирается нажатием Alt+H. Оно дает доступ к встроенной справочной информации в специальном окне. Справочная информация имеется по всем аспектам интегрированной среды Турбо Паскаль. (Также в строке статуса появляются подсказки для меню в одну строку и диалоговых окон, когда бы ни была выбрана команда.) После выбора этого пункта меню на экран выводится выпадающее меню.

Вы можете нажать Ctrl+F 1 на любом слове для получения справочной информации. Если слово не найдено, выполняется поиск вперед по оглавлению и показывается ближайший соответствующий текст.

Команда Contents (содержание) открывает окно Help с основной таблицей содержания. Из этого окна можно перейти к любой другой части системы справочной информации.

Команда Index (оглавление)- Shift+Fl открывает диалоговое окно, показывающее полный список ключевых слов справочной информации (специально высвеченный текст на экране справочной информации, позволяющий быстро передвигаться к соответствующему экрану).

Команда Topic search (поиск раздела) - Ctrl+Fl показывает справочную информацию по языку и по текущему выбранному элементу.

Контрольные вопросы

Интегрированная среда Турбо Паскаль. Возможности.

Структура экрана. Основные элементы.

Меню ИС. Основные пункты.

Лабораторная работа 2

Цель работы: Приобретение навыков написания программ с использованием различных форматов вывода данных.

Для организации циклов в ТП можно использовать конструкцию FOR…TO (DOWNTO) … DO. Такая конструкция позволяет реализовать повторение в Паскаль-программах. Ее называют перечисляемым циклом или циклом со счетчиком. В этом операторе указываются следующие параметры:

– имя переменной, в которой хранится число повторений цикла (переменной цикла или счетчика цикла),

– некоторое начальное значение для переменной цикла (счетчика), которое она получает при первом выполнении цикла),

– некоторое конечное значение для переменной цикла, достигнув которое повторение цикла прекращается (условие завершения цикла).

Внимательно рассмотрите приведенный ниже пример программы, иллюстрирующий использование перечисляемого цикла, и выясните, что эта программа делает.

```
Program Summing;
VAR
      I, Sum, a: INTEGER;
BEGIN
      Sum:=0:
      FOR i:=1 TO 10 DO
         BEGIN
            Read(a);
            Sum:=Sum+a;
         END;
WriteLn('Sum= ',Sum);
ReadLn;
END.
```

Задача 1. Дан массив действительных чисел A размерности n×m ($1 \le n \le 10$, $1 \le m \le 10$). Составьте программу, которая будет находить максимальный элемент в нем, а также номер строки и номер столбца, в котором этот элемент расположен.

Ввод исходных данных организуйте следующим образом:

nm

$a_{11}\ a_{12}\ a_{13}\ \ldots\ a_{1m}$

$a_{21}\ a_{22}\ a_{23}\ \ldots\ a_{2m}$

$a_{31}\ a_{32}\ a_{33}\ \ldots\ a_{3m}$

. . . . . .

$a_{n1}\ a_{n2}\ a_{n3}\ \ldots\ a_{nm}$

Результат выведите следующим образом:

$a_{ij}ij$ (где a_{ij}– максимальный элемент, i и j – номер строки и столбца соответственно, где максимальный элемент расположен).

Задача 2. Разработайте программу, которая в массиве целых чисел A размерности n будет переставлять элементы так, чтобы вначале будут разместились отрицательные элементы, а затем – неотрицательные. Размерность n ($1 \le n \le 10$) массива A и значения его элементов введите с клавиатуры. На экран в первой строке выведите исходный массив, во второй строке – массив, полученный в результате перестановки элементов.

Работоспособность программы проверьте для следующих наборов исходных данных: 1) n=1, A=(5); 2) n=10, A=(0; -5; 3; 0; 10; -8; 6; 12; -4; 7).

Задача 3. Дан массив целых чисел A размерности n ($1 \le n \le 10$). Составьте программу, которая будет находить все различные числа в нем и подсчитывать, сколько раз каждое из этих чисел встречается в массиве. Ввод исходных данных организуйте следующим образом:

1-я строка: n;

2-я строка: $a_1\ a_2\ a_3\ \ldots\ a_n$.

Результат выведите следующим образом:

m (m – найденное количество различных чисел)

Различные числа

$b_1\ b_2\ b_3\ \ldots\ b_m$

Частота чисел

$k_1\ k_2\ k_3\ \ldots\ k_m$

При разработке программы используйте алгоритм быстрого последовательного поиска.

Внимательно рассмотрите приведенные ниже примеры программ **Chastot1** и **Chastot2**. Выясните, чем они различаются. Выполните их для нескольких одинаковых наборов исходных данных и сформулируйте задачу, которую они решают.

```
PROGRAM CHASTOT1;
VAR A: ARRAY[1..20] OF REAL;
   B: ARRAY[1..20] OF REAL;
   K: ARRAY[1..20] OF INTEGER;
   I,J,N,M: INTEGER;
BEGIN
   WriteLn('Vvedite N');
   ReadLn(N);
   WriteLn('Vvedite N chisel ');
   FOR I:=1 TO N DO
```

```
    Read(A[I]);
  M:=0;
  FOR I:=1 TO N DO
    BEGIN
      B[M+1]:=A[I]; J:=1;
      WHILE B[J]<>A[I] DO
        J:=J+1;
      IF J=M+1
        THEN
          BEGIN M:=M+1; K[M]:=1; END
        ELSE K[J]:=K[J]+1;
    END;
  WriteLn('Kolichestvo razlichnyx simvolov ', M:3);
  FOR I:=1 TO M DO
    WriteLn('B[',I:2,']=',B[I]:2:0,'    K[',I:2,']=', K[I]:2);
END.

PROGRAM CHASTOT2;
VAR A: ARRAY[1..20] OF REAL;
  K: ARRAY[1..20] OF INTEGER;
  I,J,N,M: INTEGER;
BEGIN
  WriteLn('Vvedite N');
  ReadLn(N);
  WriteLn('Vvedite N chisel ');
  FOR I:=1 TO N DO
    Read(A[I]);
  M:=0;
  FOR I:=1 TO N DO
    BEGIN
      A[M+1]:=A[I]; J:=1;
      WHILE A[J]<>A[I] DO
        J:=J+1;
      IF J=M+1
        THEN
          BEGIN M:=M+1; K[M]:=1; END
        ELSE K[J]:=K[J]+1;
    END;
  WriteLn('Kolichestvo razlichnyx simvolov ', M:3);
```

```
  FOR I:=1 TO M DO
    WriteLn('R[',I:2,']=',A[I]:2:0,'    K[',I:2,']=', K[I]:2);
END.
```

Лабораторная работа 3

Цель работы: Приобретение навыков написания циклических программ с использованием конструкцию REPEAT … UNTIL.

Кроме рассмотренной в предыдущей лабораторной работе конструкции FOR…TO (DOWNTO) … DO для организации циклов в ТП можно использовать. При использовании в программе этой циклической конструкции последовательность операторов (тело цикла) обрамляется словами REPEAT и UNTIL. В любом случае последовательность операторов, входящих в тело цикла, выполняется один раз, после чего проверяется условие завершения цикла, записанное следом за словом UNTIL. Если это условие выполняется, цикл завершается. В противном случае – тело цикла повторяется еще раз, после чего снова проверяется условие завершения цикла. Обобщенная форма записи оператора REPEAT … UNTIL выглядит следующим образом:

```
REPEAT
  Оператор_1;
  Оператор_2;
  . . . . . .
  Оператор_N;
UNTILУсловие;
```

Внимательно рассмотрите приведенный ниже пример программы, иллюстрирующий использование перечисляемого цикла. Наберите этот текст программы в ТП и выполните программу несколько раз для различных значений входных данных. Сформулируйте условие задачи, которую решает эта программа.

```
ProgramPoisk;
VARi,b,n: INTEGER;
VAR c: STRING;
VAR a: ARRAY[1..100] OF INTEGER;
BEGIN
  WriteLn('Vvedite n i b');
  ReadLn(n,b);
  WriteLn('Vvedite ', n, ' chisel');
  FOR i:=1 TO n DO
    Read(a[i]);
  c:='da'; i:=0;
```

```
 REPEAT
  i:=i+1;
 UNTIL a[i]=b;
 IF i>n
   THEN c:='net';
 WriteLn(c);
END.
```

Для создания циклов в ТП существует третья и последняя конструкция – WHILE … DO. Общий вид этой конструкции следующий:

```
WHILE Условие DO
BEGIN
 Оператор_1;
 Оператор_2;
 . . . . . .
 Оператор_N;
END;
```

В конструкции WHILE … DO проверка условия выхода выполняется в начале, а не в конце цикла, поэтому, если условие не удовлетворяется до начала выполнения цикла, то тело цикла игнорируется и выполняется оператор, стоящий сразу же после окончания тела цикла.

Задача. Решите рассмотренную выше задачу с использованием конструкции цикла WHILE … DO, вместо конструкции REPEAT … UNTIL.

Наберите в ТП текст программы, приведенной ниже. Исполните ее на компьютере при нескольких наборах входных данных. Выясните, что делает эта программа.

```
PROGRAMFIB;
VARn,i,F1,F2,F: INTEGER;
BEGIN
ReadLn(n);
IF n=1
 THEN F:=0
 ELSE IF n=2
       THEN F:=1
       ELSE
        BEGIN
         F1:=0; F2:=1; i:=2;
         WHILE i<n DO
          BEGIN
```

```
            F:=F1+F2; F1:=F2; F2:=F; i:=i+1;
          END;
        END;
          WriteLn('n = ', n, '  Fib = ', F);
END.
```

Наберите в ТП текст программы Max_FIB. Исполните эту программу для всех значений not 1 до 10. Сформулируйте условие задачи, которую решает эта программа.

```
PROGRAMMax_FIB;
VAR n,i,F1,F2,F: INTEGER;
BEGIN
   ReadLn(n);
   F1:=0; F2:=1; F:=1;
   WHILE F<=n DO
      BEGIN
        F:=F1+F2; F1:=F2; F2:=F; i:=i+1;
      END;
   WriteLn('n = ', n, ' Max_Fib = ', F1);
END.
```

Лабораторная работа 4

Цель работы: Приобретение навыков написания программ, использующих символьные переменные.

Для описания символьных переменных используется тип CHAR. Символьным переменным можно присваивать значения символа. Их можно сравнивать друг с другом или с символом. При сравнении символов сравниваются их ASCII-коды. Считается, что один символ больше другого только в том случае, если он имеет больший ASCII-код. Например, 'A' меньше 'B', поскольку ASCII-код символа 'A' равен 65, а ASCII-код символа 'B' равен 66. В таблице, приведенной ниже, перечислены функции, которые могут применяться при работе с символами.

Функция	Назначение
Chr(X: BYTE): CHAR	Возвращает символ, соответствующий ASCII-коду числа X.
Ord(X: CHAR): BYTE	Возвращает число, соответствующее символу Xв

	ASCII-таблице.
UpCase(X: CHAR): CHAR	Преобразует символы из строчных букв в прописные.
Pred(X: CHAR): CHAR	Возвращает символ, который предшествует в ASCII-таблицесимволу X.
Succ(X: CHAR): CHAR	Возвращает символ, который следует в ASCII-таблице за символом X.

Наберите в ТП программу, приведенную ниже. Выполните ее. Сформулируйте задачу, которую она решает.

```
PROGRAM Table_of_Char;
VAR i: BYTE;
BEGIN
   FOR i:=0 TO 255 DO Write(Chr(i):2);
END.
```

Внимательно рассмотрите приведенный ниже пример программы. В этой программе используется функция ReadKey, которая читает символ с клавиатуры. Для того чтобы функция ReadKey работала, в программе оператором **USESCrt;** подключен модуль Crt, в котором находится описание этой функции. Наберите этот текст программы в ТП и выполните программу несколько раз, нажимая различные клавиши на клавиатуре. Сформулируйте условие задачи, которую решает эта программа.

```
PROGRAMKlavihi;
USESCrt;
VARch: CHAR;
BEGIN
   REPEAT
   ch:=ReadKey;
   Write(ch: 2);
  UNTIL (ch='n') OR (ch='N');
  WriteLn;
END.
```

Для обработки строк (цепочек символов) в ТП существует специально предназначенный тип данных STRING (строка). Переменная типа STRING состоит из цепочки символов, то есть элементов типа CHAR. Поэтому этот тип данных занимает промежуточное место между простыми и структурированными типами данных.

При описании строковой переменной используется зарезервированное слово STRING. В квадратных скобках за ним может следовать максимальный

размер строки. Если этот размер отсутствует, то считается, что строка имеет размер, равный 255. Примеры объявления переменных типа STRING:

```
VAR
Stroka: STRING;
Family: STRING[15];
```

Существует два пути обработки переменных типа STRING. Первый путь предполагает обработку всей строки как единого целого, то есть единого объекта. Второй путь рассматривает строку как составной объект, состоящий из отдельных символов, то есть элементов типа CHAR, которые при обработке доступны каждый в отдельности. Для обращения к конкретному элементу строки указывается в квадратных скобках номер его позиции в строке.

