МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ

«БРАТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Кафедра информатики и прикладной математики

УТВЕРЖДАЮ:
Проректор по учебной работе
_____ Е.И. Луковникова
«_____» _____ 20 г.

ПРОГРАММА УЧЕБНОЙ ПРАКТИКИ
(практика по получению первичных профессиональных умений и навыков, в том числе первичных умений и навыков научно-исследовательской деятельности) №2

НАПРАВЛЕНИЕ ПОДГОТОВКИ

09.03.02 ИНФОРМАЦИОННЫЕ СИСТЕМЫ И ТЕХНОЛОГИИ

ПРОФИЛЬ ПОДГОТОВКИ

Информационные системы и технологии

Квалификация (степень) выпускника: бакалавр

СОДЕРЖАНИЕ ПРОГРАММЫ **Стр.**

1. ВИД, ТИП ПРАКТИКИ И СПОСОБЫ ЕЕ ПРОВЕДЕНИЯ

1.1 Вид практики – учебная.

1.2 Тип практики – практика по получению первичных профессиональных умений и навыков, в том числе первичных умений и навыков научно-исследовательской деятельности.

1.3 Способ проведения:

- стационарная;
- выездная.

Для лиц с ограниченными возможностями здоровья выбор мест прохождения практик должен учитывать состояние здоровья и требования по доступности.

2. ПЕРЕЧЕНЬ ПЛАНИРУЕМЫХ РЕЗУЛЬТАТОВ ОБУЧЕНИЯ ПО ПРАКТИКЕ, СООТНЕСЕННЫХ С ПЛАНИРУЕМЫМИ РЕЗУЛЬТАТАМИ ОСВОЕНИЯ ОБРАЗОВАТЕЛЬНОЙ ПРОГРАММЫ

Вид деятельности выпускника

Практика охватывает круг вопросов, относящихся к научно-исследовательской деятельности бакалавра в соответствии с компетенциями и видами деятельности, указанными в учебном плане.

Цель практики

- закрепление и углубление теоретических знаний, полученных при обучении;
- подготовка к осознанному и углубленному изучению общепрофессиональных и специальных дисциплин;
- формирование компетенций, связанных с готовностью к кооперации с коллегами при работе в коллективе; получением навыков организации и управления малыми коллективами; применением различных информационные систем и технологий для решения практических задач в производственной сфере; способностью осуществлять организацию рабочих мест, их техническое оснащение, размещение компьютерного оборудования.

Задачи практики

- получение первичных профессиональных умений и навыков;
- закрепление теоретических знаний, умений и навыков, полученных обучающимися в процессе аудиторных занятий;
- расширение профессионального кругозора обучающихся;
- изучение опыта работы, соответствующей основным видам будущей профессиональной деятельности обучающихся по направлению 09.03.02 Информационные системы и технологии.
- проведение поиска и систематизации научной информации в определенных областях знания с использованием современных информационных технологий;
- выработка способности и умения анализировать и представлять полученные в ходе исследования результаты в виде законченных отчетов.

Код компетенции	Содержание Компетенций	Перечень планируемых результатов обучения по практике
1	2	3
ОК-2	готовность к кооперации с коллегами, работе в коллективе, знание принципов и методов организации и управления малыми коллективами	**знать:** − методы организации и управления малыми коллективами; **уметь:** − работать в коллективе и применять различные методы организации и управления малыми коллективами; **владеть:** − методиками организации и управления малыми коллективами.
ОПК-1	владение широкой общей подготовкой (базовыми знаниями) для решения практических задач в области информационных систем и технологий	**знать:** − теоретические основы общей подготовки для решения практических задач в области информационных систем и технологий; **уметь:** − применять различные информационные системы и технологии для решения практических задач в производственной сфере; **владеть:** − базовыми знаниями для решения практических задач в области информационных систем и технологий.
ОПК-5	способность использовать современные компьютерные технологии поиска информации для решения поставленной задачи, критического анализа этой информации и обоснования принятых идей и подходов к решению	**знать:** − основные информационно- коммуникационные технологии поиска информации; **уметь:** − использовать возможности информационно-вычислительных сетей для решения практических задач; **владеть:** − навыками поиска информации для решения поставленной задачи и обоснования принятых идей и подходов к решению вычислительных задач.
ПК-23	готовность участвовать в постановке и проведении экспериментальных исследований.	**знать:** – методики проведения экспериментальных исследований; **уметь:** – планировать постановку и проведение экспериментальных исследований; **владеть:** – навыками участия в постановке и проведении экспериментальных исследований.

ПК-24	способность обосновывать правильность выбранной модели, сопоставляя результаты экспериментальных данных и полученных решений.	**знать:** – методологию определения целей и задач проведения экспериментальных исследований; **уметь:** – проводить экспериментальные исследования; **владеть:** – современными инструментальными средствами планирования экспериментов и анализа их результатов.

3. МЕСТО ПРАКТИКИ В СТРУКТУРЕ ОБРАЗОВАТЕЛЬНОЙ ПРОГРАММЫ

Учебная (практика по получению первичных профессиональных умений и навыков, в том числе первичных умений и навыков научно-исследовательской деятельности) №2 является обязательной.

Учебная (практика по получению первичных профессиональных умений и навыков, в том числе первичных умений и навыков научно-исследовательской деятельности) №2 базируется на знаниях, полученных при изучении таких учебных дисциплин, как: «История отрасли и введение в специальность», «Информатика», «Информационные технологии», «Технология программирования».

Основываясь на изучении перечисленных дисциплин, Учебная (практика по получению первичных профессиональных умений и навыков, в том числе первичных умений и навыков научно-исследовательской деятельности) №2 представляет основу для изучения дисциплин: «Технологии обработки информации», «Теория информационных процессов и систем», для прохождения производственной и преддипломной практики, а также – основу для подготовки к государственной итоговой аттестации.

Такое системное междисциплинарное изучение направлено на достижение требуемого ФГОС ВО уровня подготовки по квалификации «бакалавр».

4. ОБЪЕМ ПРАКТИКИ, ЕЕ ПРОДОЛЖИТЕЛЬНОСТЬ

Объем практики: 3 зачетных единиц.
Продолжительность: 2 недели/ 108 академических часов

4.1. Распределение объема практики по видам учебных занятий и трудоемкости

- *обязательное наличие часов по контактной работе обучающихся с преподавателем.*

Вид учебных занятий	Трудоемкость (час.)
1	2
I. Контактная работа обучающихся с преподавателем (всего)	**36**
Лекции (Лк)	2
Практические занятия (ПЗ)	34
II.Самостоятельная работа обучающихся (СР)	**66**
Подготовка к практическим занятиям	40
Подготовка к зачету с оценкой	10
Подготовка и формирование отчета по практике	10
III. Промежуточная аттестация: зачет с оценкой	**6**

5. СОДЕРЖАНИЕ ПРАКТИКИ

№ раз-дела и темы	*Наименование раздела (этапа) практики*	*Трудо-ем-кость, (час.)*	*Виды учебных занятий, включая самостоятельную работу обучающихся и трудоемкость; (час.)*		
			учебные занятия		*самостоятельная работа обучающихся**
			лекции (ввдные)	*практические занятия*	
1	2	3	4	5	6
1.	**Подготовительный этап**	**4**	**2**	**-**	**2**
1.1.	Инструктаж по технике безопасности	2	2	-	-
1.2.	Ознакомление с рабочей программой по практике	2	2	-	-
2.	**Исследовательский этап**	**80**		**34**	**46**
2.1.	Обработка файловых структур данных	20	-	10	16
2.2.	Разработка программ модульной структуры	25	-	12	15
2.3.	Динамическая память	25	-	12	15
3.	**Обработка и анализ полученной информации (материала)**	**14**	**-**	**-**	**14**
4	**Подготовка отчета по практике**	**10**	**-**	**-**	**10**
	ИТОГО	**108**	**2**	**34**	**72**

6. ФОРМЫ ОТЧЕТНОСТИ ПО ПРАКТИКЕ

6.1. Дневник практики

Дневник является обязательной формой отчетности и заполняется практикантом непосредственно во время прохождения практики.

На титульном листе дневника указывается:

- Ф.И.О. , учебная группа обучающегося: ИСиТ-…..;
- код и наименование направления подготовки: 09.03.02 Информационные системы и технологии;
- направленность: профиль Информационные системы и технологии;
- место проведения практики: ФГБОУ ВО «БрГУ»;
- период практики: 4 семестр 2 курса, 22-23 недели;
- Ф.И.О. руководителя практики от университета.

Содержательная часть дневника включает краткие сведения о выполняемой работе по конкретным датам с указанием объема времени (в часах), затраченного на выполнение конкретного вида работы.

Итогом заполнения дневника является заключение руководителя практики от университета.

6.2. Отчет по практике

6.2.1. Требования к отчету по практике

На протяжении всего периода прохождения практики в соответствии с индивидуальным заданием, практикант знакомится с информацией, документами, собирает, обобщает и обрабатывает необходимый материал в соответствии с заданием руководителя, а затем представ-

ляет его в виде письменного отчета по практике (Отчет).

Содержание отчета по практике определяется руководителем практики от университета кафедры информатики и прикладной математики, с учетом общих требований к прохождению практики и индивидуального задания практиканта.

Структурными элементами Отчета являются:

- титульный лист;
- задание на практику;
- содержание;
- введение;
- основная часть;
- заключение;
- список использованных источников;
- приложения (при необходимости).

На титульном листе Отчета указывается:

- полное название факультета: Естественнонаучный факультет и кафедры: кафедра информатики и прикладной математики;
- Ф.И.О., учебная группа обучающегося: ИСиТ-….;
- Ф.И.О. руководителя практики от университета с указанием ученой степени, ученого звания.

В содержании указываются все разделы Отчета с указанием страниц.

Во введении необходимо сформулировать и описать цели и задачи практики.

В состав основной части входят следующие разделы:

- сбор и систематизация информации по теме исследования;
- методика и результаты исследования;
- обзор информации по теме исследования, с учетом задания руководителя.