Примеры операций со строками:

```
Family:='Bil';        {1 оператор}
Family:= Family + ' ' + 'Getc';   {2 оператор}
Family:= '  ';{3 оператор}
Family[1]:= 'A';     {4 оператор}
Family[2]:= #49;   {5 оператор}
```

В результате выполнения первого оператора строка Family будет содержать текст – Bil. После выполнения второго оператора к тексту Bil справа будет добавлен пробел и текст Getc. Выполнение третьего оператора приведет к тому, что строка Family будет содержать два пробела (это не означает, что она станет пустой). Четвертый оператор на первой позиции строки Family разместит символ A, а пятый оператор на второй позиции строки Family разместит цифру 1, поскольку #49 это есть ASCII-код цифры 1.

Для обработки строк в ТП имеется целый ряд функций и процедур. Ниже рассмотрим наиболее важные из них.

Функция Length позволяет определить фактическую длину текстовой строки, хранящейся в указанной переменной (а не величину предельного размера строки, установленную при декларации).

Функция UpCase преобразовывает строчную литеру в прописную. Эта функция рассчитана на обработку отдельного символа. Поэтому для обработки строки символов с помощью этой функции приходится организовывать цикл. Русские литеры не могут обрабатываться этой функцией.

Функция Copy предназначена для копирования фрагмента некоторой строки из одной переменной в другую. Вызывая функцию, необходимо указывать следующие параметры:

– имя строки, из которой должен извлекаться копируемый фрагмент;

– позицию в строке, начиная с которой будет копироваться фрагмент;

– число копируемых символов.

Следует иметь в виду, что сообщения об ошибке не будет, если начальная или конечная позиции копируемого фрагмента находятся вне пределов исходной строки символов. Результатом выполнения операции в первом случае будет строка нулевой длины, во втором – фрагмент от начальной позиции копирования до конца исходной строки.

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

Процедура Insert может быть использована для вставки фрагмента из одной строки в другую. При этом задается номер позиции в строке, с которой будет вставляться фрагмент.

Процедура Delete позволяет удалить из строки символов фрагмент. В качестве аргументов в этой процедуре указывается имя строковой переменной, из которой будет удаляться фрагмент, номер позиции, с которой расположен удаляемый фрагмент, и длина удаляемого фрагмента.

Наберите текст программы, приведенной ниже. Исполните ее и сформулируйте задачу, которую программа решает.

```
PROGRAM Cherta;
VAR
  stroka: STRING[60];
  dlin_str: BYTE;
BEGIN
  stroka:='_____'+
  '_____';
  FOR dlin_str:=1 TO 12 DO
    BEGIN
      stroka[0]:=Chr(dlin_str*5);
      WriteLn(stroka);
      WriteLn;
END;
END.
```


Лабораторная работа 5

Цель работы: Приобретение навыков написания программ с использованием различных форматов вывода данных.

Краткие теоретические сведения

В процедурах вывода Write и WriteLn имеется возможность записи выражения, определяющего ширину поля вывода. В приведенных ниже форматах используются следующие обозначения:

I, p, q – целочисленное выражение;

R –выражение вещественного типа;

B – выражение булевского типа;

Ch –выражение символьного типа;

S –выражение строкового типа;

-цифра;

* - знак “+” или “-“;

_-пробел.

I-выводится десятичное представление величины I, начиная с позиции расположения курсора.

Значение I	Выражение	Результат
134	Write (I);	134
287	Write (I,I,I);	287287287

I:p -выводится десятичное представление величины I в крайние правые позиции поля шириной p.

Значение I	Выражение	Результат
134	Write (I: 6);	_ _ _134
1	Write (I: 1);	_ _ _ _ _ _ _ _ _ 1
312	Write (I+I:7)	_ _ _ _624

R- в поле шириной 18 символов выводится десятичное представление величины R в формате с плавающей точкой. Если R>=0.0, используется формат _#.##########E*##.

Если R<0.0, формат имеет вид: -#.##########E*##.

Значение R	Выражение	Результат
715.432	Write (R);	_ _ 7.1543200000E+02
-1.919E+01	Write (R);	_ -1.9190000000E+01
567.986	Write (R/2);	_ _ 2.8399300000E+02

R:p –в крайние правые позиции поля шириной p символов выводится десятичное представление значения R в формате с плавающей точкой. Если R>=0.0, используется формат _ _…_##..#E*##, причем минимальная длина поля вывода составляет 7 символов. Если R<0.0, формат имеет вид:

_ _ …_--#.##..#E*##. Минимальная длина поля вывода 8 символов. После десятичной точки выводится, по крайней мере, одна цифра.

Значение R	Выражение	Результат
511.04	Write (R:15);	5.110400000E+02
-511.04	Write (R:15);	-5.11040000E+02
46.78	Write (-R:12);	-4.67800E+01

R:p:q –в крайние правые позиции поля шириной p символов выводится десятичное представление значения R в формате с фиксированной точкой,

причем после десятичной точки выводится q цифр (0<=q<=24), представляющих дробную часть числа. Если q=0, ни дробная часть, ни десятичная точка не выводится. Если q>24, то при выводе используется формат с плавающей точкой.

Значение R	Выражение	Результат
511.04	Write (R:8:4);	511.0400
-46.78	Write (R:15);	_-46.78
-46.78	Write (R:9:4);	_-46.7800

Ch-начиная с позиции курсора выводится значение Ch.

Значение Ch	Выражение	Результат
’X’	Write (Ch);	X
‘Y’	Write (Ch);	Y
’!’	Write (Ch, Ch, Ch);	!!!

Ch:p-вкрайнююправую позицию поля шириной p выводится значение Ch.

Значение Ch	Выражение	Результат
’X’	Write (Ch:3);	_ _X
‘Y’	Write (Ch:5);	_ _ _ _Y
’!’	Write (Ch:2, Ch:4);	_!_ _ _!

S- начиная с позиции курсора, выводится значение S (строка или массив символов, если его длина соответствует длине строки).

Значение S	Выражение	Результат
’DayN’	Write (S);	DayN
‘Ведомость 11’	Write (S);	Ведомость 11
’RRRDDD’	Write (S, S);	RRRDDDRRRDDD

S:p- значение S выводится в крайние правые позиции поля шириной p символов.

Значение S	Выражение	Результат
’DayN’	Write (S:10);	_ _ _ _ _ DayN
‘Ведомость 11’	Write (S:13);	_Ведомость 11
’RRRDDD’	Write (S:7, S:7);	_RRRDDD_RRRDDD

B- выводится результат выражения B True или False, начиная с текущей позиции курсора.

Значение B	Выражение	Результат
TrueWrite (B);		True
False	Write (B, not B);	False True

B:p- в крайние правые позиции поля шириной p символов выводится результат булевского выражения B True или False.

Значение B	Выражение	Результат
TrueWrite (B:6);		_ _ True

False Write (B:10); _ _ _ _ _False

True Write (B:5,not B:7); _True_ _False

Оператор записи WriteLn аналогичен процедуре Write, но после вывода последнего в списке значения для текущей процедуры WriteLn происходит перевод курсора к началу следующей строки.

Процедура WriteLn, записанная без параметров, вызывает перевод строки.

Пример программы с использованием процедур ввода-вывода данных с различными форматами выводимых данных.

```
Program Demo;
    VarA,B,S:Integer;
Begin
    Writeln('Введите сторону A = ');
    Readln(A);
    Writeln('Введитесторону B = ');
    Readln(B);
    S:=A*B;
    Writeln('-----------------------------------------');
    Writeln(' | СторонаA |  | Сторона B |   | Площадь | ');
    Writeln('-----------------------------------------');
    Writeln('|',A:7,B:11,S:11, '|' :5);
    Writeln('-----------------------------------------');
End.
```

В результате работы данной программы на экране будет изображена следующая таблица:

```
-----------------------------------------------------
| Сторона A |  | Сторона B |   | Площадь |
-----------------------------------------------------
|       8              4                 32      |
-----------------------------------------------------
```

Каждая строка этой таблицы будет печататься с первой позиции новой строки экрана.

Контрольные вопросы

Общие сведения о форматах выводимых данных.

Форматы I, I:p. Примеры.

Форматы R, R:p, R:p:q. Примеры.

Форматы Ch, Ch:p. Примеры.

Форматы S, S:p. Примеры.

Форматы B, B:p. Примеры

Лабораторная работа 6

Цель работы: Приобретение навыков написания программ с использованием условных операторов.

Условные операторы

Условные операторы предназначены для выбора к исполнению одного из возможных действий (операторов) в зависимости от некоторого условия (при этом одно из действий может быть пустым, т. е. отсутствовать). В качестве условий выбора используется значение логического выражения.

В Турбо Паскале имеются два условных оператора: *if и case.*

Оператор условия if

Оператор условия *if* является одним из самых популярных средств, изменяющих естественный порядок выполнения операторов программы.

Он может принимать одну из следующих форм:

– if <условие> then <оператор1>
else <оператор2>;

– if <условие> then <оператор>;

В переводе с английского языка данные форматы можно определить как:

– ЕСЛИ<условие>ТО<оператор1>ИНАЧЕ<оператор2>
– ЕСЛИ<условие>ТО<оператор>

Оператор условия *if* выполняется следующим образом. Сначала вычисляется выражение, записанное в условии. В результате его вычисления получается значение булевского типа.

В первом случае, если значение выражения есть *True* (истина), выполняется *<оператор1>*, указанный после слова *then* (в переводе –“то”). Если результат вычисления выражения в условии есть *False* (ложь), то выполняется *<оператор2>*.

Во втором случае, если результат выражения *True*, выполняется *<оператор>*, если *False* - оператор, следующий сразу за оператором *if*. Операторы *if* могут быть *вложенными*.

Пример фрагмента программы с оператором условия if:

```
…
Read(Ch) ;
if Ch='N' then Parol:= True
else  Parol:= False;
Read(X) ;
if Parol = True then
if X = 100 then Write('Парольикодправильные')
else
```

```
begin
   Writeln('Ошибкавкоде');
   Halt(1)
end;
        …
```

В данном примере с клавиатуры считывается значение переменной символьного типа *Ch*. Затем проверяется условие *Ch='N'*. Если оно выполняется, то переменной *Parol* булевского типа присваивается значение *True*, если условие не выполняется, *False*. Затем с клавиатуры считывается значение кода *X*. Далее оператор if проверяет условие Parol = True. Если оно имеет значение *True*, то выполняется проверка введенного пароля оператором *if X=100*. Если условие *X=100* имеет значение *True*, то выводится сообщение "Пароль и код правильные", и управление в программе передается на оператор, следующий за словом *end*, если оно имеет значение *False*, выполняется составной оператор, стоящий после, слова *else*, который выводит на экран видеомонитора сообщение "Ошибка в коде", и вызывает стандартную процедуру *Halt(1)* для остановки программы.

Особенность применения оператора if. При использовании вложенных условных операторов может возникнуть синтаксическая неоднозначность, например:

if условие1 then if условие2 then <оператор1> else <оператор2>

Возникающая двусмысленность, к какому оператору *if* принадлежит часть *else<оператор2>,* разрешается тем, что служебное слово *else* всегда ассоциируется (связывается) с ближайшим по тексту служебным словом *if,* которое еще не связано со служебным словом *else*.

В связи с этим следует проявлять аккуратность при записи вложенных операторов условия.

Пример 1. Составить программу, которая вычисляет частное двух целых чисел. В связи с тем, что делить на нуль нельзя, организовать контроль ввода данных.

Для контроля вводимых значений делителя используем оператор условного перехода *if ... then ... else.*

Текст программы может выглядеть следующим образом:

```
program Primer1;
var
A, B : integer;
Rezult: real;
Begin
```

```
Write('Введите значение делимого A: ');
Read(A) ;
Write('Введите значение делителя B: ');
Read(B) ;
if B=0          {Контроль ввода числа B}
                then Writeln('На нуль делить нельзя')     {Условие выполнено}
                else
                {Условие не выполнено}
                begin
        Rezult := A / B;
        Writeln('Частное чисел ',A,' и ' ,B, ' = ', Rezult);
                end;
end.
```

Оператор выбора case

Если один оператор *if* может обеспечить выбор из двух альтернатив, то оператор выбора *case* позволяет сделать выбор из произвольного числа имеющихся вариантов. Он состоит из выражения, называемого *селектором* (*selection — выбор альтернативы*), и *списка параметров*, каждому из которых предшествует список *констант выбора* (список может состоять и из одной константы).

Формат записи оператора case:

```
case <выражение-селектор> of
      <список1>: <оператор1; >
      <список2>: <оператор2; >
      …
      <списокN>: <операторN>
else <оператор>
end;
```

Оператор *case* работает следующим образом. Сначала вычисляется значение выражения-селектора, затем обеспечивается реализация того оператора, константа выбора которого равна текущему значению селектора. Если ни одна из констант не равна текущему значению селектора, выполняется оператор, стоящий за словом *else.* Если слово *else* отсутствует, активизируется оператор, находящийся за словом *end*, т. е. первый оператор за границей *case.*

Селектор должен относиться к одному из целочисленных типов (находящихся в диапазоне — 32768..32767): булевскому, литерному или пользовательскому. Список констант выбора состоит из произвольного количества значений, или диапазонов, отделенных друг от друга запятыми.

Границы диапазона записываются двумя константами через разграничитель "..". Тип констант в любом случае должен совпадать с типом селектора. В синтаксическом описании, приведенном выше, предполагается использование одного оператора для каждой альтернативы, но при необходимости можно задать несколько операторов, сгруппировав их в составной оператор. В то же время ветвь *else* допускает использование последовательности операторов, разделенных символом ";".

Правила использования оператора case. При использовании оператора выбора *case* должны выполняться следующие правила:

1. Значения выражения "переключателя", записанного после служебного слова *case,* должны принадлежать дискретному типу (лат. discretus — прерывистый, дробный, состоящий из отдельных частей); для целого типа они должны лежать в диапазоне *integer*.

2. Все константы, предшествующие операторам альтернатив, должны иметь тип, совместимый с типом выражения.

3. Все константы в альтернативах должны быть уникальны в пределах оператора варианта (т. е. повторения констант в альтернативах не допускаются); диапазоны не должны пересекаться и не должны содержать констант, указанных в данной или других альтернативах.

Формы записи оператора case.

Селектор интервального типа:

```
case I of
1..10 : Writeln ('число ', I: 4, ' вдиапазоне 1- 10');
11.. 20 : Writeln ('число ', I:4, ' в диапазоне 11-20');
21.. 30 : Writein ('число', I:4, ' в диапазоне 21-30')
else Writeln ('число ', I:4, ' вне пределов контроля');
end;
```

Селектор целочисленного типа:

```
caseIof
1 :Z := I + 10;
2 :Z := I + 100;
3 :Z :=I + 1000;
end;
```

Селектор перечисляемого пользовательского типа:

```
var
     Season: (Winter, Spring, Summer, Autumn) ;
begin
case Season of
  Winter: Writeln('Winter');
```

```
  Spring: Writeln('Spring');
  Suauner: Writeln (' Summer') ;
  Autumn: Writeln('Autumn')
 end; {конец case}
end;
```

Пример 2. Составить программу, которая по введенному пользователем номеру дня недели выводит на экран его название.

```
program Day_Week;
var Day : byte;
begin
Write ('Введите номер дня недели: ');
Readln(Day) ;
caseDayof{Вычисление значения селектора и выбор}
1:  Writeln('Понедельник') ;
2:  Writeln('Вторник') ;
3:  Writeln('Среда');
4:  Writeln('Четверг');
5:  Writeln<'Пятница');
6:  Writeln('Суббота' ) ;
else
        Writeln('Воскресенье');
        end;
end.
```

В данном примере на экран выводится приглашение: 'Введите номер дня недели:', с клавиатуры считывается целочисленное значение дня недели и присваивается переменной *Day*. Затем в зависимости от значения селектора DAY обеспечивается реализация того оператора, константа выбора которого равна текущему значению селектора. Например, если значение *Day* равно 3, то реализуется оператор *Writeln('Среда')*. Если значение *Day* равно 7, а ни одна из констант не равна этому значению селектора, то выполняется оператор, стоящий за словом *else* (на экран выводится текст *'Воскресенье'*). Если слово *else* отсутствует, активизируется оператор, находящийся за словом *end*, т. е. первый оператор за границей *case*. Если значение *Day* не равно значению ни одной из констант выбора *(например, Day=8 или Day=0),* то активизируется оператор, находящийся за словом *end*, т. е. первый оператор за границей *case* - оператор *end*.

Контрольные вопросы

Условные операторы. Оператор if. Формат, описание. Основные правила использования. Особенности использования вложенного оператора if.

Примеры использования оператора if.

Оператор выбора case. Формат, описание.

Правила использования оператора case. Примеры использования.

Лабораторная работа 7

Цель работы: формирование знаний и умений по работе со структурными типами данных. Приобретение навыков написания программ с использованием массивов.

Краткие теоретические сведения

Пусть требуется составить программу, которая формирует одномерный массив вводом с клавиатуры, находит в массиве элементы, заданные пользователем, подсчитывает их количество и выводит номер первого найденного элемента.

В разделе описания констант укажем значение константы *Count=10*. В программе значение этой константы определяет количество элементов массива. Употребление константы в описании размеров массива предпочтительнее, так как в случае изменения размеров массива не нужно будет вносить изменения во весь текст программы, а достаточно только один раз указать в разделе описания констант новое значение константы *Count*. Тогда раздел описания констант и переменных в программе будет таким: *Const Count=10;*

В связи с этим описание массива зададим так: *M : array [1..Count] ofByte;*

Введем переменные целого типа: *N* - значение искомого элемента; *A* - номер первого элемента массива, значение которого равно *N*; *B* - количество таких элементов в массиве; *I* - переменная, выполняющая функции параметра цикла и одновременно служащая указателем номера очередного элемента массива.

В связи с тем, что ни один подходящий элемент еще не найден, присвоим переменным *A* и *B* значение 0. Затем выведем на экран приглашение на ввод значения искомого элемента считаем это значение с клавиатуры. На Паскале это запишется следующим образом:

Write('Введите значение элемента массива для поиска : ');

Readln(N) ;

Поиск элемента массива, значение которого равно введенному числу *N*, выполняется в циклическом сравнении значений всех элементов от первого до последнего со значением числа *N*, поэтому запишем его в виде цикла с параметром.

Оператор *if M[I] = N then* ...выполняет сравнение значения очередного элемента массива с заданным значением *N*. Если условие *M[I] = N* выполняется, то счетчик числа найденных элементов *B* увеличивается на единицу. Так как требуется найти номер первого элемента, т. е. при первом выполнении условия *M[I] = N* запомнить номер данного элемента, то можно записать: *if B = 0 then A := I.*

Тогда блок поиска нужного элемента можно записать так:

```
for I := 1 to Count do
  if M[I] = N then
                begin
                      if B =0 then A := I;
                      B:= B+1;
                end;
```

В заключительной части программа должна вывести на экран сообщение о том, что в массиве нет искомых элементов, или сообщение о количестве элементов массива, имеющих значение, равное *N*, и напечатать номер первого такого элемента. Это можно записать следующим образом:

```
ifB=0 then
riteln('Нет таких элементов в массиве')
else
  begin
        Writeln('Количество элементов массива, имеющих значение',N,'  - ', B) ,
        Writeln('Первый элемент, совпадающий с заданным ' , A) ;
end ;
```

В целом текст программы может быть таким:

```
program Find_Elem;                {Поискэлементавмассиве}
Const
Count =10;
Var
M : array [1..Count] of byte;
N, A, B, I : Byte;
Begin                             {Основнаяпрограмма}
      for I:=1 to Count do                {ввод элементов массива}
      begin
      writeln('Введите  ',i,' элементмассива');
      readln(M[I]);
      end;                        {конецввода}
      writeln('Введенныймассив: ');
      for I:=1 to Count do        {выводэлементовмассива}
```

```
                begin
                        write(' ',M[I],' ');
                end;                    {конец вывода}
Writeln;
A := 0;                 {Нет элемента с таким значением}
B := 0;                 {Пока не найдено ни одного элемента}
Write('Введите значение элемента массива для поиска: ');
Readln(N) ;
for I := 1 to Count do {Поискэлемента, значениекоторого =N}
  if M[I] == N then
        begin
                if B = 0 thenA := I;{Запомнить номер первого элемента, равного
        N}
                B := B + 1;     {Увеличить число найденных элементов на 1}
        end;
  ifB=0 then
        Writeln('Нет таких элементов в массиве')
        else
        begin
                Writeln('Количество элементов массива, имеющих    значение',N,'-
        ',B),
                Writeln('Первый элемент, совпадающий с заданным - ', A) ;
        end;
end.
```

Пример программы нахождения в одномерном массиве максимального элемента

Пусть требуется составить программу, которая формирует одномерный массив случайных чисел, выполняет поиск максимального элемента массива, а затем выводит на экран его значение и порядковый номер в массиве.

В разделе описания запишем размер массива *Count=20*, опишем массив целых чисел *M* следующим образом: *M : array [I..Count] of byte* . Используем целые переменные для хранения значений максимального элемента массива – *Max*, его индекса –*Numer_Max*.

Перед началом поиска максимального элемента допустим, что его первый элемент и является максимальным элементом, а его индекс указывает позицию максимального элемента в массиве.

Это запишется так:

```
Max:= M[1];          {Считать 1-й,элемент максимальным}
Numer_Max:=1;     {Запомнить номер максимального элемента}
```

Повторяющийся просмотр массива с поиском максимального элемента, начиная со второго, выполняется оператором повтора с параметром, который одновременно указывает индекс очередного элемента. Сравнение очередного элемента массива с максимальным осуществляется оператором: *If M[I] > Max then…*

Если очередной элемент массива больше, чем максимальный, то следует считать его значение максимальным и запомнить его индекс.

Данный фрагмент программы запишется таким образом:

```
forI := 2 toCountdo          {Проверить все элементы, начиная со второго)
 begin
       ifM[I] >Maxthen   {Если очередной (I-й) элемент массива больше чем
 Max}
       begin
       Max := M[I];      {то считать максимальным 1-й элемент}
       Numer_Max:=I;  {и запомнить его порядковый номер}
       end;
end;
```

В заключительной части программы запишем вывод результата поиска максимального элемента массива:

```
Writeln('Максимальный элемент — ', Max);
Writeln('Он расположен на ', Numer_Max, ' месте');
```

Полный текст программы получится таким:

```
programMax_Elem;          {Поиск максимального элемента массива}
Const
Count =20;
Var
M : array [I..Count] of byte;
Max, I, Numer_Max : byte;
Begin                     {Основнаяпрограмма}
      for I:=1 to Count do          {ввод элементов массива}
            begin
                  writeln('Введите  ',i,' элементмассива');
                  readln(M[I]);
            end;                    {конецввода}
      writeln('Введенныймассив: ');
      for I:=1 to Count do {выводэлементовмассива}
            begin
                  write(' ',M[I],' ');
            end;                    {конец вывода}
```

```
Writeln;
Max := M[1] ;        {Считать 1-й элемент максимальным}
Numer_Max:=1;  {Запомнить номер максимального элемента}
forI := 2 toCountdo{Проверить все элементы, начиная со второго}
  begin
        ifM[I] >Maxthen{Если очередной (I-й) элемент массива больше чем
  Max}
        begin
              Max := M[I];      {то считать максимальным 1-й элемент}
              Numer_Max:=I; {и запомнить его порядковый номер}
        end;
  end;
Writeln('Максимальный элемент — ', Max);
Writeln('Он расположен на ', Numer_Max, '  месте');
end.
```

Контрольные вопросы

Алгоритм нахождения элемента в одномерном массиве. Пример программы.

Алгоритм нахождения максимального элемента одномерного массива. Пример программы.

Лабораторная работа № 8

Цель работы: формирование знаний и умений по работе со структурными типами данных. Приобретение навыков написания программ с использованием массивов.

Краткие теоретические сведения

Рассмотренный ранее линейный массив представлял собой вектор, т.е. линейную последовательность чисел, например:

1 2 3 –5 0 1 3 6 …

Для определения такого массива необходимо задать 1 параметр – количество элементов (например *n*). Поэтому для обработки одномерного массива достаточно было организовать цикл по одной переменной (например *i*): *for i:=1 to n do mas[i]:=…*

Двумерный массив представляют собой *матрицу*, для определения которой необходимо задать два параметра – количество строк (например *n*) и количество столбцов (например *m*). Если количество строк – 2, количество столбцов – 3, тогда матрица будет выглядеть,например, так:

1 3 -5

2 3 4

Следовательно, для обработки матрицы (двумерного массива) требуется организовать цикл по двум переменным (например *i* и*j*):

```
for i:=1 to n do
begin
        for j:=1 to m do
    begin
    mas[i,j]:=…
end;
end;
```

Пример программы нахождения суммы элементов двумерного массива

Пусть требуется составить программу, которая в двумерном массиве находит сумму его элементов и выводит ее на экран. Количество элементов массива (строк и столбцов) пользователь вводит с клавиатуры.

В начале программы опишем двумерный массив с именем *mas*, состоящий из 10 строк и 10 столбцов (максимально) типа *integer*. Затем опишем переменную *S*, обозначающую сумму элементов, *i, j*-индексы соответственно строки и столбца, *n,m*-количество строк и столбцов.

В основной программе пользователь задает размерность матрицы (количество строк и столбцов). Затем во вложенном цикле (*по i и по j*) осуществляется ввод с клавиатуры элементов матрицы, а затем вывод на экран введенного пользователем массива.

После того, как массив был введен, с ним можно непосредственно начинать работать. В данном примере необходимо найти сумму элементов двумерного массива.

Любое нахождение суммы начинается с *обнуления* переменной суммы: *s:=0.* Затем в цикле *по i и по j* вычисляется сумма:

```
for i:=1 to n do
begin
for j:=1 to m do
        begin
                s:=s+mas[i,j];
        end;
end;
```

В целом текст программы выглядит так:

```
Program matrix;
Uses crt;
```

```
var
mas:array [1..10,1..10] of integer;
s:integer;
i,j,n,m:integer;
Begin
Clrscr;
writeln('Введите количество строк n:');
readln(n);
writeln('Введите количество столбцов m:');
readln(m);
for i:=1 to n do        {ввод элементов двумерного массива}
        begin
        for j:=1 to m do
                begin
writeln('Введите  ',i,',',j,'-йэлементматрицы: ');
readln(mas[i,j]);
                end;
        end;
writeln('Введенный массив: ');
for i:=1 to n do        {вывод элементов двумерного массива}
        begin
          for j:=1 to m do
          write(mas[i,j]:5);
                end;                            {конецвывода}
s:=0;                                           {обнулениесуммы}
for i:=1 to n do
 begin
        for j:=1 to m do
                begin
                        s:=s+mas[i,j];{вычислениесуммыэлементов}
                end;
 end;
write('Summa:');
write('S= ',S);         {вывод на экран полученной суммы}
 End.
```

Пример программы нахождения количества положительных и отрицательных элементов двумерного массива

Пусть требуется составить программу, которая подсчитывала бы количество положительных и количество отрицательных элементов двумерного массива и записывала бы положительные и отрицательные элементы в два разных массива.