В заключении излагаются основные результаты прохождения практики, оценивается успешность решения поставленных задач и степень достижения цели.

Список использованных источников должен включать в себя основную и дополнительную литературу, нормативные документы, специальную литературу, методические и рекомендательные материалы, использованные при подготовке и написании отчета и состоять не менее чем из 10 позиций.

Приложения размещают в Отчет при необходимости.

В качестве приложений могут быть представлены различные нормативные документы, схемы, рисунки, карты, таблицы и т.д.

Отчет должен быть выполнен аккуратно, без исправлений. Объем отчета должен составлять 20 – 30 страниц.

Выдача задания, прием и защита отчета по практике проводится в соответствии с календарным учебным графиком.

6.2.2. Примерная тематика индивидуальных заданий

1. Алгоритмизация и программирование файловых структур данных, вывода данных в файл, чтения данных из файла, обработки файловых данных в соответствии с заданным вариантом (индивидуальное задание у руководителя практики).

2. Разработка программ с использованием подпрограмм-функций и подпрограмм-процедур, обращения к подпрограммам, выбора параметров подпрограмм в соответствии с заданием (индивидуальное задание у руководителя практики). Описание типов данных и подпрограмм разместить в отдельном модуле, который подключается к основной программе.

3. Работа с динамической памятью для решения математических задач, имеющих алгоритмический характер в соответствии с заданием (индивидуальное задание у руководителя практики).

7. ПЕРЕЧЕНЬ УЧЕБНОЙ ЛИТЕРАТУРЫ И РЕСУРСОВ СЕТИ ИНТЕРНЕТ, НЕОБХОДИМЫХ ДЛЯ ПРОВЕДЕНИЯ ПРАКТИКИ

№	*Наименование издания*	*Количество экземпляров в библиотеке*, шт.	*Обеспеченность (экз./чел.)*
1	2	4	5
1.	Зеленяк, О. П. Практикум программирования на Turbo Pascal. Задачи, алгоритмы и решения : учебное пособие / О. П. Зеленяк. - 3-е изд., перераб. и доп. - Санкт-Петербург : ДиаСофтЮП ; Москва : ДМК-Пресс, 2007. - 320 с.	21	1
2.	Горохов, Д. Б. Программирование на языке Pascal : методические указания к выполнению лабораторных работ / Д. Б. Горохов. - Братск : БрГУ, 2017. - 144 с.	11	1
3.	Информатика. Базовый курс : учебник для бакалавров и специалистов / Под ред. С. В. Симоновича. - 3-е изд. - Санкт-Петербург : Питер, 2014. - 640 с.	76	1
4.	Незнанов А.А. Программирование и алгоритмизация : учебник / А. А. Незнанов. - М. : Академия, 2010. - 304 с.	10	0,5
5.	Новожилов,О.П. Информатика : учебное пособие / О. П. Новожилов. - 2-е изд., испр. и доп. - М. : Юрайт, 2012. - 564 с	16	0,8
6.	Платонов Ю.М. Информатика: учебное пособие / Ю.М. Платонов, Ю.Г. Уткин, М.И. Иванов; Министерство транспорта Российской Федерации, Московская государственная академия водного транспорта.-Москва: Альтаир: МГАВТ, 2014.- 226 с.: табл., схем., ил.; То же [Электронный ресурс].- URL: http://biblioclub.ru/index.php?page =book&id=429784	ЭР	1
7.	Колокольникова А.И. Информатика:учебное пособие / А.И. Колокольникова, Е.В. Прокопенко, Л.С. Таганов.- Москва: Директ-Медиа, 2013.-115 с.- Библиограф. в кн.- ISBN 978-5-4458-2864-8; То же [Электронный ресурс].- URL: http://biblioclub.ru/index.php?page =book&id=210626	ЭР	1
8.	Zotero – обработка библиографической информации: учебное пособие / Е.Ю. Шахова, Л.В. Васильева, А.Н. Ефремова. – Братск: ФГБОУ ВПО «БрГУ», 2014. – 160 с.	43	1
9.	Технология программирования / Ю.Ю. Громов, О.Г. Иванова, М.П. Беляев, Ю.В. Минин ; Министерство образования и науки Российской Федерации, Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Тамбовский государственный технический университет». - Тамбов : Издательство ФГБОУ ВПО «ТГТУ», 2013. - 173 с. : ил. - Библиогр. в кн. - ISBN 978-5-8265-1207-4 ; То же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=277802 (29.09.2017).	ЭР	1

8. ПЕРЕЧЕНЬ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, ИСПОЛЬЗУЕМЫХ ПРИ ПРОВЕДЕНИИ ПРАКТИКИ, ВКЛЮЧАЯ ПЕРЕЧЕНЬ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ И ИНФОРМАЦИОННЫХ СПРАВОЧНЫХ СИСТЕМ

- ОС Windows 7 Professional.
- Microsoft Office 2007 Russian Academic OPEN No Level.
- Антивирусное программное обеспечение Kaspersky Security.
- Pascal ABC.

9. ОПИСАНИЕ МАТЕРИАЛЬНО-ТЕХНИЧЕСКОЙ БАЗЫ, НЕОБХОДИМОЙ ДЛЯ ПРОВЕДЕНИЯ ПРАКТИКИ

9.1. Описание материально-технической базы

Вид занятия	*Наименование аудитории*	*Перечень основного оборудования*
1	2	3
ЛК	Дисплейный класс	Учебная мебель 14-ПК: Процессор AMD A6-6400 APU; RAM 4 Gb; HDD 500 Gb. Монитор TFT 19 LG1953S-SF; Принтер: HP LaserJet P3005.
ПЗ	Дисплейный класс /помещение для самостоятельной работы	14-ПК: Процессор AMD A6-6400 APU; RAM 4 Gb; HDD 500 Gb Монитор TFT 19 LG1953S-SF; Принтер: HP LaserJet P3005.
СР	Читальный зал №1	10 ПК i5-2500/H67/4Gb (монитор TFT19 Samsung);принтер HP LaserJet P2055D

9.2. Перечень баз практики

Практика проводится, как правило, на выпускающей кафедре информатики и прикладной математики, осуществляющей подготовку бакалавров по профилю «Информационные системы и технологии».

Обучающиеся могут быть направлены для прохождения практики на базе профильных предприятий (организаций, учреждений) на основании договоров с этими предприятиями (организациями, учреждениями): ООО «Центр облачных технологий» (г. Братск); ООО «ГЛОБАЛ Пойнт» (г. Санкт-Петербург).

10. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ОБУЧАЮЩИХСЯ ПО ВЫПОЛНЕНИЮ ПРАКТИЧЕСКИХ ЗАДАНИЙ

Задание 1. Обработка файловых структур данных.
Порядок выполнения:

1. Ознакомиться с теоретическим материалом.
2. Записать алгоритм в виде блок-схемы.
3. Составить и отладить программу обработки файловых структур данных, записать данные в файл, считать данные из файла, обработать файловые данные и записи.

Задание 2. Составление программ модульной структуры.
Порядок выполнения:

1. Ознакомиться с теоретическим материалом.
2. Записать алгоритм в виде блок-схемы.
3. Разработать и отладить программу модульной структуры. Описание типов данных и подпрограмм разместить в отдельном модуле, который подключается к основной программе.

Задание 3. Размещение данных в динамической памяти.

1. Изучить особенности размещения данных в динамической памяти и приемы работы со связанными динамическими структурами (списками, деревьями и т.п.)
2. Записать алгоритм в виде блок-схемы.
3. Составить и отладить программу с использованием динамической памяти

Форма отчетности: отчет по практике.

Задания для самостоятельной (индивидуальной) работы:

1. Проработать теоретический материал по заданной теме;
2. Выполнить задания в соответствии с выбранным вариантом;

Рекомендации по выполнению заданий

Практическая работа №1

Обработка файловых структур данных

Цель работы: овладение навыками алгоритмизации и программирования файловых структур данных

Типы файлов. Внутренняя структура файлов

Для сохранения информации в языке ТР существует возможность определять файловый тип.

Любой файл имеет три характерные особенности.

1. У него есть имя – программа может работать одновременно с несколькими файлами.
2. Он содержит компоненты одного типа (любой тип, кроме файлового типа).
3. Длина вновь создаваемого файла никак не оговаривается и ограничивается только емкостью устройств внешней памяти.

В зависимости от способа объявления выделяются три вида файлов:

1. Типизированные (file of < тип >);
2. Текстовые (text);
3. Нетипизированные (file).

С каждой переменной файлового типа связано понятие текущего указателя файла. Текущий указатель можно принимать как скрытую переменную (неявно описанную вместе с файловой переменной), которая обозначает (указывает на) конкретный элемент файла.

Пример: описание переменной вида

```
Var f: file of integer;
```

понимается как список любого количества целых чисел, расположенных на некотором ВЗУ (например, МД).

Как правило, все действия с файлом (чтение, запись) производятся поэлементно, причем в этих действиях участвует элемент файла, который обозначается текущим указателем файла.

Общая технологическая последовательность действий с любым файлом:

1. Описать файловую переменную.
2. Связать файловую переменную с файлом на диске (Assign).
3. Открыть файл (Append, Reset, Rewrite).
4. Читать из файла или записывать в файл (Read, Write).
5. Закрыть файл (Close).

Внутренняя структура типизированного файла

Типизированные файлы – совокупность компонентов одинаковой длины и структуры. В конце файла размещен маркер - Eof. Нумерация компонентов с 0. Доступ к информации прямой и последовательный.

Графическая интерпретация внутренней структуры типизированного файла:

Комп.0	Комп.1	Комп.2	……	Комп.N	EOF

Внутренняя структура текстового файла

Текстовые файлы – это совокупность строк разной длины. В конце каждой строки размещается специальный признак (маркер) – Eoln (конец строки), а в конце файла – Eof (конец файла). Доступ к строкам последовательный.

Графическая интерпретация внутренней структуры текстового файла:

строка	EoLn	строка	EoLn	…	строка	EOF

Маркеры EoLn размещаются при записи данных в файл соответствующими операторами (WriteLn).