В начале программы опишем двумерный массив с именем *mas*, состоящий из 10 строк и 10 столбцов (максимально) типа *integer*, а также массивы *B* и*C* для положительных и отрицательных элементов соответственно. Затем опишем переменные *k, l*-количество положительных и отрицательных элементов соответственно, *i, j* - индексы соответственно строки и столбца, *n,m*-количество строк и столбцов.

Проверка на положительность элементов может выглядеть так:

```
for i:=1 to n do
  begin
        for j:=1 to m do
              begin
              if mas[i,j]>0 then…
```

Если условие выполняется (i,j –й элемент является положительным), то количество положительных элементов k увеличивается на 1 и заполняется массив B:

```
if mas[i,j]>0 then
   begin
     k:=k+1;
     B[k]:=mas[i,j];
   end;
```

Аналогично выполняется проверка на отрицательность, количество отрицательных элементов *l* увеличивается на 1 и заполняется массив C.

В целом текст программы выглядит так:

```
Program matrix_2;
Uses crt;
Var
mas:array [1..10,1..10] of integer;
B:array [1..10] of integer;
C:array [1..10] of integer;
i,j,k,l,n,m:integer;
Begin
Clrscr;
writeln('Введите количество строк n:');
readln(n);
writeln('Введите количество столбцов m:');
```

```
  readln(m);
  {ввод-вывод элементов матрицы mas}
for i:=1 to n do
   begin
        for j:=1 to m do
          begin
               if mas[i,j]>0 then
                 begin
                      k:=k+1;
                      B[k]:=mas[i,j];
                 end;
          end;
               if mas[i,j]<0 then
                 begin
                      l:=l+1;
                      C[l]:=mas[i,j];
               end;
        end;
{Вывод на экран полученных массивов B и C}
write('Массив из положительных : ');
for i:=1 to k do
write('  ',B[i]);
writeln;
write('Массив из отрицательных : ');
for i:=1 to l do
write('  ',C[i]);
readln;
  End.
```


Контрольные вопросы

Принципы работы с двумерными массивами.

Примеры программ для решения задач на обработку двумерных массивов.

Лабораторная работа № 9

Цель работы: формирование знаний и умений по работе с подпрограммами. Приобретение навыков написания программ с использованием функций.

Краткие теоретические сведения

Понятие структурного программирования. Подпрограммы.

Значительное увеличение сложности задач, решаемых с помощью ЭВМ, приводит к увеличению размеров и сложности программ, что порождает дополнительные трудности при их разработке и отладке. Увеличение продолжительности жизненного цикла программ приводит к тому, что с течением времени из-за изменения условий использования программ возникает необходимость их модификации, повышения их эффективности, удобства пользования ими.

Для разрешения возникших при этом проблем в практике программирования выработан ряд приемов и методов, которые принято называть *методами структурного программирования.*

Под ***структурным программированием*** понимают такие методы разработки и записи программы, которые ориентированы на максимальные удобства для восприятия и понимания ее человеком. При прочтении программы в ее следующих друг за другом фрагментах должна четко прослеживаться логика ее работы, т. е. не должно быть "скачков" на фрагменты программы, расположенные где-то в другом месте программы.

Структурное программирование — "программирование без go to", т. е. не используются операторы перехода без особой необходимости. В связи с этим отдельные фрагменты программы представляют собой некоторые логические (управляющие) структуры, которые определяют порядок выполнения содержащихся в них правил обработки данных. Любая программа получается построенной из стандартных логических структур, число типов которых невелико.

Основные логические структуры

Следование– последовательность операторов, групп операторов, выполняемых друг за другом в порядке их следования в тексте программы.

Ветвление– управляющая структура, которая в зависимости от выполнения заданного условия определяет выбор для исполнения одного из двух или более заданных в этой структуре групп операторов.

Повторение– цикл, в котором группа операторов может выполняться повторно, если соблюдается заданное условие.

Технология нисходящего программирования базируется на методе программирования "сверху вниз". Часто этот метод *называют методом пошаговой детализации*. Основой такого метода является идея постепенной декомпозиции исходной задачи на ряд подзадач. Сначала формулируется самая грубая модель решения, отдельные детали которой на первом этапе могут быть довольно расплывчатыми. По мере разработки программы, разбивая наиболее

неясные части алгоритма и добиваясь все более точных и детализированных формулировок получают более подробное решение, как бы опускаясь с большой высоты ниже и начиная при этом различать более мелкие детали. Решение отдельного фрагмента сложной задачи может представлять собой самостоятельный программный блок, называемый *подпрограммой.* Такой процесс детализации продолжается до тех пор, пока не станут ясны все детали решения задачи. В этом случае программу решения сложной задачи можно представить как иерархическую совокупность относительно самостоятельных фрагментов — подпрограмм.

Подпрограммой называют обособленную, оформленную в виде отдельной синтаксической конструкции и снабженную именем часть программы. Использование подпрограмм позволяет, сосредоточив в них подробное описание некоторых операций, в остальной программе только указывать имена подпрограмм, чтобы выполнить эти операции. Такие вызовы подпрограммы возможны неоднократно из разных участков программы, причем при вызове подпрограмме можно передать некоторую информацию (различную в разных вызовах), чтобы одна и та же подпрограмма выполняла решение подзадачи для разных случаев.

Функции в Паскале

За наличие подпрограмм как средства структурирования программ язык программирования Турбо Паскаль называется процедурно-ориентированным.

Подпрограммы в Турбо Паскале реализованы посредством процедур и функций. Имея один и тот же смысл и аналогичную структуру, процедуры и функции различаются назначением и способом их использования.

Функция — это независимая именованная часть программы, которую можно вызвать по имени для выполнения определенных действий. Структура функции повторяет структуру программы.

Особенности использования функции:

– функция передает в точку вызова скалярное значение;

– имя функции может входить в выражение как операнд.

Например, функция Chr(65) возвратит в точку вызова символ A (код ASCII – 65), Sqr(X) – возведет в квадрат значения целого или вещественного X и возвратит в точку вызова вычисленное значение квадрата числа X.

Все процедуры и функции языка Турбо Паскаль делятся на две группы: встроенные (стандартные) и определенные пользователем. Первые входят в состав языка и вызываются для выполнения по строго фиксированному имени. Вторые разрабатываются и именуются самим пользователем. Все стандартные средства расположены в специализированных библиотечных модулях, которые имеют системные имена.

Стандартные библиотечные модули

В систему Турбо Паскаль версии 6.0 и старше включены восемь модулей: *System, Crt, Dos, Graph, Graph3, Overlay, Printer, Турбо3* и специализированная библиотека *Турбо Vision.* Модуль *System* подключается *по умолчанию*, все остальные должен подключать программист с помощью зарезервированного слова *uses*.

Например:

UsesCrt, Dos, Printer;

Назначение каждого модуля:

System– содержащиеся в нем подпрограммы обеспечивают работу всех остальных модулей системы.

Crt– содержит средства управления дисплеем и клавиатурой компьютера.

Dos– включает средства, позволяющие реализовывать различные функции DOS.

Graph3– поддерживает использование стандартных графических подпрограмм версии Турбо Паскаль 3.0.

Overlay– содержит средства организации оверлейных программ.

Printer– обеспечивает быстрый доступ к печатающему устройству.

Turbo3– обеспечивает максимально возможную совместимость с версией Турбо Паскаль 3.0.

Graph– содержит пакет графических средств, обеспечивающих эффективную работу с адаптерами CGA, EGA, VGA, IBM 3270 и т.д.

Турбо Vision– библиотека объектно-ориентированных подпрограмм для разработки пользовательских интерфейсов.

Встроенные функции и процедуры

Модуль *System* подключается к программе автоматически, поэтому его имя не указывается в разделе *Uses*. По этой причине программе становятся доступны его встроенные процедуры и функции.

Арифметические процедуры и функции

Abs(X:real/integer): real/integer– вычисление абсолютной величины X. Тип результата совпадает с типом параметра.

ArcTan(X:real):real– вычисление угла, тангенс которого равен X радиан.

Cos(X:real) : real– вычисление косинуса X, параметр задает значение угла в радианах.

Exp(X:real) : real– вычисление экспоненты X, т.е. значение E в степени X. E является основанием натурального логарифма и равно 2.718282.

Frac(X:real):real– вычисление дробной части X.

Int(X:real):real– вычисление целой части X.

Ln(X:real):real– вычисление натурального логарифма X, т, е. логарифма по основанию e (e = 2.718282).

Pi:real– возвращает значение числа Пи (3.141592653897932385).

Sin(X:real):real– вычисление синуса X. Параметр задает значение угла в радианах.

Sqr(X)– возведение в квадрат значения целого или вещественного значения X. Тип результата совпадает с типом параметра.

Sqrt(X:real):real– вычисление квадратного корня из X.

Random:real– генерирует значение случайного числа из диапазона 0..0.99.

Random(I:word):word– генерирует значение случайного числа из диапазона 0..I.

Randomize– изменение базы генератора случайных чисел.

Скалярные процедуры и функции

Dec(X{,n})– процедура уменьшает значение целочисленной переменной X на величину n. При отсутствии необязательного параметра n значение X уменьшается на единицу.

Inc(X{,n})– процедура увеличивает значение целочисленной переменной X на n. При отсутствии необязательного параметра n значение X увеличивается на единицу.

Pred(S)– функция возвращает элемент, предшествующий S в списке значений типа. Тид результата совпадает с типом параметра. Если предшествующего S элемента не существует, возникает программное прерывание.

Succ(S)– функция возвращает значение, следующее за S в списке значений типа. Тип результата совпадает с типом параметра. Если следующее за S значение отсутствует, возникает программное прерывание.

Odd(I: integer): boolean– возвращает True, если I нечетное, и False, если I четное.

Функции преобразования типов

Chr(I:byte):char– возвращает символ стандартного кода обмена информацией с номером, равным значению I. Если значение параметра больше 255, возникает программное прерывание.

Ord(S):longint– возвращает порядковый номер значения S в множестве, определенном типом S.

Round(X:real):longint– возвращает значение X, округленное до ближайшего целого числа.

Trunc(X:real):longint– возвращает ближайшее целое число, меньшее или равное X, если X >= 0, и большее или равное X, если X< 0.

Процедуры управления программой

Delay(I:word)– задержка выполнения программы на I мс.

Exit– выход из выполняемого блока в окружающую среду. Если текущий блок является процедурой или функцией, выход производится во внешний блок. Если Exit указана в операторной части основной программы, программа прекращает работу, и управление передается системе программирования.

Halt(N:word)– прекращение выполнения программы и передача управления системе программирования (если выполнялся .PAS-файл) или DOS (если выполнялся .EXE-файл). N –код завершения программы, передаваемый в операционную систему.

RunError(ErrCode:word)– прекращение выполнения программы и генерация ошибки времени выполнения. ErrCode – параметр типа byte, содержащий номер ошибки.

Специальные процедуры и функции

FillChar(P,Dl,Z)– заполняет побайтно область основной памяти заданным значением (заполнителем). Является одной из самых быстродействующих процедур. Область начинается с первого байта указанной переменной P и имеет размер, заданный параметром Dl. P – переменная любого типа; Dl – целочисленное выражение, указывающее длину; Z – заполнитель, выражение литерного или байтового типа.

Hi(I:integer):byte– выделяет старший байт значения I и помещает его в младший байт результата. Старший байт результата равен 0.

Lo(I: integer): byte– выделяет младший байт значения I и помещает его в младший байт результата. Старший байт результата равен 0.

SizeOf(IT:integer):word– вычисляет объем основной памяти в байтах, которую занимает указанная переменная или тип. IT – идентификатор переменной или типа данных.

Swap(I: integer): integer– обменивает содержимое младшего и старшего байтов целочисленного выражения, заданного параметром I типа integer.

Вызов стандартной процедуры или функции

Для использования стандартной процедуры или функции к программе подключается тот или иной специализированный библиотечный модуль, в котором записана данная стандартная процедура или функция (исключение составляет модуль *System*, так как он подключается к программе автоматически), для чего имя специализированного библиотечного модуля указывается в разделе *uses*. Затем в программе записывается вызов процедуры или функции, для чего записывается ее имя и указываются фактические параметры, например: *Pi, Sin(X), Chr(125), Inc(X,5).* Так как после выполнения функции ее значение присваивается имени, то имя функции используется в выражении.

Контрольные вопросы

Понятие структурного программирования. Определение подпрограмм.

Функции. Особенности использования.

Стандартные библиотечные модули. Понятие стандартных функций и процедур.

Лабораторная работа № 10

Цель работы: формирование знаний и умений по работе с подпрограммами. Приобретение навыков написания программ с использованием функций.

Краткие теоретические сведения

Функция, определенная пользователем, состоит из заголовка и тела функции.Заголовок содержит зарезервированное слово function, идентификатор (имя) функции, заключенный в круглые скобки, необязательный список формальных параметров и типвозвращаемого функцией значения. Тело функции представляет собой локальный блок, по структуре аналогичный программе.