Внутренняя структура нетипизированного файла

Нетипизированные файлы – совокупность компонентов одинаковой длины, но в отличие от типизированных файлов произвольной структуры. Совместимы с любыми другими файлами. Возможен как прямой, так и последовательный доступ к данным.

Графическая интерпретация внутренней структуры нетипизированного файла:

Комп.0	Комп.1	Комп.2	……	Комп.N	EOF

Установочные и завершающие операции

В эту группу входят четыре операции, реализованные в виде стандартных процедур: Assign, Reset, Rewrite, Flush, Close.

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

<ф.п.> - файловая переменная (идентификатор);

<имя файла или л.у.> - текстовое выражение, содержащее полное имя файла или логическое устройство.

Пример:

Assign (f, ‘d: \mydir\myfile.dat’);

Процедура предназначена для установки связи между конкретным физическим файлом на магнитном носителе и переменной файлового типа, которая будет являться представителем этого файла в программе.

Reset (<ф.п.>); *Rewrite (<ф.п.>);*

<ф.п.>- файловая переменная, связанная ранее процедурой Assign с уже существующим файлом или логическим устройством. Процедуры предназначены для открытия файлов. Под открытием понимается поиск файла на внешнем носителе, образование специальных системных буферов для обмена с ним и установка текущего указателя файла на его начало (т.е. на нулевой элемент).

Разница между этими двумя процедурами заключается в начальных действиях с файлами. Процедура Reset предполагает, что инициируемый файл уже существует, в противном случае возникает ошибка. Процедура Rewrite допускает, что открываемый файл может еще не существовать; в этом случае она создает заданный файл. Если же файл существует, то Rewrite очищает его.

Flush (<ф.п.>);

Используется для завершения обменов с файлами без его закрытия. Очищает внутренний буфер файла и, таким образом, гарантирует сохранность всех последних изменений файла на диске. Процедура игнорируется, если файл был инициирован для чтения процедурой Reset. Эта процедура имеет смысл, если с файлом проводились операции записи, и используется редко, т. к. процедура Close производит такие же действия.

Close (< ф.п. >);

Завершает действия с файлом, который указывается в качестве параметра. При этом ликвидируются внутренние буфера, образованные при открытии этого файла. После этого файловую переменную можно связать посредством процедуры Assign с каким-либо другим дисковым файлом

Операции ввода-вывода

В эту группу входят две операции (процедуры), которые собственно, и реализуют действия по чтению информации из файла и записи информации в файл.

Read (< ф.п.>, < список ввода >);

Данная процедура предназначена для чтения значений из файла в оперативную память.

<ф.п.> - имя файловой переменной, к которой была применена одна из операций открытия (Reset или Rewrite);

<список ввода> - список переменных, в которые будут помещены читаемые из файла значения. Тип этих переменных должен совпадать с типом компонентов файла.

Выполнение процедуры Read происходит следующим образом. Начиная с текущей позиции указателя файла будут последовательно читаться значения, содержащиеся в файле. Каждое прочитанное значение будет присваиваться очередной переменной из списка ввода. После каждого акта чтения указатель файла смещается на следующую позицию.

Если в процессе выполнения процедуры Read текущий указатель файла будет установлен на позицию, не содержащую информации, т. е. будет достигнут конец файла, то чтение будет прекращено, процедура Read завершается и возникает ситуация «конец файла». Возникновение этой ситуации можно проверить с помощью внутренней функции (Eof).

Write (< ф.п. >, < список вывода >);

Данная процедура позволяет записывать в файл информацию из программы. Значение очередного выражения из списка вывода помещается в файл в место, отмеченное текущим указателем. После этого текущий указатель будет продвинут на одну позицию и действия повторятся для следующего выражения из списка вывода.

Eof (< ф.п. >);

Логическая функция (boolean), тестирующая конец файла. Возвращает True, если файловый указатель стоит в конце файла. При записи это означает, что очередной компонент будет добавлен в конец файла, при чтении – файл исчерпан.

Специальные операции

К ним относятся: Erase, Rename, Chdir, Mkdir, Rmdir, Getdir, IOResult (описаны в стандартном модуле Dos).

Данная группа операций предназначена для действий с элементами файловой системы – каталогами и файлами.

Rename (< ф.п. >, < новое имя >);

<новое имя> - строковое выражение. Переименовывает файл. Перед выполнением процедуры необходимо закрыть файл процедурой Close.

Erase (< ф.п. >)

Уничтожает файл. Перед выполнением процедуры файл необходимо закрыть.

Getdir (< устройство >, < каталог >);

<устройство> - выражение типа Word, содержащее номер устройства (0 – устройство по умолчанию, 1 – диск A, 2 – диск B и т. д.);

<каталог> - переменная типа String, в которой возвращается путь к текущему каталогу на указанном диске.

Данная процедура позволяет определить имя текущего каталога.

Mkdir (< каталог >);

Создает новый каталог. < каталог > - выражение типа String, задающее путь к каталогу. Последним именем пути указывается имя создаваемого каталога.

Rmdir (< каталог >);

Удаляет каталог. Удаляемый каталог должен быть пустым.

IOResult

Функция возвращает условный признак (тип результата Word) завершения последней операции ввода-вывода.

Если операция завершилась успешно, функция возвращает 0. Однако данная функция становится доступной только при отключенном автоконтроле ошибок ввода-вывода. Директива компилятора {$I -} отключает, а директива {$I + } включает автоконтроль.

Если автоконтроль включен, то при обнаружении ошибки выполнение программы прекращается, а на экран выводится краткое сообщение, содержащее условный номер ошибки.

Если автоконтроль отключен, то при обнаружении ошибки все последующие обращения к вводу-выводу блокируются, пока не будет вызвана функция IOResult.

Пример. В данном фрагменте программы предусматривается редактирование файла, имя которого содержит переменная Name. Перед редактированием требуется убедиться, что этот файл существует и переименовать его - заменить расширение на bak (резервная копия). Если файл с таким расширением существует, то его надо удалить.

```
…
Var
  Fi: text;                {Исходный файл}
  Fo: text;                {Отредактированный файл}
  Name: string;
  Name_bak: string;
  K: word;
Const
  Bak='.bak';
Begin
  ….
{ формируется имя файла с расширением bak в переменной name_bak}
k:=Pos('.', name);
if k=0 then k:= length(name)+1;
name_bak:= copy(mame, 1, k-1)+Bak;
{Проверяется существование исходного файла}
Assign(fi, name);
{$ I-}              {Директива компилятора отключает автоконтроль ошибок ввода/ вывода}
Reset(fi);
{если файл не существует, то программа завершает работу и управление передается системе}
If  IOResult <> 0 then Halt;
Close(fi);          {исходный файл существует и мы его закрываем}
{Проверяется существование резервного файла}
Assign(fo, name_bak);
Reset(fo);
{$ I+}              {Директива компилятора включает автоконтроль}
{если файл существует, то он удаляется}
If IOResult = 0 then
  Begin
    Close(fo);
    Erase(fo);
  end;
Rename(fi, name_bak);        {переименование исходного файла}
Reset(fi);
Assign(fo, name);
Rewrite(fo);
….
```

Проверка на существование *.bak файла в данном примере необходима, т.к. обращение к процедуре Rename вызовет ошибку, если этот файл существует.

Типизированные файлы

Длина любого компонента типизированного файла строго постоянна, что дает возможность организовывать прямой доступ к каждому из них по его порядковому номеру.

Операции перемещения по файлу

Seek (< ф.п. >, < № компонента >) – смещает текущий указатель файла к требуемому компоненту.

Truncate (< ф.п. >) – отсечение от файла его хвостовой части, начинающейся от текущей позиции указателя включительно.

Filesize (< ф.п. >) – функция возвращает общее число компонентов в файле.

Filepos (< ф.п. >) – функция возвращает номер компонента, на который установлен текущий указатель файла.

Пример. Записать корни из чисел от 1 до 5 в файл. Затем эти значения считать из файла и распечатать.

```
Program ops;
 Var
   R: real; i: integer;
```

```
  Bul: file of real;            {типизированный файл}
 Begin
   Assign (bul, 'c:\tp\bul.dat');
   Rewrite (bul);            {открытие файла, создание нового}
   For I: = 1 to 5 do
    Begin
      R: = sqrt (i);
      Write (bul, r);          {запись числа в файл}
    End;
   Close (bul);               {закрытие файла}
   Reset (bul);               {открытие существующего файла}
   For I: = 1 to 5 do
     Begin
       Read (bul, r);         {чтение одного числа из файла}
       Writeln (r : 6 : 2);   {число выводится на экран}
     End;
End.
```

Добавление данных в файл

Чтобы добавить информацию в файл, необходимо его открыть, переместить текущий указатель файла в конец и только после этого осуществить запись в файл необходимых данных. В завершении файл необходимо закрыть. Следующий фрагмент программного кода иллюстрирует перечисленные действия.

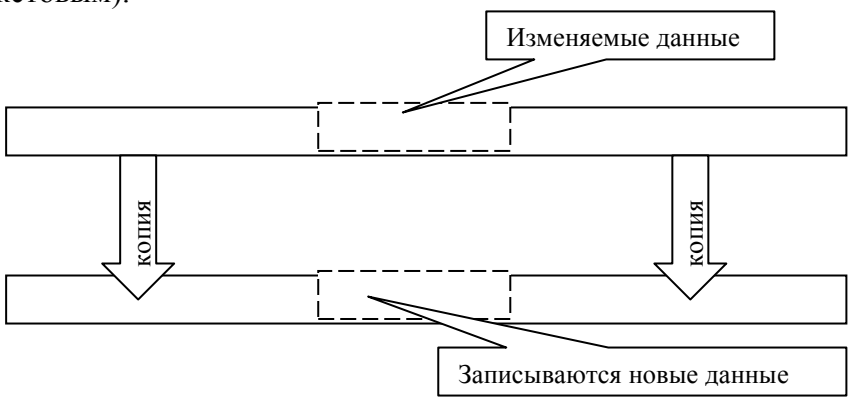
```
.....
assign (f, ''c:\tp\bul.dat');
reset (f);                    {открыть существующий файл}
seek (f, filesize (f));       {указатель в конце файла}
write ('введите число r : ');
readln (r);                   {ввод с клавиатуры одного числа}
write (f, r);                 {запись  числа в файл}
close (f);                    {файл закрывается}
.....
```

Редактирование файла

Для изменения каких-либо данных в файле, необходимо открыть два файла: файл, подлежащий изменению, и новый файл, в котором создается обновленная версия. Часть данных, размещенная до изменяемого фрагмента, копируется в новый файл (рис. 3.9).

Затем в новый файл записываются новые (измененные) данные. Остается скопировать в новый файл данные, размещенные после изменяемого фрагмента. После этого старый файл удаляется, а новый переименовывается – ему присваивается имя старого файла.

Замечание. Данный способ редактирования файла применим как к файлам прямого доступа (типизированным и нетипизированным), так и к файлам последовательного доступа (текстовым).



Пример. Создать файл Sotr.dat, содержащий сведения о N сотрудниках (фамилия, должность, оклад). В этом файле необходимо изменить должность у сотрудника, фамилия которого вводится с клавиатуры. Все данные из нового файла выводятся на экран.

```
Program Redakt;
```

```
Const  N=20;
Type
  Zap= record
     Fam: string [25];
     Dol: string [15];
     Sum: real;
  End;
Var
   f, z: file of zap;                {новый и старый файлы}
   fam_N: string[25];                {запрашиваемая фамилия сотрудника}
   dol_N: string [15];               {новая должность сотрудника}
   b: zap;
   k: integer;
begin
  clrscr;
  assign(z, 'sotr.dat');             {связали файловую переменную}
  Rewrite(z);                        {создаем файл}
  For k:=1 to N do
    Begin                            {ввод данных о сотруднике с клавиатуры}
      Write('введите фамилию'); Readln(b.fam);
      Write('введите должность'); Readln(b.dol);
      Write('введите оклад'); Readln(b.sum);
      Write(z, b);                   {данные о сотруднике записываются в файл)}
    End;
Close (z);
{Редактирование файла}
assign(a, 'sotr_N.dat');             {связывание файловой переменной}
Rewrite(f);                          {создание  нового файла}
Reset(z);                            {открытие старого файла для чтения}
Write('введите фамилию сотрудника'); Readln(fam_N);
Write('введите его новую должность'); Readln(dol_N);
{считываются данные из старого файла и переписываются в новый файл}
While Not Eof(z) do                  {пока не достигнут конец файла}
  Begin
     Read(z, b);                     {считывается запись из старого файла}
     If b.fam=fam_N then
       b.dol:= dol_N;                {меняются данные  у искомого сотрудника}
     Write(f, b);                    {данные записываются в новый файл}
 End;
Close(z);  close(f);                 { файлы закрываются}
Erase(z);                            { старый файл удаляется}
Rename(f, 'sotr.dat');               {переименование нового файла}
End.
```

Текстовые файлы

Начальные и завершающие операции

Assign(<ф.п.>) - связывание файловой переменной с физическим файлом на диске.

Reset (< ф.п. >) – для чтения из файла.

Rewrite (< ф.п. >) – для записи в файл (полное обновление).

Append (< ф.п. >) – открытие файла для добавления новых строк в конец существующего файла.

Close(< ф.п. >) – закрыть файл.

Settextbuf (var f : text; var buf) – процедура, определяющая буфер для обмена с текстовым файлом. Буфер располагается в самой программе. Процедура выполняется до открытия файла.

Операции ввода-вывода

При записи информации в текстовый файл:

Write (< ф.п. >, < список вывода >);

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

Writeln (< ф.п. >);

<список вывода> - список переменных (1 или несколько), значения которых будут записаны в файл.

Writeln в отличие от write добавляет в файл маркер Eoln, т. е. переходит к следующей строке текстового файла.

<список вывода> может содержать переменные следующих типов:

char [: m]
string [: m]
integer [: m]
real [: m : n]
boolean [: m]

m и n – необязательные квалификаторы (целого типа); они определяют каким образом значение переменной будет записано в файл.

Для char: выводится символ, определяемый переменной; если используется квалификатор m > 1, то перед символом выводится (m - 1) пробел.

Пример:

```
Var a: char;
…………………..
a: = 'f ';
write (TF, a);        {в файл запишется символ 'f '}
write (TF, a : 3);    {в файл запишется '␣␣f '}
```

Для string: выводится определяемая переменной; если используется квалификатор m > L (где L – длина строки), то перед символьной строкой будет выведено (m - L) пробелов.

Пример:

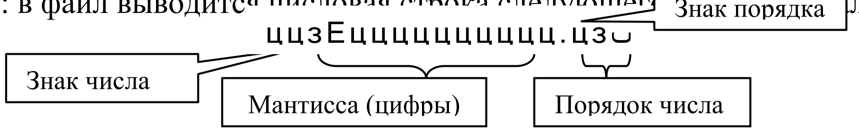
```
Var b: string
………………….
b: = 'крот'
Write (TF, b);
Write (TF, b : 4);
Write (TF, b : 8);           {в файл запишется '␣␣␣␣крот'}
```

Для integer: выводится числовая строка, соответствующая значению переменной (перевод числа в числовую строку - автоматически), если используется квалификатор m > L (L – длина числовой строки), то перед строкой будет выведено (m - L) пробелов.

Пример:

```
Var
 C: integer;
…………………..
c: = 45;
write (TF, c);
write (TF, c : 5);           {в файл запишется '␣␣␣45'}
```

Для real: в файл выводится числовая строка следующег… лон):



Если квалификатор m > 18, то числу предшествует (m - 18) пробелов.

Если квалификатор m = 18 – эталонная строка.

Если квалификатор m < 18 – выводится m символов, причем сначала отбрасывается начальный пробел, затем округляется мантисса (минимальное количество цифр в мантиссе равно 2).

Если квалификатор типа : m : n, то число выводится в естественной форме с n дробными цифрами.

Пример:

```
Var
   X, y: real;
  …………………
  X: = 23.5;  y: = 23. 456789;
```

```
Write (TF, x);
Write (TF, y : 10);
Write (TF, y : 6 : 2);
Write (TF, y : 6 : 0);
```

Для boolean: в файл выводится последовательность в виде true или false, определяемая значением переменной. Если квалификатор m > L (L – длина выражения), то сначала выводится (m - L) пробелов.

При считывании информации из текстового файла:

Read (< ф.п. >, < список ввода >);

Readln (< ф.п. >, < список ввода >);

Readln (< ф.п. >);

<список ввода> - список переменных (1 или несколько), которым будет присвоена информация, считанная из файла.

Readln в отличие от read после считывания списка осуществляет переход к началу следующей строки файла (оставшаяся часть строки пропускается).

Пустой оператор readln пропускает текущую строку в файле.

Пример:

```
Readln (f, x1, x2, x3);
Readln (f, x4, x5);
Read (f);
```

Пример. Создать текстовый файл grup.dat, содержащий данные о N студентах (фамилия, год рождения, специальность). Вывести на экран фамилии студентов с заданным годом рождения.

```
Program File_Text;
Uses Crt;
Const N=25;
Var
 fp: Text;                              {файловая переменная}
 Fam: string[20];                       {фамилия}
 God: word;                             {год рождения}
 Spec: string[10];                      {специальность}
 K: integer;
 God_Is: word;                          {искомый год}
Begin
 Clrscr;
 Assign(fp, 'c:\grup.dat');
 Rewrite(fp);                           {создается файл}
 For k:=1 to N do
  Begin
   Write('введите фамилию'); readln(fam); {ввод с клавиатуры}
   Write('введите год'); readln(god);
   Write('введите специальность'); readln(spec);
   Writeln(fp, fam);                    {запись в файл}
   Writeln(fp, god);  Writeln(fp, spec);  {запись в файл}
  End;
 Close(fp);
Reset(fp);      {открывается файл, указатель на начало файла}
Write('Введите требуемый год рождения'); readln(god_Is);
Writeln('Список фамилий студентов, родившихся в ' , God_Is:4,' году');
While Not Eof (fp) do
 Begin
  Readln(fp, fam);                      {чтение из файла}
  Readln(fp, god);
  Readln(fp, spec);
  If god=god_Is then writeln(fam);      {вывод фамилии на экран}
 End;
Close(fp);
End.
```

В созданном текстовом файле маркеры EoLn записаны после каждого данного.

fam	EoLN	god	EoLN	spec	EoLN	fam	EoLN	……	Eof

Нетипизированные (блочные) файлы

Компонентами файла являются блоки (порции) данных определенного размера. По умолчанию длина блока 128 байт.

С помощью нетипизированных файлов можно организовывать эффективные операции ввода-вывода при работе с внешними файлами, т. к. позволяет организовывать доступ к любым дисковым файлам, независимо от их структуры; используются «быстрые» дисковые операции ввода-вывода.

1) Объявление:

Var < ф.п. >: file;

2) Связывание файловой переменной с физическим файлом на диске:

Assign (< ф.п. >, < 'спецификация файла '>);

3) Открытие нетипизированного файла:

Reset (< ф.п. >, N);

Rewrite (< ф.п.>, N);

N: word – определяет размер компонентов файлов (<= 65535 байт). Если N – отсутствует, то по умолчанию размер равен 128 байт.

Для обеспечения максимальной скорости обмена размер элемента файла должен быть кратным размеру физического сектора (кластера) диска 512 байт, 1024 байт.

Для того, чтобы обеспечить полное чтение всего файла, нужно установить размер элемента равным 1 (иначе будут присутствовать неполные блоки).

4) Обмен информацией: используются процедуры

BlockRead (< ф.п. >; < буфер >; < n1 >;< n2 >);

BlockWrite (< ф.п. >, < буфер >, < n2 >[, < n2 >]);

<ф.п > - имя нетипизированного файла (файловая переменная);

<буфер> - переменная, которая участвует в обмене данными;

< n1 > – число записей, обрабатываемых за один вызов;

< n2 > – число фактически обработанных записей.