В целом структура функции, определенной пользователем имеет вид:

```
function<имя> (Формальные параметры) : <тип результата>;
const ...
type ...
var
begin
        <операторы>
end;
```

В разделе операторов должен находиться, по крайней мере, один оператор, *присваивающий имени функции значение*. В точку вызова возвращается результат последнего присваивания.

Обращение к функции осуществляется по имени с необязательным указанием списка аргументов. Каждый аргумент должен соответствовать формальным параметрам, указанным в заголовке, и иметь тот же тип.

Пример программы с использованием функции, определенной пользователем

Пусть требуется разработать программу вычисления выражения:

$$Z=(A^{5} + A^{-3})/2* A^{M},$$

в которой возведение в степень выполняется *функцией Step*.

```
program DemoFunc;
  Var
  M    : integer;
  A,Z,R :real ;
  {Функция вычисления  степени.N - степень, X – число, возводимое в
  данную степень. N, X — формальные параметры; результат,
  возвращаемый функцией в точку вызова, имеет вещественный тип}
  function Step(N : integer; X : real): real;
    Var
      I : integer;
      Y : real;
  begin
     Y:=1;
    forI:=1 toNdo{ЦиклвычисленияN—йстепеничислаX)
      Y:=Y*X;
      Step:=Y ;    {Присваивание функции результата вычисления степени}
  end; {Конец функции}
Begin{Начало основной программы}
  Write('Введите значение числа A и показатель степени M');
  Readln(A,M) ;
  Z:=Step(5,A) ; {Вызов функции с передачей ей фактических параметров
  N=5, X=A}
  Z:=Z+ Step(3,I/A); {Вызов функции с передачей ей фактических
  параметров N=3, X=1/A}
  ifM=0 thenR:=I{если число возводится в нулевую степень, то результат
  всегда равен 1}
  elseifM>0 thenR:=Step(M,A){Вызов функции Step с передачей ей
  фактических параметров M, A}
        elseR:=Step(-M,A); {Вызов функции с передачей ей фактических
        параметров: - M, отрицательная степень}
  Z:=Z/(2*R) ;
  Writeln(' Для A= ',A,'M= ',M,' Значение выражения= ',Z);
end.
```

В начале программы описываются переменная целого типа *M* и переменные вещественного типа *A, Z, R*, после этого описывается функция

вычисления степени числа *Step* с формальными параметрами *N* и *X,* результат, возвращаемый функцией в точку вызова, - вещественного типа.

В описании функции вводятся две *локальных (местных) переменныхI* и *Y*. Переменная *I* служит для подсчета числа повторений цикла, а в *Y* накапливается значение степени как произведения *N* одинаковых сомножителей. В заключение функции присваивается значение вычисленного произведения.

В начале выполнения основной программы на экран выводится запрос *"Введите значение числа A и показатель степени M"* и считывается с клавиатуры значение вещественного числа A и целого числа M.

Затем выполняется оператор:

```
Z:=Step(5,A);
```

Осуществляется вызов функции *Step* с передачей ей фактических параметров *5, A*. Их значения присваиваются формальным параметрам функции *N* и *X*. По окончании вычисления степени числа значение функции *Step*, вычисленное для фактических параметров *5* и *A*, присваивается переменной *Z*. Аналогично в операторе:

```
Z := Z + Step(3,l/A);
```

сначала осуществляется вызов функции *Step* с передачей ей фактических параметров *3, 1/A,* после чего значение переменной *Z* увеличивается на величину возвращенного в основную программу результата вычисления функции *Step*.

Операторы:

```
if M=0 then R:=1
        else if M>0 then R:=Step(M,A)
                    else R:=Step(- M,A);
```

проверяют условия *M=0, M>0* и в зависимости от их соблюдения либо присваивает переменной *R* значение 1 *(при M=O)*, либо выполняет вызов функции *Step* для *фактических значений M, A* или *-M, A*, а после вычисления значения функции *Step* присваивает его переменной R.

Оператор:

```
Z:=Z/(2*R);
```

выполняет вычисление значения выражения, а затем присваивает вычисленное значение переменной *Z*.

В заключение программы стандартная процедура *Writeln* выводит на экран сообщение о результате вычислений степени *M* числа *A*.

Контрольные вопросы

Структура функции, определенной пользователем.

Пример программы с использованием функции, определенной пользователем.

Лабораторная работа № 11

Цель работы: формирование знаний и умений по работе с подпрограммами. Приобретение навыков написания программ с использованием процедур.

Краткие теоретические сведения

Описание процедуры включает *заголовок (имя)* и *тело процедуры*. Заголовок состоит из зарезервированного слова *procedure*, идентификатора (имени) процедуры и необязательного, заключенного в круглые скобки, списка формальных параметров с указанием типа каждого параметра. *Имя процедуры*– идентификатор, уникальный в пределах программы. *Тело процедуры* представляет собой локальный блок, по структуре аналогичный программе.

Описания меток, констант, типов и т. д. действительны только в пределах данной процедуры. В теле процедуры можно использовать любые глобальные константы и переменные.

Общая структура описания процедур:

```
procedure<имя> (Формальные параметры);
const ... ;
type . . . ;
var  . . . ;
begin
        <операторы>
end ;
```

Пример программы с использованием процедуры, определенной пользователем

В качестве примера опишем процедуру, которая прерывает выполнение программы и выдает соответствующее сообщение об ошибке.

```
procedure Abort(Msg: string);
begin
Writeln('Ошибка: ', Msg);
Halt(1);
end ;
```

В данной процедуре пользователя использована переменная *Msg* типа *string*, в которой хранится текст сообщения о характере ошибки, вызвавшей прерывание программы. Для прерывания выполнения программы используется стандартная процедура *Halt* из стандартного библиотечного модуля *System*.

Процедура не может выполниться сама, ее необходимо вызвать по имени и указать фактические параметры того же типа, что и формальные. Количество и тип формальных параметров равны количеству и типу фактических параметров.

В качестве примера приведем фрагмент программы, в котором используется описанная выше процедура *Abort*.

```
programDemoProc;          {Подсчет суммы десяти введенных целых
          положительных чисел: если будет введено отрицательное число,
          прервать выполнение}
   const
   Limit= 10;  {Ограничение на количество вводимых чисел)
   var
   Count, Item, Sum : integer;
 procedure Abort(Msg: string);     {описаниеиреализацияпроцедуры Abort}
begin
    Writeln('Ошибка: ', Msg);
    Halt(1);
end ;
Begin                    {основная программа}
Count:= 0;
Sum: = 0 ;
while (Count < Limit) do   {Условиевыполнениецикла}
          begin
          Count:= Count+1;
```

```
            Write('Введите ', Count, '-е целоечисло: ');
        ReadIn(Item);
                if Item < 0 then {Есливведеноотрицательноечисло}
            Abort('Введено отрицательное число! '); {Вызов процедуры}
            Sum:= Sum+Item;
        end;
    Writeln('Сумма введенных чисел равна ', Sum);
    end.
```

В разделе описания программы описываются константа *Limit*, ограничивающая количество вводимых чисел; в разделе описания переменных описываются переменные *Count, Item, Sum* типа *integer*.

В начале программы обнуляются значения количества введенных чисел *Count* и их сумма *Sum*. Потом выполняется цикл, пока очередное вводимое число меньше предельного, заданного значением константы *Limit*. Сначала устанавливается номер очередного числа, затем на экран выводится приглашение *"Введите 1-е (2-е и т.п.) число"*, считывается значение числа с клавиатуры в переменную *Item*. Затем проверяется условие *Item<0*.

Если условие выполняется, то вызывается процедура *Abort*, которой передается фактический параметр-значение типа string: *"Введено отрицательное число"*. Это значение присваивается формальному параметру *Msg* процедуры *Abort*. Процедура *Abort* выводит на экран сообщение об ошибке и печатает текст сообщения – значение параметра *Msg: "Ошибка: Введено отрицательное число"*, после чего вызывает стандартную процедуру *Halt(1)*, которая прерывает выполнение программы.

Если условие *Item<0* не выполняется, то значение суммы *Sum* увеличивается на значение введенного числа *Item*, и управление передается в заголовок цикла для проверки условия *Count < Limit*. Если условие соблюдается, то тело цикла выполняется еще раз, иначе цикл завершается, а управление в программе передается оператору, следующему за циклом, т. е. за резервированным словом *end*, обозначающим окончание составного оператора в теле цикла. После этого на экран выводится сообщение: *"Сумма введенных чисел равна"* и печатается значение переменной *Sum*. На этом выполнение программы завершается.

Механизм передачи параметров

Как было показано в приведенных выше примерах программ с использованием процедур и функций, в заголовке процедуры или функции может быть задан список параметров, которые называются *формальными.* Название "формальные" эти параметры получили в связи с тем, что в этом списке заданы только имена для обозначения исходных данных и результатов работы процедуры, а при вызове подпрограммы на их место будут подставлены *конкретные значения* (выражений) и имен. Этот список указывается после имени подпрограммы и заключается в круглые скобки.

В списке формальных параметров должны быть перечислены имена формальных параметров и их типы. Имя параметра отделяется от типа двоеточием, а параметры друг от друга - точкой с запятой. Имена параметров одного типа можно объединять в подсписки, в которых имена отделяются друг от друга запятой.

Между формальными и фактическими параметрами должно быть полное соответствие:

– формальных и фактических параметров должно быть одинаковое количество;

– порядок следования фактических и формальных параметров должен быть один и тот же;

– тип каждого фактического параметра должен совпадать с типом соответствующего формального параметра.

Параметры-значения. Параметры-значения используются только для передачи исходных данных из основной программы в подпрограмму (процедуру или функцию), в списке формальных параметров они перечисляются через запятую с обязательным указанием их типов, как было, например, в выше приведенных примерах:

```
procedure Abort(Msg: string);
function Step(N : integer; X : real): real;
```

Если формальный параметр объявлен как параметр-значение, то фактическим параметром может быть произвольное выражение. При вызове подпрограммы фактические параметры вычисляются и используются как начальные значения формальных параметров, т. е. осуществляется подстановка значений. Если формальный параметр определен как параметр-значение, то перед вызовом процедуры это значение вычисляется, полученный результат помещается во временную память и передается процедуре. Даже если фактический параметр – простейшее выражение в виде константы или

переменной, все равно процедуре будет передана лишь копия этой константы (переменной). В процессе выполнения подпрограммы формальные параметры могут изменяться, но это никак не отразится на соответствующих фактических параметрах-переменных, которые сохранят те значения, которые имели до вызова подпрограммы, так как меняются не они, а их копия. Поэтому параметры-значения нельзя использовать для передачи результатов из подпрограммы в основную программу.

Пример программы с использованием передачи параметров по значению:

```
program Pr1;
   var
      A,B : real;
   {Процедура вычисления квадратов двух чисел и вывода на экран их суммы}
   procedure Sum_Square(X, Y : real);  {X,Y - формальныепараметры}
   begin
      X:=X*X;
      Y:=Y*Y;
      Writeln('Сумма квадратов = ',X+Y);
   end;                    {Конец процедуры}
begin{Начало основной программы}
      A:=1.5;
      B:=3.4;
      Sum_Square (A,B) ; {Вызов процедуры Sum_Square с передачей ей
      значений фактических параметров A и B}
end.
```

При вызове процедуры *Sum_Square* с фактическими параметрами *A, B* значения этих параметров (один раз) копируются в соответствующие формальные параметры *X, Y,* и дальнейшие преобразования формальных параметров *X,Y* внутри процедуры *Sum_Square* уже никак не влияют на значения переменных *A, B*.

Параметры-переменные. Параметры-переменные используются для определения результатов выполнения процедуры и в списке формальных параметров перечисляются после зарезервированного слова Var с обязательным указанием типа. Каждому формальному параметру, объявленному как параметр-переменная, должен соответствовать фактический параметр в виде переменной соответствующего типа, например:

```
procedure Example(var M,N : integer; var Y : real) ;
```

Если формальный параметр определен как параметр-переменная, то при вызове процедуры ей передается сама переменная, а не ее копия, и изменение параметра-переменной приводит к изменению фактического параметра в вызывающей программе. Следовательно, исходные данные в процедуру из программы могут передаваться как через параметры-значения, так и через параметры-переменные, а результаты работы процедуры возвращаются в вызывающую программу только через параметры-переменные.

Пример программы, использующей параметры-переменные:

```
program Sum_Sub_Square;
   var
      A,B : real ;
      SumAB, SubAB : real;
      {Процедура с параметрами-переменными Sum, Sub}
      procedure Sum_Sub(X,Y : real; var Sum, Sub : real);
      begin
          Sum:=X*X+Y*Y;
          Sub:=X*X-Y*Y;
      end;                    {Конецпроцедуры}
  begin{Начало основной программы}
      A:=1.5;
      B:=3.4;
      Sum_Sub(A,B,  SumAB,SubAB);     {Вызов процедуры с передачей ей
      фактических параметров-значений A, B и параметров-переменных
      SumAB, SubAB}
      Writeln('Сумма квадратов чисел',A,' и ',B,'= ', SumAB);
      Writeln('Разность квадратов чисел',A,'и',B,'=', SubAB);
  end.
```


Контрольные вопросы

Описание процедуры. Общая структура.