Если n2 < n1, то ошибка ввода – вывода. В обеих процедурах нужно следить за тем, чтобы размер переменной-буфера был равен N * n1, где N - размер элемента, указанный в процедурах reset или rewrite. Дополнительно допускается использование процедур и функций seek, filesize и filepos для организации доступа к любому компоненту нетипизированного файла.

5) Закрытие файла

close < ф.п. >;

Пример. Копирование файла. Исходному файлу соответствует файловая переменная Filein, копии соответствует файловая переменная Fileout.

```
Program filecopy;
  Const
     Recsize = 512;                    {размер буфера обмена}
  Var
    Fileout, filein: file;
    Buf: array [1 .. recsize] of char;     {буфер обмена}
    Numw, numr: word; {фактически записанные и фактически считанные}
  Begin
       Assign (filein, 'c:\ inform.doc');
       reset (filein, 1);
       assign (fileout, 'c:\ inform1. doc');
       rewrite (fileout, 1);
        repeat
             blockread (filein, buf, recsize, numr);
             blockwrite (fileout, buf, numr, numw);
        until (numr = 0) or (numw <> numr);
     writeln ('копирование закончено');
     close (filein); close (fileout);
```

```
end.
```

Практическая работа №2

Разработка программ модульной структуры

Цель работы: Овладение навыками разработки программ с использованием рекурсивных алгоритмов и подпрограмм. Знакомство с особенностями реализации риемов модульного программирования.

Подпрограммы. Процедуры и функции

Процедуры и функции представляют относительно самостоятельные фрагменты программы, оформленные особым образом и снабженные именами.

Отличие функции от процедуры: результатом исполнения подпрограммы-функции является некоторое одно значение, присваиваемое имени подпрограммы-функции. Процедура используется в тех случаях, когда необходимо получить несколько результатов.

Подпрограмма представляет собой инструмент, с помощью которого любая программа была бы разделена на ряд относительно независимых друг от друга частей.

Каждая подпрограмма должна быть описана в разделе описаний основной программы или в разделе описания другой подпрограммы: т. е. необходимо указать заголовок подпрограммы и тело. В заголовке объявляется имя подпрограммы и формальные параметры. Тело имеет структуру обычной программы на языке ТР, т. е. состоит из раздела описаний и раздела исполняемых операторов. В разделе описаний подпрограммы могут встретиться описания подпрограммы низшего уровня и т. д.

Все переменные, описанные внутри подпрограммы, называются *локальными* для этой подпрограммы. При входе в подпрограмму низшего уровня становятся доступными не только объявленные в ней переменные, но и сохраняется доступ ко всем именам верхнего уровня, т. е. подпрограмме доступны только те объекты, которые описаны до описания данной подпрограммы. Эти объекты называются *глобальными* по отношению к подпрограмме. Имена, локализованные в подпрограмме, могут совпадать с ранее объявленными глобальными именами. В этом случае локальное имя как бы «закрывает» глобальное и делает его недоступным.

Подпрограмма-процедура

Описание (объявление процедуры):

Procedure < имя > (g_1: t_1; g_2: t_2; … ; var g_i: t_i; …..);
    < раздел описаний локальных имен >
begin
     < операторы >
end;

< имя > - имя процедуры;

g_i – имена формальных параметров;

t_i – типы формальных параметров (простой тип, либо ранее описанный).

Типом любого формального параметра может быть только стандартный или ранее объявленный тип. Поэтому нельзя объявить процедуру следующим образом:

```
Procedure s (a: array [1 .. 10] of real);
```

Если в подпрограмму передается весь массив, то следует предварительно описать его тип, например:

```
Type
   Mas = array [1 .. 10] of real;
.....
procedure s (a: mas);              {mas -  ранее определенный тип}
.....
```

Так как строка является своеобразным массивом, то передача в подпрограмму осуществляется аналогично:

```
Type
     Stroca = string [15];
.....
procedure st (s: stroca);          {stroca -  ранее определенный тип}
.....
```

При описании процедур различают параметры-значения и параметры-переменные.

Параметры-значения определяют исходные данные для работы подпрограммы. Формальные параметры, являющиеся результатом вычислений в процедуре, записываются после слова Var и называются параметрами-переменными.

Обращение к процедуре:

< имя > (b_1, b_2, …);

< имя > - имя вызываемой процедуры;

b_i – фактические параметры, которые должны соответствовать формальным по количеству, типу и порядку следования. Фактические параметры могут быть заданы константами, переменными или выражениями.

При вызове процедуры параметры-значения получает от фактических параметров их значения (создается копия в оперативной памяти). Соответствующие фактические параметры могут быть заданы любым выражением, результат вычисления которого по типу совпадает с типом соответствующего формального параметра.

При вызове процедуры параметры-переменные получают от соответствующих фактических параметров их адреса (копия не создается), т. е. область памяти имеет два имени: имя фактического параметра и имя формального параметра-переменной. Соответствующие фактические параметры могут быть заданы только переменными.

Пример. Создать п/процедуру, вычисляющую сумму двух чисел.

```
……
Procedure Sum (Var s:real; a,b: real);        {s – параметр-переменная (результат)}
begin
   s:=a+b;
end;
……
Sum(s1,12,56);                                {вызов п/процедуры с фактическими параметрами}
……
```

Подпрограмма-функция

Описание (объявление функции):

```
Function < имя > (g1: t1; g2: t2; …..): tф;
   < раздел описаний локальных имен >
begin
   < операторы >
   <имя>:= <результат>;  {имени подпрограммы-функции  присваивается возвращаемый результат}
end;
```

g_i – имена формальных параметров;

t_i – типы параметров (стандартный или ранее описанный);

tф – тип возвращаемого значения.

При обращении к функции указывается имя вызываемой функции и фактические параметры. Вызов подпрограммы-функции осуществляется в выражении, записанном в каком либо операторе (например, в операторе присваивания).

```
< переменная >: = < имя > (b1, b2, ….);
```

Пример. Создать п/функцию, вычисляющую сумму двух чисел.

```
……
Function Sum (a,b:real) : real;
 Var s:real;    {описание рабочей переменной}
 Begin
    S:=a+b;
     Sum:=s;  {имени п/функции присваивается результат}
 End;
……
summa:=Sum(k,m);          {вызов п/функции }
……
```

Примеры использования процедур и функций

Пример. В одномерных массивах a и b подсчитать сумму положительных элементов и количество отрицательных элементов.

```
Program pp;
Const
  Z = 100;
Type
  Mas = array [1.. z] of real;
Var
   Na, nb, I, ka, kb: integer;
   Sa, sb: real;
   A, b: mas;
Procedure vvod (m: integer; var c: mas);
{подпрограмма формирования одномерного массива  датчиком случайных чисел}
var
  J: integer;
Begin
   For j: = 1 to m do
   Begin
       C [j]: = random * 100 – 40;
   End;
  Writeln;
End;
Procedure vivod (m: integer; c: mas);
{подпрограмма вывода элементов одномерного массива  на экран}
var
  J: integer;
Begin
   For j: = 1 to m do
   Begin
      Write (c [j] : 6 : 1, '  ');
   End;
  Writeln;
End;
Procedure sum (c: mas; m: integer; var s: real);
{ подпрограмма подсчета суммы положительных элементов}
var
   j: integer;
begin
  s: = 0;
  for j: = 1 to m do
     if c [j] > 0 then s: = s + c [j];
end;
function kol (c: mas; m: integer): integer;
{функция подсчета количества отрицательных элементов}
var
   k, j: integer;
begin
  k: = 0;
  for j: = 1 to m do
  if c [j] < 0 then k: = k + 1;
  kol: = k; {возвращаемое значение присваивается имени функции}
end;
begin   {основная программа}
 writeln ('введите размер массива a'); readln (na);
 vvod (na, a);              {вызов процедуры vvod}
Writeln('Исходный массив a ');
  vivod (na, a);             {вызов процедуры vivod}
 sum(a, na, sa);           {вызов процедуры sum}
 writeln('сумма положительных элементов sa = ' , sa : 6 : 1);
 ka: = kol (a, na);        {вызов функции kol}
 writeln ('количество отрицательных элементов ka = ' , ka : 3);
 writeln ('введите размер массива b'); readln (nb);
```

```
 vvod (nb, b);            {вызов процедуры vvod}
Writeln('Исходный массив b ');
vivod (nb, b);            {вызов процедуры vivod}
 sum (b, nb, sb);         {вызов процедуры sum}
 writeln ('сумма положительных элементов sb = ' , sb : 6 : 1);
 kb: = kol (b, nb);       {вызов функции  kol}
 writeln ('количество отрицательных элементов kb = ' , kb : 3);
end.
```

Пример. Вычислить $Z = a^m$, m – целое, a – вещественное по формуле:

$$Z = a^m = \begin{cases} 1, m = 0 \\ a^m, m > 0 \\ 1/a^m, m < 0 \end{cases}$$, учитываем, что $1/a^m = (1/a)^m$

```
Program stepen;
Uses crt;
Var
   A,z: real;
   M: integer;
Procedure st (n: integer; x: real; var y: real);
 Var I: integer;
 Begin
    Y: = 1;   {вычисляется x^n , как произведение n сомножителей}
     For I: = 1 to n do
     Y: = y * x;
 End;
Begin            {основная программа}
  Clrscr;
   Write ('введите a и m');  readln (a, m);
   If m = 0 then z: = 1
         Else
              If m > 0 then st (m, a, z)      Else st (m, 1/a, z);
  Writeln ('z = ' , z : 10 : 4);
End.
```

Пример. В матрице A(n, n) подсчитать в каждой строке количество отрицательных элементов.

```
Program podprog;
Uses crt;
Const n = 10;
  Type
      Mass = array [1 .. n, 1 .. n] of real;
      Mass1 = array [1 .. n] of integer;
Var
   I, j: integer;
   A: mass;
   K: mass1;
Procedure kol (var y: mass1; x: mass, m: integer);
Var
    i, j: integer;
Begin
    For i: = 1 to m do
        Begin
           Y [i]: = 0;
           For j: = 1 to m do
              If x [i, j] < 0 then y [i]: = y [i] + 1;
         End; End;
Begin                    {основная программа}
Clrscr;
Writeln(' Исходный массив');
For I: = 1 to n
```

```
  Begin
    For j: = 1 to n
      Begin
        A[I, j]:= -100 +random(200); {используется датчик случайных чисел}
        Write (a [I, j] : 6 : 2);
     End;     Writeln;
  End;
Kol (k, a, n);                {вызов процедуры}
Writeln('Количество отрицательных элементов в каждой строке');
For i: = 1 to n do
  Write (k [i] : 4);
Writeln;
End.
```

Рекурсия и опережающее описание

Рекурсия – это такой способ организации вычислительного процесса, при котором подпрограмма в ходе выполнения обращается сама к себе.

Пример. Вычислить n!. По определению n!=1*2*3*…*n. Также справедливо соотношение n!=(n-1)!*n.

```
Program Factorial;
 Var
   N: word;
   F: real;
Function Fact(n: word): real;
  Begin
    If n=0 then Fact:=1
    Else
      Fact:= Fact(n-1)* n;          {подпрограмма вызывает саму себя}
  End;
Begin                               {основная программа}
   Write('введите число n : '); readln(n);
   F:= Fact(n);
   Writeln(n:2, '!=', F:8:0);
End.
```

Рекурсивная форма организации алгоритма обычно выглядит изящнее итерационной и дает более компактный текст программы, но при выполнении, как правило, медленнее и может вызвать переполнение программного стека. При каждом вызове подпрограммой самой себя ее состояние должно быть сохранено во временной области памяти, называемой программным стеком. Состояние подпрограммы определяется адресом, где произошла остановка вызывающей подпрограммы, и значением всех ее локальных переменных. Поэтому одна из проблем, возникающая при организации рекурсивных алгоритмов – это переполнение программного стека. Однако необходимо понимать, что существуют классы задач, которые решаются только с помощью рекурсивных алгоритмов.

Рекурсивный вызов может быть косвенным, т.е. когда подпрограмма вызывает саму себя через другую подпрограмму.

```
Procedure A(j:byte);
  Begin
    ……..
     B(j);            {вызов подпрограммы B}
    ……..
  End;

Procedure B(j:byte);
  Begin
    ……..
     A(j);            {вызов подпрограммы A}
    ……..
  End;
```

Однако при таком описании не выполняется главный принцип, что каждый идентификатор перед использованием должен быть описан.

Чтобы такие вызовы стали возможны, вводится опережающее описание.

```
Procedure B(j:byte); Forward;  {опережающее описание B}
Procedure A(j:byte);
 Begin
  ……..
   B(j);          {вызов подпрограммы B}
  ……..
 End;

Procedure B;  {краткий заголовок}
 Begin
  ……..
   A(j);          {вызов подпрограммы A}
  ……..
 End;
```

Опережающее описание заключается в том, что объявляется лишь заголовок подпрограммы B, а ее тело заменяется стандартной директивой Forward. Теперь в подпрограмме A можно использовать вызов подпрограммы B, т.к. компилятор теперь знает все ее формальные параметры и может правильным образом организовать ее вызов.

Пример. Вычислить сумму $\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\ldots+\frac{1}{n}$, используя рекурсию.

```
Program Summa;
Const k=10;
Var sum: real;
Function S (n: integer): real;
  Begin
    If n=1 then S:=1/2
     Else
       S:=S(n-1)+ 1/(n+1);       {рекурсивный вызов}
  End;
Begin            {основная программа}
  Sum:= S(k);
  Writeln('сумма ряда = ' , sum:10:4);
End.
```

Модули

В языке Turbo Pascal модуль (unit) по определению считается отдельной программой. Если подпрограмма (см. раздел 4) является структурным элементом Pascal-программы и не может существовать вне нее, то модуль представляет собой отдельно хранимую и независимо компилируемую программную единицу.

В самом общем виде модуль представляет собой совокупность (коллекцию) программных ресурсов, предназначенных для использования другими модулями и программами. Под ресурсами в данном случае понимаются любые программные объекты языка Turbo Pascal - константы, типы, переменные, подпрограммы. Важно понимать, что модуль сам по себе не является выполняемой программой - его объекты *используются* другими программными единицами.

Модули представляют собой прекрасный инструмент для разработки библиотек прикладных программ и мощное средство модульного программирования. Важная особенность модулей заключается в том, что компилятор Turbo Pascal размещает их программный код в отдельном сегменте памяти. Максимальная длина сегмента не может превышать 64 Кбайта, однако количество одновременно используемых модулей ограничивается лишь доступной памятью, что дает возможность создавать весьма крупные программы.

Структура модуля

Модуль имеет следующую структуру:

```
UNIT <имя модуля>;
  INTERFACE
```

```
    <интерфейсная часть>
  IMPLEMENTATION
    <исполняемая часть>
  BEGIN
     <инициирующая часть>
END.
```

Здесь UNIT - зарезервированное слово (единица), которое начинает заголовок модуля.

<имя модуля> - правильный идентификатор. Имя файла, в котром сохраняется исходный код модуля *должно совпадать* с именем модуля.

INTERFACE - зарезервированное слово (интерфейс), начинает интерфейсную часть модуля (обязательный раздел).

IMPLEMENTATION - зарезервированное слово (выполнение), начинает исполняемую часть (обязательный раздел);

BEGIN - зарезервированное слово, начинает инициирующую часть. Эта часть модуля необязательна;

END - зарезервированное слово - признак конца модуля.

Таким образом, модуль состоит из заголовка и трех разделов, любой из которых может быть пустой.

Для правильной работы среды Turbo Pascal и возможности подключения средств, облегчающих разработку крупных программ, имя модуля должно совпадать с именем дискового файла, в который помещается исходный текст модуля. Имя модуля служит для его связи с другими модулями и основной программой. Эта связь устанавливается специальным предложением

```
USES <сп.модулей>
```

Здесь USES - зарезервированное слово (использует); <сп. модулей> - список модулей, с которыми устанавливается связь; элементами списка являются имена модулей, отделяемые друг от друга запятыми, например:

```
Uses CRT, Graph, Global;
```

Если объявление USES ... используется, оно должно открывать раздел описаний основной программы. Модули могут использовать другие модули. Предложение USES в модулях может следовать либо сразу за зарезервированным словом INTERFACE, либо сразу за словом IMPLEMENTATION.

Интерфейсная часть открывается зарезервированным словом INTERFACE. В этой части содержатся объявления всех глобальных объектов модуля (типов, констант, переменных и подпрограмм), которые должны стать доступными основной программе и/или другим модулям. При объявлении подпрограмм в интерфейсной части указывается только их заголовок.

Все константы и переменные, объявленные в интерфейсной части модуля, равно как и глобальные константы и переменные основной программы, помещаются компилятором Турбо Паскаля в общий сегмент данных (максимальная длина сегмента 65536 байт). Порядок появления различных разделов объявлений и их количество может быть произвольным.

Исполняемая часть содержит тела подпрограмм, объявленных в интерфейсной части. В ней могут объявляться локальные для модуля объекты - вспомогательные типы, константы, переменные и подпрограммы, а также - метки, если они используются в инициирующей части.

Описанию подпрограммы, объявленной в интерфейсной части модуля, в исполняемой части должен предшествовать заголовок, в котором можно опускать список формальных переменных (и тип результата для функции), так как они уже описаны в интерфейсной части. Но если заголовок подпрограммы приводится в полном виде, т.е. со списком формальных параметров и объявлением результата, он должен совпадать с заголовком, объявленным в интерфейсной части,

Локальные переменные и константы, а также все программные коды, порожденные при компиляции модуля, помещаются в общий сегмент памяти.

В *инициирующей части* размещаются исполняемые операторы, содержащие некоторый фрагмент программы. Эти операторы исполняются до передачи управления основной

программе и обычно используются для подготовки ее работы или инициализации объектов модуля.

Таким образом, механизм модулей позволяет скрыть детали реализации тех или иных программных подсистем, предоставив в распоряжение использующих программ строго определенную совокупность интерфейсных объектов. Если необходимо, например, расширить модуль введением новых процедур или изменить реализацию какой-либо процедуры, то если интерфейс модуля при этом останется неизменным, такая модификация никак не отразится на использующих программах.

Компиляция и использование модулей

Модуль компилируется точно таким же образом, как и обычные программы; возможна компиляция из интегрированной среды или с помощью компилятора командной строки. Но так как модуль не является непосредственно выполняемой единицей, то в результате его компиляции образуется дисковый файл с расширением TPU (Turbo Pascal Unit), при этом имя файла берется из имени файла с исходным текстом модуля.

Для того, чтобы получить доступ к интерфейсным объектам модуля, необходимо указать в программе или модуле имя нужного TPU-файла. Соответствующая конструкция называется спецификацией используемых модулей (uses).

При наличии спецификации использования считаются известными все описания из интерфейсной части подключенного модуля. К интерфейсным объектам модуля можно обращаться в программе точно так же, как если бы они были описаны в самой этой программе.

Схема использования модулей может образовывать древовидную структуру любой сложности, но при этом недопустимо явное или косвенное обращение модуля к самому себе. Так, например, следующие отношения являются ошибочными, т.к. к модулю A подключен модуль B (модуль A обращается к ресурсам модуля B) и в тоже время к модулю B подключен модуль A (модуль B обращается к ресурсам модуля A):

```
unit A;                 unit B;
interface               interface
  uses B;                 uses A;
end.                    end.
```

Если программа использует несколько модулей, то их разделы инициализации будут выполнены в том же порядке, в котором эти модули перечислены в спецификации использования.

При трансляции программы или модуля, использующего другие модули, компилятор последовательно отыскивает файлы, содержащие коды используемых модулей, с тем, чтобы подключить их к компилируемой программе. При этом компилятор работает по следующей схеме:

– Компилятор просматривает содержимое системного библиотечного файла модулей torbo.tpl (Turbo Pascal Library). Этот файл будет кратко описан далее в этом разделе.

– Если искомый модуль не найден в файле TURBO.TPL, то компилятор осуществляет поиск соответствующего TPU-файла в текущем каталоге.