Формальные, фактические параметры.

Параметры-значения, параметры-переменные.

Примеры программ с использованием процедур, определенных пользователем.

Лабораторная работа № 12

Цель работы: формирование знаний и умений по изучению методов внутренних сортировок. Приобретение навыков реализации алгоритмов сортировки.

Краткие теоретические сведения

Под сортировкой понимают процесс переупорядочивания некоторого множества объектов с целью их размещения в заданном порядке. Это универсальный вид деятельности с точки зрения обработки данных, которые представляют собой последовательность *ключей.* С помощью сортировки добиваются такого их размещения, чтобы было выполнено условие:

$$f(a[1]) \leftarrow f(a[2]) \leftarrow f(a[3]) \leftarrow \ldots \leftarrow f(a[N]),$$

где символ $\leftarrow$ означает знак предшествования, а *f*-некоторая функция упорядочивания. При упорядочивании по возрастанию, после сортировки будет выполнено условие:

$$a[1] \le a[2] \le a[3] \le \ldots \le a[N]$$

В ходе сортировки элементы последовательности меняются местами. Сортировка называется *устойчивой*, если на этапе замены два одинаковых ключа не меняются местами. Сортировка называется *внутренней*, если все сортируемые ключи размещаются в оперативной памяти. Если некоторая часть ключей размещается на внешнем носителе, то сортировка называется *внешней*.

Методы внутренней сортировки

В данных лабораторно-практических работах будут рассмотрены методы, у которых на входе - вектор с произвольным положением ключей, а на выходе - этот же вектор упорядочен по *возрастанию*.

Существует три группы методов внутренней сортировки (сортировка включением, сортировка выбором, обменная сортировка). В данной лабораторной работе рассмотрены методы сортировки включением и выбором.

Сортировки включением

Сортировка прямым включением.Элементы условно разделяют на *готовую* a[1],...,a[i-1] и *входную последовательности* a[i],...,a[n]. Сначала i=2. На каждом шаге, увеличивая i на единицу, берут i-й элемент входной

последовательности и передают в готовую последовательность, вставляя на подходящее место.

Итак, в начале рассматривается второй элемент *(i=2)* последовательности ключей. Он сравнивается с первым элементом *(a[i-1])* и если он его меньше, то они меняются местами. В противном случае проход заканчивается, *i* увеличивается на *1* и процесс повторяется. Заметим, что упорядоченная последовательность *a[i-1]* называется *готовой последовательностью* и новый элемент вставляется в готовую последовательность на свое место.

На каждом проходе происходит перемещение *i*-того элемента в готовую последовательность, а некоторое число элементов сдвигается вправо. Данный процесс перемещения называется *просачиванием* элемента.

*Здесь и далее, во всех процедурах сортировки ключи упорядочиваются по возрастанию. На входе процедур – количество элементов массива (**n**), на выходе – упорядоченный массив элементов(**a**).*

```
Procedure Straight_Insertion(n:integer; Var a;array[-400..1000] of integer);
  Var
    i,j:word;
    x:integer;
Begin
  For i:=2 To n Do
  begin
    x:=a[i]; a[0]:=x; j:=i-1;
    While x<a[j] Do
    begin
      a[j+1]:=a[j]; j:=j-1;
    end;
      a[j+1]:=x
  end;
End;{Straight_Insertion}
```

Сортировка бинарными включениями. Алгоритм сортировки прямыми включениями можно легко улучшить, пользуясь тем, что готовая последовательность a[1],...,a[i-1], в которую нужно включить новый элемент, уже упорядочена. Поэтому место включения можно найти значительно быстрее, применив бинарный поиск, который исследует средний элемент готовой последовательности и продолжает деление пополам, пока не будет найдено место включения. Модифицированный алгоритм сортировки называется *сортировкой бинарными включениями.*

```
Procedure Binary_Insertion(n:word;Var a: array[-400..1000] of integer);
  Var
    i,j,l,r,m:word;
    x:integer;
Begin
  For i:=2 To n Do
      begin
        x:=a[i]; l:=1; r:=i-1;
        While l<=r Do
            begin
                m:=(l+r) div 2;
                If x<a[m] Then r:=m-1
                Else l:=m+1
            end;
        For j:=i-1 DownTo l Do a[j+1]:=a[j];
          a[l]:=x
        end;
End;{Binary_Insertion}
```

Сортировка выбором

Прямой выбор. Этот метод основан на следующем правиле: выбираем элемент с наименьшим ключом. Он меняется местами с первым элементом. Эти операции затем повторяются с оставшимися *n*-1 элементами, затем с *n*-2 элементами, пока не останется только один элемент - *наибольший*. Этот метод называемый *сортировкой простым выбором,* в некотором смысле противоположен сортировке простыми включениями; при сортировке простым выбором рассматриваются *все* элементы *входного массива* для нахождения элемента с наименьшим ключом, и этот *один* очередной элемент отправляется в *готовую последовательность*.

```
Procedure Straight_Selection(n:word;Var a array[-400..1000] of integer);
  Var
    i,j,k:word;
    x:integer;
Begin
  For i:=1 To n-1 Do
    begin
      x:=a[i]; k:=i;
      For j:=i+1 To n Do
      If x>a[j] Then
```

```
            begin
               k:=j; x:=a[j];
            end;
            a[k]:=a[i]; a[i]:=x;
         end;
End;{Straight_Selection}
```

Контрольныевопросы

Понятие сортировки. Виды сортировок.

Сортировки включением. Описание алгоритмов методов сортировки прямыми и бинарными включениями.

Сортировки включением. Описание алгоритмов методов сортировки прямыми и бинарными включениями.

Сортировка выбором. Описание алгоритма.

Фрагменты программ для реализации данных методов сортировок.

Лабораторная работа № 13

Цель работы: формирование знаний и умений по работе с множествами. Приобретение навыков обработки множеств.

Краткие теоретические сведения

Основные понятия и определения

Множество – это структурированный тип данных, представляющий собой набор взаимосвязанных по какому-либо признаку или группе признаков объектов, которые можно рассматривать как единое целое. Каждый объект во множестве называется *элементом множества*.

Все элементы множества должны принадлежать одному из скалярных типов, кромевещественного. Этот тип называется *базовым типом множества*. Базовый тип задается диапазоном или перечислением. *Область значений* типа множество – набор всевозможных подмножеств, составленных из элементов базового типа.

В выражениях на языке Паскаль значения элементов множества указываются в квадратных скобках:[1,2,3,4], [‘a’,’b’,’c’], [‘a’..’z’]. Если множество не имеет элементов, оно называется *пустым* и обозначается, как [ ]. Количество элементов множества называется его *мощностью*.

Формат записи множественных типов:

```
Type
```

```
   <имя типа> = setof<элемент 1,…., элементN>;
Var
   <идентификатор, ….> :  <имя типа>;
```

Можно задать множественный тип и без предварительного описания:

```
Var
<идентификатор, …> :  setof<элемент1, …>;
```

Пример:

```
Type
   Simply  = setof  ‘a’ ..’h’;
   Number = setof  1..31;
Var
   Pr :Simply;
   N :Number;
   Letter : setofchar; {определение множества без предварительного описания
   в разделе типов}
```

В данном примере переменная *Pr* может принимать значения символов латинского алфавита от ‘a’ до ‘h’; *N* – любое значение в диапазоне 1..31; *Letter*– любой символ. Попытка присвоить другие значения вызовет программное прерывание.

Количество элементов множества не должно превышать 256, соответственно номера значений базового типа должны находиться в диапазоне 0..255. Контроль диапазонов осуществляется включением директивы {$R+}. Объем памяти, занимаемый одним элементом множества, составляет 1 бит.

Объем памяти для переменной типа множество вычисляется по формуле:

$$Объем\ памяти = (MaxDIV\ 8) – (MinDIV\ 8) + 1,$$

где *Max* и *Min* – верхняя и нижняя границы базового типа.

Операции над множествами

При работе с множествами допускается использование операций отношения “=”, “<>”, “>=”, “<=”, объединения, пересечения, разности множеств и операции in. Результатом выражения с применением этих операций является значение *True* или *False*.

Операция “равно” (=). Два множества A и B считаются равными, если состоят из одних и тех же элементов. Порядок следования элементов в сравниваемых множествах значения не имеет.

Например:

Значение A	*Значение B*	*Выражение*	*Результат*
[1,2,3,4]	[1,2,3,4]	A=B	True
[‘a’, ’b’, ’c’]	[‘c’, ‘a’]	A=B	False
[‘a’..’z’]	[‘z’..’a’]	A=B	True

Операция “не равно” (<>). Два множества A и B считаются не равными, если они отличаются по мощности или по значению хотя бы одного элемента.

Например:

Значение A	*Значение B*	*Выражение*	*Результат*
[1,2,3]	[3,1,2,4]	A<>B	True
[‘a’..’z’]	[‘b’.. ‘z’]	A<>B	True
[‘c’..’t’]	[‘t’..’c’]	A<>B	False

Операция “больше или равно” (>=). Операция “больше или равно” (>=) используется для определения принадлежности множеств. Результат операции A >=B равен True, если все элементы множества B содержаться в множестве A. В противном случае результат равен False.

Например:

Значение A	*Значение B*	*Выражение*	*Результат*
[1,2,3,4]	[2,3,4]	A>=B	True
[‘a’..’z’]	[‘b’.. ‘t’]	A>=B	True
[‘z’,’x’,’c’]	[‘c’,’x’]	A>=B	True

Операция “меньше или равно” (<=). Эта операция используется аналогично предыдущей операции, но результат выражения A<=B равен True, если все элементы множества A содержаться во множестве B. В противном случае результат равен False.

Значение A	*Значение B*	*Выражение*	*Результат*
[1,2,3]	[1,2,3,4]	A<=B	True
[‘d’..’h’]	[‘z’.. ‘a’]	A<=B	True
[‘a’,’v’]	[‘a’,’n’,’v’]	A<=B	True

Операция in. Операция in используется для проверки принадлежности какого-либо значения указанному множеству. Обычно применяется в условных операторах.

Значение A	*Значение B*	*Результат*
2	if A in [1,2,3] then..	True
‘v’	if A in [‘a’..’n’] then..	True
X1	if A in [X0,X1,X2,X3] then..	True

При использовании операции in проверяемое на принадлежность значение и множество в квадратных скобках не обязательно предварительно описывать в разделе описаний. Операция in позволяет эффективно и наглядно производить сложные проверки условий, заменяя иногда десятки других операций. Например, выражение if(a=1) or (a=2) or (a=3) or (a=4) or (a=5) or (a=6) then…можнозаменитьболеекороткимвыражением if a in [1..6] then… .

Часто операцию in пытаются записать с отрицанием: *XNOTinM.* Такая запись является ошибочной, так как две операции следуют подряд; правильная инструкция имеет вид*: NOT (XinM).*

Объединение множеств (+). Объединением двух множеств является третье множество, содержащее элементы обоих множеств.

Например:

Значение A	*Значение B*	*Выражение*	*Результат*
[1,2,3]	[1,4,5]	A+B	[1,2,3,4,5]
[‘A’..’D’]	[‘E’.. ‘Z’]	A+B	[‘A’..’Z’]
[]	[]	A+B	[]

Пересечение множеств (*). Пересечением двух множеств является третье множество, которое содержит элементы, входящие одновременно в оба множества.

Например:

Значение A	*Значение B*	*Выражение*	*Результат*
[1,2,3]	[1,4,2,5]	A*B	[1,2]
[‘A’..’Z’]	[‘B’.. ‘R’]	A*B	[‘B’..’R’]
[]	[]	A*B	[]

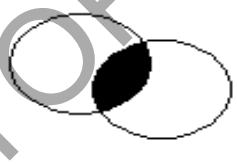
Разность множеств (-). Разностью двух множеств является третье множество, которое содержит элементы первого множества, не входящие во второе множество.

Например:

Значение A	*Значение B*	*Выражение*	*Результат*
[1,2,3,4]	[3,4,1]	A-B	[2]
[‘A’..’Z’]	[‘D’.. ‘Z’]	A-B	[‘A’..’C’]
[X1,X2,X3,X4]]	[X4,X1]	A-B	[X2,X3]

Результат операций рад двумя множествами можно наглядно представить с помощью закрашенных частей двух кружочков:

Объединение



Пересечение

Разность

Использование в программе данных типа **set** дает ряд преимуществ: значительно упрощаются сложные операторы if, увеличивается степень наглядности программы и понимания алгоритма решения задачи, экономятся память, время компиляции и выполнения.

Имеются и отрицательные моменты, основной из них – отсутствие в языке Паскаль средств ввода-вывода элементов множества, поэтому программист сам должен писать соответствующие процедуры.

Иллюстрацией описания и операций над множествами может служить следующий фрагмент программы:

```
ProgramDem_Mno;   {демонстрация операций над множествами}
Type
   Digits=set of 0..9;
```

```
Var
   D1, D2, D3, D:Digits;
Begin
   D1:=[2,4,6,8];      {заполнение множеств}
   D2:=[0..3,5];
   D3:=[1,3,5,7,9];
   D:=D1+D2;     {объединение множеств D1 и D2}
   D:=D+D3;        {объединение множеств D и D3 }
   D:=D-D2;        {разность множеств D и D2 }
   D:=D*D1;        {пересечение множеств D и D1}
end.
```

Как видно из текста программы, сначала описан тип *Digits=setof 0..9,* затем описаны переменные *D1,D2,D3,D* этого типа. В первой части программы осуществляется заполнение множеств, а затем над множествами выполняются операции объединения, пересечения, разности.