– Если в текущем каталоге нужный файл не найден, то поиск продолжается в каталогах, заданных в альтернативе **Options/Directories/Unit Directories** для интегрированной среды или в параметре /O вызова TPC- компилятора.

– Если на предыдущих шагах TPU-файл не найден, то компилятор прекращает работу и выдает диагностическое сообщение об ошибке.

– Если компилятор активизирован посредством альтернатив **Compile/Make** или **Compile/Build**, то вышеуказанные шаги проводятся в поисках исходных текстов используемых модулей, которые будут оттранслированы перед трансляцией самой программы. При этом подразумевается, что имя файла с исходным текстом модуля совпадает с именем модуля и имеет расширение .PAS.

При стандартных настройках интегрированной среды результаты компиляции размещаются в оперативной памяти. Для создания дискового файла с расширением TPU (компилированный модуль) или с расширением EXE (компилированная программа) необходимо настроить интегрированную среду.

Команда **File/Change dir ...** позволяет задать текущий каталог (в нем будет создан соответствующий файл).

Команда **Compile/Destination** позволяет задать, куда именно разместить результаты компиляции. Необходимо задать значение – **Disk.**

Далее программа или модуль компилируется с использованием любого из режимов (**Compile, Make, Build**) и соответствующий файл будет создан на диске.

Стандартные модули

Turbo Pascal имеет несколько стандартных модулей, в которых, собственно, и содержатся все упоминаемые в книге системные процедуры и функции. Имена этих модулей следующие: SYSTEM, DOS, CRT, PRINTER, GRAPH, OVERLAY, TVRBO3 и GRAPH3. Модули GRAPH, TURBO3 и GRAPH3 содержатся в одноименных TPU-файлах, остальные входят в состав библиотечного файла TURBO.TPL. Лишь один модуль SYSTEM подключается к любой программе автоматически, все остальные становятся доступны только после указания их имен в списке, следующем за словом USES.

Модуль **SYSTEM**. В него входят все процедуры и функции стандартного Паскаля, а также встроенные процедуры и функции Турбо Паскаля, которые не вошли в другие стандартные модули (например, INC, DEC, GETDIR и т.п.). Как уже отмечалось, модуль SYSTEM подключается к любой программе независимо от того, объявлен ли он в предложении USES или нет, поэтому его глобальные константы, переменные и подпрограммы считаются встроенными в Турбо Паскаль.

Модуль **PRINTER.** Делает доступным вывод текстов на матричный принтер. В нем определяется файловая переменная LST типа TEXT, которая связывается с логическим устройством PRN.

Модуль **CRT**. В нем сосредоточены процедуры и функции, обеспечивающие управление текстовым режимом работы экрана. С помощью входящих в модуль подпрограмм можно перемещать курсор в произвольную позицию экрана, менять цвет выводимых символов и окружающего их фона, создавать окна. Кроме того, в модуль включены также процедуры «слепого» чтения клавиатуры и управления звуком (см. пункт 3.5.2).

Модуль **GRAPH**. Содержит обширный набор типов, констант, процедур и функций для управления графическим режимом работы экрана. С помощью подпрограмм, входящих в модуль GRAPH, можно создавать разнообразные графические изображения и выводить на экран текстовые надписи стандартными или разработанными программистом шрифтами. Подпрограммы модуля GRAPH после соответствующей настройки могут поддерживать различные типы аппаратных графических средств. Настройка на имеющиеся в распоряжении программиста технические средства графики осуществляется специальными программами – драйверами (BGI-файлы), которые не входят в библиотечный файл GRAPH.TPU, но поставляются вместе с ним.

Модуль **DOS**. В модуле собраны процедуры и функции, открывающие доступ программам к средствам дисковой операционной системы MS DOS (см. пункт 3.9.4).

Модуль **OVERLAY**. Он необходим при разработке громоздких программ с перекрытиями.

Два библиотечных модуля TURBO3 и GRAPH3 введены для совместимости с ранней версией 3.0 системы Турбо Паскаль.

Практическая работа №3

Динамическая память

Цель работы: Овладение навыками размещения данных в динамической памяти. Освоение приемов работы со связанными динамическими структурами.

Указатели и динамическая память

Все переменные, объявленные в программе, являются статическими и размещаются в одной непрерывной области оперативной памяти (ОП), которая называется сегментом данных. Максимальный размер сегмента 64 Кб.

Динамическая память (куча) - это оперативная память ПК, предоставляемая программе при ее работе, за вычетом сегмента данных (64 Кбайт), программного стека (обычно 16

Кбайт) и собственно тела программы. Размер динамической памяти можно варьировать в широких пределах. По умолчанию этот размер определяется всей доступной памятью ПК.

Для управления размерами динамической памяти служит директива компилятора $M. Директива размещается в самом начале программы и содержит три целочисленных параметра. Первый параметр определяет максимальный размер памяти, выделяемый под программный стек, а два следующих параметра задают минимальный и максимальный размеры динамической памяти ({$M 16384б, 1024, 650000}).

Динамическое размещение данных означает использование динамической памяти непосредственно при работе программы. В отличие от этого статическое размещение осуществляется компилятором Турбо Паскаля в процессе компиляции программы. При динамическом размещении заранее не известны ни тип, ни количество размещаемых данных, к ним нельзя обращаться по именам, как к статическим переменным. Обращение к динамическим данным осуществляется через так называемые ссылочные имена (указатели).

Адреса и указатели

Указатель - это переменная, которая в качестве своего значения содержит адрес байта памяти.

В ПК адреса задаются совокупностью двух шестнадцатиразрядных слов, которые называются сегментом и смещением. *Сегмент* - это участок памяти, имеющий длину 65536 байт (64 Кбайт) и начинающийся с физического адреса, кратного 16 (т.е. 0, 16, 32, 48 и т.д.). *Смещение* указывает, сколько байт от начала сегмента необходимо пропустить, чтобы обратиться к нужному адресу.

Как правило, в Турбо Паскале указатель связывается с некоторым типом данных. Такие указатели называют *типизированными*. Для объявления типизированного указателя используется значок ^, который помещается перед соответствующим типом.

<имя указателя>: ^ <базовый тип>

Пример:

```
type
PerconPointer = ^PerconRecord;     { тип  - указатель на запись}
 PerconRecord = record
   Name : string;
   Job    : string;
   Next : PerconPointer:              {указатель на запись}
 end;
var
 p1 : ^ integer;           {указатель на целый тип}
 p2 : ^ real;              {указатель на вещественный тип}
p3: PerconPointer;      {указатель на запись}
```

Указатели могут ссылаться на еще не объявленный тип данных (см. пример). При объявлении типа Perconpointer мы сослались на тип Perconrecord, который предварительно в программе объявлен, не был. Однако объявление этого типа должно быть сделано в этом же разделе описания типов.

В Турбо Паскале можно объявлять указатель и не связывать его при этом с каким-либо конкретным типом данных. Для этого служит стандартный тип POINTER.

Var

<имя указателя>: Pointer;

Пример: var pp : pointer;

Указатели такого рода называют *нетипизированными*. Поскольку нетипизированные указатели не связаны с конкретным типом, с их помощью удобно динамически размещать данные, структура и тип которых меняются в ходе работы программы.

Среди всех возможных указателей в языке выделяется один специальный указатель (nil), который "никуда не указывает". Указатель nil считается константой, совместимой с любым ссылочным типом. Иными словами, это значение молено присваивать любому указательному типу.

Над значениями ссылочных типов допускаются две операции сравнения *на равенство и неравенство*; эти операции проверяют, ссылаются ли два указателя на одно и то же место в памяти, и обозначаются обычным образом - знаками '=' и '<>', например:

```
Sign  := P1=P2;

if P1 <> nil then .  .  .
```

Для того чтобы указателю присвоить некоторое значение, можно воспользоваться унарной операцией *взятия указателя*. Например, если имеется описание

```
var
  i : integer;
```

то применение этой операции к переменной i дает в качестве результата значение типа "указатель на целое"; поэтому оператор присваивания

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

правомочен, так как в обеих его частях стоят конструкции одного и того же типа. В результате такого присваивания P1 получит в качестве своего нового значения указатель на переменную i (или, попросту говоря, адрес переменной i).

Еще одна операция, *разыменование* (обозначается символом ^) по определению имеет тип, совпадающий с базовым типом переменной-указателя; в частности, P1^ считается переменной целого типа, поэтому возможен оператор присваивания:

```
P1^:=P1^ + 2;
```

Процедуры и функции для работы с динамической памятью

Функция ADDR(X). Возвращает результат типа POINTER, в котором содержится адрес аргумента.

Здесь X - любой объект программы (имя любой переменной, процедуры, функции). Аналогичный результат возвращает операция @ .

Функция CSEG. Возвращает значение, хранящееся в регистре CS микропроцессора (в начале работы программы в регистре CS содержится сегмент начала кода программы). Результат возвращается в слове типа WORD.

Процедура DISPOSE(TP). Возвращает в кучу фрагмент динамической памяти, который ранее был зарезервирован за типизированным указателем TP. При повторном использовании процедуры применительно к уже освобожденному фрагменту возникает ошибка периода исполнения.

Функция DSEG. Возвращает значение, хранящееся в регистре DS микропроцессора (в начале работы программы в регистре DS содержится сегмент начала данных программы). Результат возвращается в слове типа WORD.

Процедура FREEMEM(P, S). Возвращает в кучу фрагмент динамической памяти длиной S байт, который ранее был зарезервирован за нетипизированным указателем P. При повторном использовании процедуры применительно к уже освобожденному фрагменту возникает ошибка периода исполнения.

Процедура GETMEM(P,S). Резервирует за нетипизированным указателем P фрагмент динамической памяти требуемого размера (S байт). За одно обращение к процедуре можно зарезервировать не более 65521 байта динамической памяти. Если нет свободной памяти требуемого размера, возникает ошибка периода исполнения.

Процедура MARK(PTR). Запоминает текущее значение указателя кучи HEAPPTR. Здесь PTR - указатель любого типа, в котором будет возвращено текущее значение HEAPPTR, Используется совместно с процедурой RELEASE для освобождения части кучи.