Вторым примером работы с множествами может служить следующая задача: описать множество M (1..50) и сделать его пустым. Вводя целые числа с клавиатуры, заполнить множество 10 элементами.

```
ProgramInput_Mno;
Var
    M:set of 1..50;
    X,I:integer;
Begin
    M:= [ ];                {M – пустоемножество}
    for I:=1 to 10 do
    begin
       write('введите ', I, ' –йэлементмножества: ');
       readln (X);
       if (XinM) then    {если введенное число входит в множество M}
            begin
               writeln(X,' помещен в множество 1..50');
               M:=M+[X];
            end;
    end;
    writeln;
end.
```

В разделе описания переменных описано множество целых чисел от 1 до 50, переменная X целого типа, которая используется для считывания числа-кандидата в множество, и целая переменная I, используемая для подсчета количества введенных чисел. В начале программы применена операция инициализации множества M, так как оно не имеет элементов и является пустым:

M:= [ ];

Заполнение множества элементами производится с использованием оператора повтора for, параметр которого I будет указывать порядковый номер вводимого элемента. Операция заполнения множества записывается оператором присваивания:

M:=M+[X];

Контроль заполнения множества записан операцией проверки принадлежности in. Если условие XinM выполняется, выводится сообщение о том, что число X помещено в множество.

Третий пример демонстрирует описание множества гласных и согласных букв русского языка и определяет количество гласных и согласных букв в предложении, введенном с клавиатуры пользователем.

Зададим тип Letters –множество букв русского языка, затем опишем переменные этого типа: Glasn-множество гласных букв, Sogl-множество согласных букв. Вводимое с клавиатуры предложение опишем переменной Text типа String. Для указания символа в строке Text применим переменную I типа Byte.Для подсчета количества гласных и согласных букв опишем переменные Gи S. Проверку принадлежности символов, составляющих предложение множествам гласных или согласных букв русского языка запишем с использованием цикла for , параметр I которого, изменяясь от 1 до значения длины предложения, будет указывать порядковый номер символа в предложении. Принадлежность очередного символа предложения множеству гласных или согласных букв запишем операцией in. Если символ является гласной буквой (Text[I] inGlasn), то счетчик гласных букв G увеличивается на 1. Аналогично с согласными буквами. Текст программы может выглядеть так:

```
ProgramGlasn_Sogl;
Type
   Letters=set of ‘А’..’я’;
```

```
Var
   Glasn, Sogl:Letters;
   Text:String;
   I:Byte;
   G,S:Byte;
Begin
   Glasn:=['А','а', 'Е','е','И','и','О','о','У','у','Э','Ю','ю', 'Я','я'];
   Sogl:=['Б'..'Д','б'..'д','Ж','ж','З','з','К'..'Н','к'..'н','П'..'Т','п'..'т','Ф'..'Щ','
          'ф'..'щ','Ъ','ъ','Ь','ь'];
   Write('Введите предложение');
   Readln(Text);
   G:=0;
   S:=0;
   For I:=1 to Length(Text) do
   Begin
      If Text[I] in Glasn then G:=G+1;
      If Text[I] in Sogl then S:=S+1;
   End;
   Writeln('В предложении" ',Text,' "',G,'гласных и ',S,'согласных букв');
End.
```

Контрольные вопросы

Множества. Основные понятия и определения.

Операции над множествами.

Фрагменты программ с использованием множеств.

4 РАЗДЕЛ КОНТРОЛЯ ЗНАНИЙ

4.1 Задания для контролируемой самостоятельной работы студентов

Самостоятельная работа студентов направлена на совершенствование их умений и навыков по дисциплине «Алгоритмизация и программирование». Цель самостоятельной работы студентов – способствование усвоению в полном объеме учебного материала дисциплины через систематизацию, планирование и контроль собственной деятельности. Преподаватель дает задания по самостоятельной работе и регулярно проверяет их исполнение.

Вариант 1

1. Разработайте алгоритм для вычисления следующей функции:

$$y = 2 \mid 3x+1 \mid - \mid 5 - 2x \mid + 3.$$

Алгоритм должен быть записан без использования функции модуль.

а) Запишите алгоритм на алгоритмическом языке.

б) Представьте алгоритм в виде блок-схемы.

в) Исполните алгоритм для трех точек, принадлежащих различным интервалам числовой прямой.

2. Разработайте алгоритм для вычисления первых *n*слагаемых следующей суммы ряда:

$$S = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots .$$

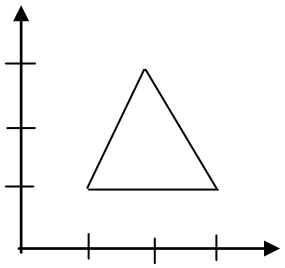
Переменная *x*– вещественная.

а) Запишите алгоритм на алгоритмическом языке.

б) Представьте алгоритм в виде блок-схемы.

в) Исполните алгоритм для $n = 4$.

3. Даны n точек с координатами (x_1; y_1), (x_2; y_2), …, (x_n; y_n). Координаты точек – вещественные числа. Разработайте алгоритм для подсчета количества точек, попавших в фигуру, изображенную на рисунке, включая ее границу.



а) Запишите алгоритм на алгоритмическом языке.

б) Представьте алгоритм в виде блок-схемы.

в) Исполните алгоритм для четырех точек с координатами (1; 1), (2; 2), (3;1), (1;3).

4. Дана матрица a размерности *n*x*m*, элементы которой целые числа. Разработайте алгоритм, который будет заменять в ней положительные элементы на нули.

а) Запишите алгоритм на алгоритмическом языке.

б) Представьте алгоритм в виде блок-схемы.

в) Исполните алгоритм для следующей матрицы:

$$\begin{vmatrix} -2 & 0 & 3 & -4 \\ 1 & -4 & 2 & 0 \\ 3 & 1 & 0 & -6 \end{vmatrix}.$$

Вариант 2

1. Разработайте алгоритм для вычисления следующей функции:

$$y = |\, 4x+2| - 2 \,|\, 3 - x \,| + 5.$$

Алгоритм должен быть записан без использования функции модуль.

а) Запишите алгоритм на алгоритмическом языке.

б) Представьте алгоритм в виде блок-схемы.

в) Исполните алгоритм для трех точек, принадлежащих различным интервалам числовой прямой.

2. Разработайте алгоритм для вычисления первых *n*слагаемых следующей суммы ряда:

$$S = x + \frac{x^3}{3 \cdot 4} + \frac{x^5}{5 \cdot 6} + \frac{x^7}{7 \cdot 8} + \ldots \ .$$

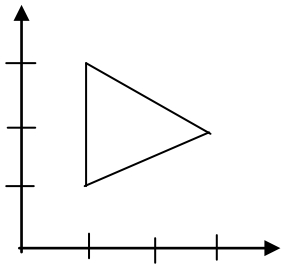
Переменная *x*– вещественная.

а) Запишите алгоритм на алгоритмическом языке.

б) Представьте алгоритм в виде блок-схемы.

в) Исполните алгоритм для $n = 4$.

3. Даны *n* точек с координатами (x_1; y_1), (x_2; y_2), …, (x_n; y_n). Координаты точек – вещественные числа. Разработайте алгоритм для подсчета количества точек, попавших в фигуру, изображенную на рисунке, включая ее границу.



а) Запишите алгоритм на алгоритмическом языке.

б) Представьте алгоритм в виде блок-схемы.

в) Исполните алгоритм для четырех точек с координатами (1; 1), (2; 2), (3;1), (1;3).

4. Дана матрица a размерности *nxm*, элементы которой целые числа. Разработайте алгоритм, который будет заменять в ней отрицательные элементы на нули.

а) Запишите алгоритм на алгоритмическом языке.

б) Представьте алгоритм в виде блок-схемы.

в) Исполните алгоритм для следующей матрицы:

$$\begin{vmatrix} -2 & 0 & 3 & -4 \\ 1 & -4 & 2 & 0 \\ 3 & 1 & 0 & -6 \end{vmatrix}.$$

Вариант 3

1. Разработайте алгоритм для вычисления следующей функции:

$$y = |4x+3| - 3\,|2x - 1| - 3.$$

Алгоритм должен быть записан без использования функции модуль.

а) Запишите алгоритм на алгоритмическом языке.

б) Представьте алгоритм в виде блок-схемы.

в) Исполните алгоритм для трех точек, принадлежащих различным интервалам числовой прямой.

2. Разработайте алгоритм для вычисления первых *n*слагаемых следующей суммы ряда:

$$S = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \frac{x^6}{6!} + \ldots \ .$$

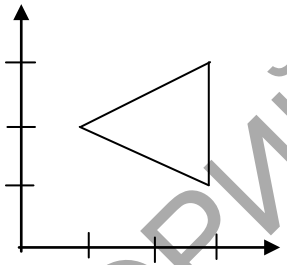
Переменная *x*– вещественная.

а) Запишите алгоритм на алгоритмическом языке.

б) Представьте алгоритм в виде блок-схемы.

в) Исполните алгоритм для $n = 4$.

3. Даны n точек с координатами (x_1; y_1), (x_2; y_2), …, (x_n; y_n). Координаты точек – вещественные числа. Разработайте алгоритм для подсчета количества точек, попавших в фигуру, изображенную на рисунке, включая ее границу.



а) Запишите алгоритм на алгоритмическом языке.

б) Представьте алгоритм в виде блок-схемы.

в) Исполните алгоритм для четырех точек с координатами (1; 1), (2; 2), (3;1), (1;3).

4. Дана матрица a размерности *n*x*m*, элементы которой целые числа. Разработайте алгоритм, который будет заменять в ней нули на число 1.

а) Запишите алгоритм на алгоритмическом языке.

б) Представьте алгоритм в виде блок-схемы.

в) Исполните алгоритм для следующей матрицы:

$$\begin{vmatrix} -2 & 0 & 3 & -4 \\ 1 & -4 & 2 & 0 \\ 3 & 1 & 0 & -6 \end{vmatrix} .$$

Вариант 4

1. Разработайте алгоритм для вычисления следующей функции:

$$y = | 5 - 2x | - 2 | x - 3 | + 2.$$

Алгоритм должен быть записан без использования функции модуль.

а) Запишите алгоритм на алгоритмическом языке.

б) Представьте алгоритм в виде блок-схемы.

в) Исполните алгоритм для трех точек, принадлежащих различным интервалам числовой прямой.

2. Разработайте алгоритм для вычисления первых *n*слагаемых следующей суммы ряда:

$$S = \frac{x^2}{2!} - \frac{x^4}{4!} + \frac{x^6}{6!} - \frac{x^8}{8!} + \ldots .$$

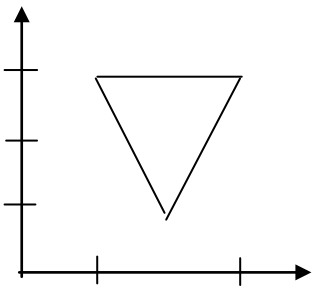
Переменная *x*– вещественная.

а) Запишите алгоритм на алгоритмическом языке.

б) Представьте алгоритм в виде блок-схемы.

в) Исполните алгоритм для $n = 4$.

3. Даны n точек с координатами (x1; y1), (x2; y2), …, (xn; yn). Координаты точек – вещественные числа. Разработайте алгоритм для подсчета количества точек, попавших в фигуру, изображенную на рисунке, включая ее границу.



а) Запишите алгоритм на алгоритмическом языке.

б) Представьте алгоритм в виде блок-схемы.

в) Исполните алгоритм для четырех точек с координатами (1; 1), (2; 2), (3;1), (1;3).

4. Дана матрица а размерности *n*x*m*, элементы которой целые числа. Разработайте алгоритм, который будет заменять в ней элементы, равные единице, на нули.

а) Запишите алгоритм на алгоритмическом языке.

б) Представьте алгоритм в виде блок-схемы.

в) Исполните алгоритм для следующей матрицы:

$$\begin{vmatrix} -2 & 0 & 3 & -4 \\ 1 & -4 & 2 & 0 \\ 3 & 1 & 0 & -6 \end{vmatrix}.$$

4.2 Контрольные вопросы

4.3 Перечень вопросов по темам семинарских занятий

Раздел 1. Основы алгоритмизации.

Тема2.Технологии проектирования алгоритмов.

Вопросы:

1. Нисходящее и восходящее проектирование алгоритмов.
2. Метод пошаговой детализации алгоритма.
3. Структурное программирование.
4. Процедурное программирование.
5. Логическое программирование.
6. Объектно-ориентированное программирование.
7. Прототипное программирование
8. Аспектно-ориентированное программирование
9. Компонентно-ориентированное программирование

Раздел 4. Программирование на языкеPascalABC.

Тема 10. IPOпринцип разработки программ.

Вопросы:

1. Простые операторы.
2. Совместимость типов переменных.
3. Ореpаторы ввода/вывода.
4. Комментарии к программе.
5. Средства отладки программ в системе PascalABC.

Тема 14. Операции над множествами.

Вопросы:

1. Понятие множества. Операции над множествами.
2. Использование множеств для решения задач.
3. Алгоритмы нахождения числа перестановок, числа сочетаний и числа размещений.
4. Описание множеств на языке Паскаль.
5. Операции над множествами на языке Паскаль.
6. Комбинаторные объекты.
7. Операции над комбинаторными объектами.

Тема 15. Средства обработки файлов.

Вопросы:

1. Обработка типизированных файлов.
2. Обработка текстовых файлов.
3. Запись как структурированный тип данных

4. Способы описания записей.
5. Порядок организации связи с файлом.
6. Открытие и закрытие файлов.
7. Чтение информации из файла.
8. Запись информации в файл.

При подготовке к семинарским занятиям рекомендуется:

1. Подготовить краткий конспект по каждому вопросу семинарского занятия.

2. Подготовить интерактивную презентацию по любому из вопросов.

4.4 Перечень вопросов к экзамену

1. Понятие алгоритма. Свойства алгоритма.
2. Типы алгоритмических структур: следование, разветвление и цикл.
3. Основная теорема структурного программирования.
4. Технология нисходящего программирования.
5. Формы записи алгоритмов.
6. Алгоритмический язык.
7. Общий вид алгоритма на алгоритмическом языке.
8. Исполнение алгоритма.
9. Линейные алгоритмы.
10. Алгоритмы с разветвлением.
11. Циклические алгоритмы.
12. Характеристики времени и памяти.
13. Быстрый последовательный поиск.
14. Двоичный поиск.
15. Прямой доступ.
16. Алгоритм проверки расстановки скобок.
17. Сортировка с прямым доступом.
18. Сортировка методом пузырька.
19. Сортировка методом прямого включения.
20. Сортировка методом прямого выбора.
21. Структура программы на языке Паскаль.
22. Форматы вывода данных в процедурах Write и WriteLn.
23. Обработка строковых данных. Типы данных CHAR и STRING.
24. Процедуры и функции для обработки строк.
25. Использование оператора цикла FOR … TO … DO.
26. Использование оператора цикла REPEAT …UNTIL.
27. Использование оператора цикла WHILE … DO.
28. Условный оператор IF. Формат. Пример использования.
29. Условный оператор CASE. Формат. Пример использования.
30. Обработка строк на языке Паскаль.
31. Обработка одномерных массивов. Пример программы.
32. Обработка двумерных массивов. Пример программы.
33. Множества. Основные понятия и определения.
34. Операции над множествами.
35. Структура экрана системы PascalABC.
36. Технология работы в системе PascalABC.
37. Встроенные функции и процедуры. Понятие библиотечного модуля.
38. Структура функции, определенной пользователем. Параметры.

39. Пример программы с определенной пользователем функцией.

40. Структура процедуры, определенной пользователем. Параметры.

41. Пример программы с определенной пользователем процедурой.

42. Механизм передачи параметров. Параметры-значения, параметры-переменные.

4.5 Задачи к экзамену

Задача 1

Дан массив действительных чисел A размерности n×m. Составьте программу на языке Паскаль, которая будет находить максимальный элемент в массиве A, а также номер строки и номер столбца, в котором этот элемент расположен.

Задача 2

Составьте программу на языке Паскаль, которая будет в массиве целых чисел A размерности n переставлять элементы так, чтобы вначале разместились отрицательные элементы, а затем – неотрицательные.

Задача 3

Составьте программу на языке Паскаль, которая будет в массиве действительных чисел A размерности n будет переставлять элементы так, чтобы вначале разместились положительные элементы, а затем – неположительные.

Задача 4

Дан массив действительных чисел A размерности n×m. Составьте программу на языке Паскаль, которая будет находить сумму чисел, находящихся в i–й строке и j–ом столбце массиве A. Элемент Aij должен учитываться один раз.

Задача 5

Составьте программу на языке Паскаль, которая будет находить номер наибольшего элемента в целочисленном массиве A размерности n.

Задача 6

Составьте программу на языке Паскаль, которая будет находить следующую сумму:

$$S = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \ldots + \frac{x^n}{n!}.$$

Задача 7

Дан массив целых чисел A размерности n. Составьте программу на языке Паскаль, которая будет подсчитывать в нем количество положительных элементов.

Задача 8

Составьте программу на языке Паскаль, которая будет подсчитывать количество положительных элементов k в массиве целых чисел A размерности n и записывать положительные элементы в другой массив B размерности k.

Задача 9

Дан массив целых чисел A размерности n. Составьте программу на языке Паскаль, которая будет переставлять в нем местами максимальный и минимальный элементы. Если в массиве есть несколько элементов с максимальным (минимальным) значением, то выбирать любой из них.

Задача 10

Задана матрица целых чисел A размерности (n×n). Составьте программу на языке Паскаль, которая для k-ой строки матрицы найдет сумму её элементов.

Задача 11

Задана матрица целых чисел A размерности (n×n). Составьте программу на языке Паскаль, которая для k-ого столбца матрицы найдет сумму его элементов.

Задача 12

Дан массив целых чисел A размерности n. Составьте программу на языке Паскаль, которая будет находить среднее значение элементов массива A.

Задача 13

На плоскости заданы три точки: A(x1;y1), B(x2;y2) и C(x3;y3). Определите, можно ли построить из них посредством соединения равнобедренный треугольник. Переменной d присвойте значение «да», если можно получить равнобедренный треугольник, в противном случае – «нет». Программу запишите на языке Паскаль.

Задача 14

На плоскости заданы три точки: A(x1;y1), B(x2;y2) и C(x3;y3). Определите, можно ли построить из них посредством соединения

равносторонний треугольник. Переменной d присвойте значение «да», если можно получить равносторонний треугольник, в противном случае – «нет». Программу запишите на языке Паскаль.

Задача 17

Дана строка a. Определите, сколько она содержит слов. Предполагается, что слова в строке разделяются одним пробелом. Программу запишите на языке Паскаль.

Задача 18

Дана строка a. Определите, сколько раз в ней встречается буква «я». Программу запишите на языке Паскаль.

Задача 19

Дана строка a. Предполагается, что слова в строке разделяются одним пробелом. Определите в ней длину k-го слова. Программу запишите на языке Паскаль.

Задача 20

На плоскости заданы три точки: A(x1;y1), B(x2;y2) и C(x3;y3). Определите, какая из них находится дальше от начала координат. Переменной d присвойте имя наиболее удаленной от начала координат точки. Программу запишите на языке Паскаль.

Задача 21

Задана матрица целых чисел A размерности (n×m). Составьте программу на языке Паскаль, которая поменяет местами в ней i-ю и j-ю строки.

Задача 22

Задана матрица целых чисел A размерности (n×n). Составьте программу на языке Паскаль, которая заполнит ее главную диагональ натуральными числами 1, 2, … , n.

Задача 23

Задана матрица целых чисел A размерности (n×n). Составьте программу на языке Паскаль, которая заполнит ее главную диагональ натуральными числами 1, 2, … , n в обратном порядке.

Задача 24

Дан массив натуральных чисел A размерности n. Составьте программу на языке Паскаль, которая будет подсчитывать в нем количество четных чисел

Задача 25

Дана строка a. Определите, сколько раз в ней встречается пробел. Программу запишите на языке Паскаль.

4.6 Критерии оценки результатов учебной деятельности студентов

Для выявления и исключения пробелов в знаниях студентов рекомендуется использовать следующие средства:

1) фронтальный опрос на лекциях, лабораторных и семинарских занятиях;
2) критериально-ориентированные тесты для контроля теоретических знаний современных информационных настольных издательских систем, основных определений, терминологии и правил набора, редактирования, форматирования и верстки текстовой и графической информации;
3) выполнение тестовых заданий с произвольной формой ответа для контроля умения анализировать и грамотно излагать и формулировать свои соображения и выводы в данной предметной области;
4) выполнение творческих заданий, которые предполагают эвристическую деятельность и поиск неформальных решений.

5 ВСПОМОГАТЕЛЬНЫЙ РАЗДЕЛ

5.1 Учебнаяпрограмма

Гляков, П.В. Программно-технические средства.Часть2.Алгоритмизация и программирование: учебнаяпрограмма учреждения высшего образования для специальности 1-21 04 01 Культурология (по напрамлениям), направления специальности 1-21 04 01-02 Культурология (прикладная, специализации 1-21 04 01-02 04 Информационные системы в культуре / П.В. Гляков. – Минск : БГУКИ, 2013. – 16 с.

5.2 Учебно-методическая карта учебной дисциплины для дневной формы получения высшего образования

Номер раздела, темы	Название раздела, темы	Количество аудиторных часов				Количество часов УСР	Форма контроля Знаний
		Лекции	Практические занятия	Семинарские занятия	Лабораторные занятия		
Раздел 1	**Основы алгоритмизации**						
Тема 1	Алгоритм и его свойства	2					
Тема 2	Технологии проектирования алгоритмов			2		2	Реферат
Тема 3	Алгоритмический язык	2			2		
Тема 4	Типы алгоритмов	2			2		
Раздел2	**Сложность алгоритмов**						
Тема 5	Характеристики времени и памяти	2				2	
Тема 6	Методы обработки информации	2			2	2	
Раздел 3	**Система программирования Pascal ABC**						
Тема 7	Структура экрана системы PascalABC			2			

Тема 8	Технология работы в системе PascalABC				2	2	
Раздел 4	**Программирование на языке PascalABC**						
Тема 9	Основы построения программ на языке PascalABC			2		2	
Тема 10	IPO принцип разработки программ			2			
Тема 11	Управляющие конструкции языкаPascalABC	2			2		
Тема 12	Процедуры и функции	2			2	2	
Тема 13	Работа со строковыми типами данных	2		2			
Тема 14	Операции над множествами			2		2	
Тема 15	Построение изображений на экране			2	2	2	
Тема 16	Средства обработки файлов			2		2	
	Всего…	**16**		**16**	**14**	**18**	

5.3 Учебно-методическая карта учебной дисциплины для заочной формы получения высшего образования

Номер раздела, темы	Название раздела, темы	Количество аудиторных часов				Количество часов УСР	Форма контроля Знаний
		Лекции	Практические занятия	Семинарские занятия	Лабораторные занятия		
Раздел 1	**Основы алгоритмизации**						
Тема 1	Алгоритм и его свойства	2					
Тема 2	Технологии проектирования алгоритмов			2		2	Реферат
Тема 3	Алгоритмический язык	2					
Тема 4	Типы алгоритмов					2	
Раздел2	**Сложность алгоритмов**						
Тема 5	Характеристики времени и памяти					2	
Тема 6	Методы обработки информации					2	
Раздел 3	**Система программирования Pascal ABC**						
Тема 7	Структура экрана системы PascalABC		2				
Тема 8	Технология работы в системе PascalABC				2	2	
Раздел 4	**Программирование на языке PascalABC**						
Тема 9	Основы построения программ на языке PascalABC	2			2		
Тема 10	IPO принцип разработки программ					2	
Тема 11	Управляющие конструкции				2		

	языкаPascalABC						
Тема 12	Процедуры и функции					2	
Тема 13	Работа со строковыми типами данных				2		
Тема 14	Операции над множествами			2			
Тема 15	Построение изображений на экране					2	
Тема 16	Средства обработки файлов					2	
	Всего…	**6**		**4**	**8**	**18**	

5.4 Список основной литературы

1. Расолька,Г.А.Паскаль. Тэорыя і практыка праграміравання: вучэб.-метад. дапам. / Г.А. Расолька, Ю.А. Крэмень. – Мінск : БДУ, 2008. – 392 с.

2. Расолька, Г.А.Паскаль. Метады праграміравання. Алгарытмы апрацоўкі даных : вучэб.-метад. дапам. / Г.А. Расолька, Ю.А. Крэмень. – Мінск : БДУ, 2008. – 296 с.

3. Ахо, А. Построение и анализ вычислительных алгоритмов / А. Ахо, Дж. Хопкрофт, Дж. Ульман.– М.: Мир, 1979. – 536 с.

4. Вирт, Н. Алгоритмы и структуры данных /Н. Вирт. – М.: Мир, 1989. – 374 с.

5. Кнут, Д. Искусство программирования для ЭВМ: В 3 т. Т.1: Основные алгоритмы / Д. Кнут. – М. : Мир, 2000. – 884 с.

5.5 Список дополнительной литературы

Дополнительная

6. Шпак, Ю.А. Turbo Pascal 7.0 на примерах / Ю.А. Шпак. – Под редакцией Ю. С. Ковтанюка. – Киев : Издательство "ЮНИОР", 2003. – 496 с.

7. Бондарев, В.М. Основы программирования / В.М. Бондарев[и др.]. – Ростов-на-Дону : Феникс, 1997. – 496 с.

8. Бородич, Ю.С. Паскаль для персональных компьютеров / Ю.С. Бородич, А.Н. Вальвачев, А.И. Кузьмич. – Минск : Выш. шк., 1991. – 384 с.

9. Марченко, А.И. Программирование в среде BorlandPascal 7.0. / А.И. Марченко. – Киев: Век и ЮНИОР, 1996. – 376 с.

10. Фаронов, В.В. Турбо Паскаль 7.0. Начальный курс : учеб.пособие / В.В. Фаронов. – М.: Нолидж, 1997. – 398 с.

11. Федоров, А. BorlandPascal в среде Windows / А. Федоров, Д. Рогаткин. – К.:Издательство ”Диалектика”,1993. – 482 с.