Функция MAXAVAIL. Возвращает размер в байтах наибольшего непрерывного участка кучи. Результат имеет тип LONGINT. За один вызов процедуры NEW или GETMEM нельзя зарезервировать памяти больше, чем значение, возвращаемое этой функцией.

Функция MEMAVAIL. Возвращает размер в байтах общего свободного пространства кучи. Результат имеет тип LONGINT.

Процедура NEW(TP). Резервирует фрагмент кучи для размещения переменной. Здесь TP - типизированный указатель. За одно обращение к процедуре можно зарезервировать не

более 65521 байта динамической памяти. Если нет свободной памяти требуемого размера, возникает ошибка периода исполнения.

Функция OFS(X). Возвращает значение типа WORD, содержащее смещение адреса указанного объекта. Здесь X - выражение любого типа или имя процедуры.

Функция PTR(SEG, OFS). Возвращает значение типа POINTER по заданному сегменту SEG и смещению OFS. Здесь SEG, OFS - выражения типа WORD. Значение, возвращаемое функцией, совместимо с указателем любого типа.

Процедура RELEASE(PTR). Освобождает участок кучи. Здесь PTR - указатель любого типа, в котором предварительно было сохранено процедурой MARK значение указателя кучи. Освобождается участок кучи от адреса, хранящегося в PTR, до конца кучи. Одновременно уничтожается список всех свободных фрагментов, которые, возможно, были созданы процедурами DISPOSE или FREEMEM.

Функция SEG(X). Возвращает значение типа WORD, содержащее сегмент адреса указанного объекта. Здесь X - выражение любого типа или имя процедуры.

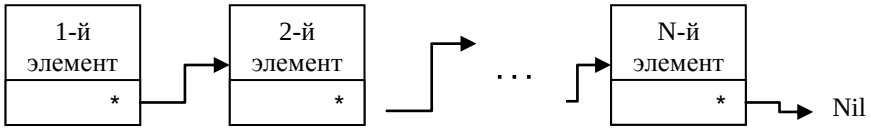
Функция SIZEOF(X). Возвращает длину в байтах внутреннего представления указанного объекта. Здесь X - имя переменной, функции или типа.

Связанные динамические данные

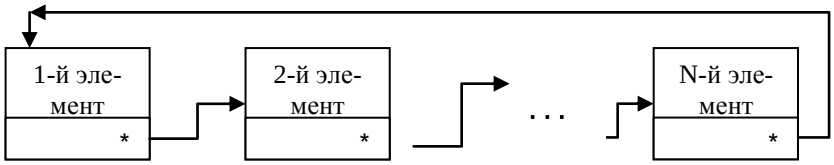
Когда трудно или вообще невозможно предсказать число объектов, обрабатываемых программой, используются динамические объекты, которые создаются по мере необходимости и объединяются (связываются) с уже существующими объектами с помощью указателей. Это дает возможность строить из динамических объектов списки, кольца и другие, более сложные структуры.

Интенсивная работа с подобными объектами характерна для задач моделирования систем массового обслуживания, где программа может, например, имитировать работу бензоколоки или станции техобслуживания; в этом случае каждый объект в списке представляет собой модель автомобиля, а весь список - модель очереди на обслуживание. Создание и удаление объектов отображают реальные события, связанные, соответственно, с прибытием автомобиля и окончанием его обслуживания.

Линейные списки – представляют собой совокупность линейно связанных однородных элементов, для которых разрешается добавление элементов между двумя другими и удалении любого элемента. Бывают *односвязные* и *двусвязные* линейные списки.



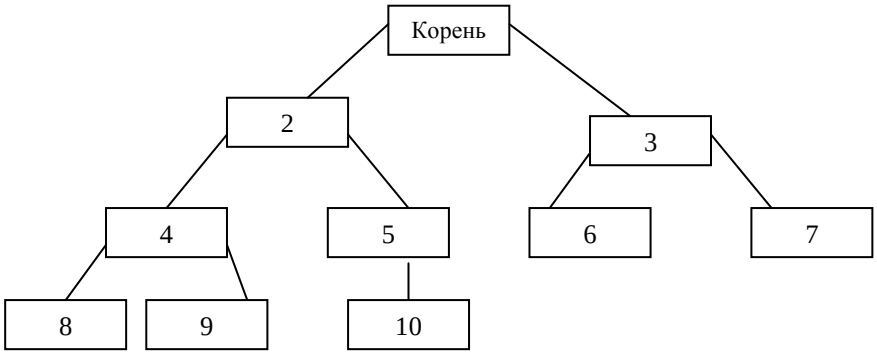
Кольцевые списки – это линейные списки, имеющие дополнительную связь между последним и первым элементами списка.



Очередь – частный случай линейного односвязного списка, для которого разрешены только два действия: добавление элементов в конец, удаление элементов из начала.

Стек – частный случай линейного односвязного списка, для которого разрешено добавлять и удалять элементы только с одного конца списка, называемого головой стека.

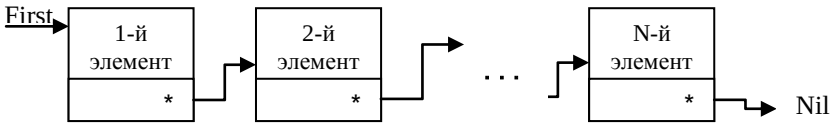
Деревья – динамические связанные данные иерархической структуры произвольной конфигурации. Дерево является рекурсивной структурой данных, поскольку каждое поддерево также является деревом. Частный случай – бинарное дерево. Бинарное дерево состоит из узлов (вершин), которые кроме данных содержат не более двух ссылок на различные бинарные деревья. Узел, не имеющий поддеревьев, называется *листом*. Исходящие узлы называются *предками*, входящие – *потомками*.



Пирамида – упорядоченное дерево, в котором значения вершин всегда больше или меньше при переходе на следующий уровень.

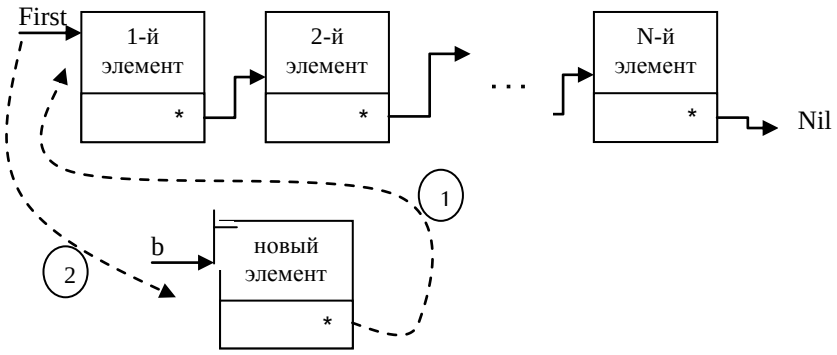
Использование указателей

Пример. Создать линейный односвязный список, содержащий сведения о 10 автомобилях (марка, год выпуска). Вывести элементы списка на экран.




```
Program Spisok;
Const n=10;
type
   Link = ^Auto;
   Auto = record
     Name : string;          { марка автомобиля }
     God : word;             { год выпуска }
     Next : Link;            {поле для связи со следующим объектом в списке}
  end;
Var
First : Link;              { указатель на начало списка }
a, b : Link;               { рабочие указатели }
K: integer;
Begin
  Clrscr;
{создание первого элемента списка}
  New(a);                  { резервируется динамическая память для указателя }
  Write('введите марку автомобиля'); Readln(a^.Name);
  Write('введите год выпуска'); Readln(a^.God);
  a^.Next:= Nil;           { первый элемент ни с чем не связан }
  First := a;              {First указывает на  первый элемент}
{создание остальных  элементов списка}
  For k:=1 to (n-1) do
   begin
    New(b);                { выделяется динамическая память для нового элемента }
    Write('введите марку автомобиля'); Readln(b^.Name);
    Write('введите год выпуска'); Readln(b^.God);
    b^.Next:= Nil;                 { новый элемент ни с чем не связан }
    a^.Next:=b;                    {предыдущий  элемент ссылается на новый}
    a:=b;                          {текущим становится  новый элемент}
   end;
{вывод  элементов списка на экран}
  a:= First;                       {текущим становится  первый элемент}
  Repeat
    Writeln(a^.Name:15, a^.God:6);
    a:=a^.Next;                    {текущим становится  следующий элемент}
  Until a=Nil;
  ……..
```

Пример. Привести фрагмент программы, который в созданном линейном списке (предыдущий пример) добавит новый элемент в начало списка. На рисунке приведена схема добавления нового элемента в начало списка.

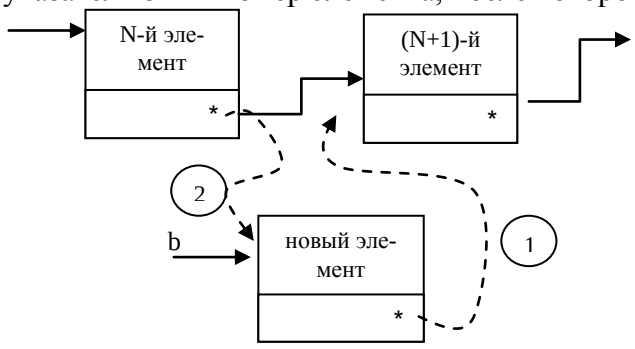


.....

```
new(b);                    { выделяется динамическая память для нового элемента }
Write('введите марку автомобиля'); Readln(b^.Name);
Write('введите год выпуска'); Readln(b^.God);
b^.Next := First;          { новый элемент ссылается на первый} {1}
First := b;                { First указывает на новый элемент (первый) }  {2}
```

.....

Пример. В созданном линейном списке добавить новый элемент после элемента, на который указывает указатель a. N- номер элемента, после которого добавляется новый.



.....

```
new(b);                    { создаем новый элемент }
Write('введите марку автомобиля'); Readln(b^.Name);
Write('введите год выпуска'); Readln(b^.God);
a:= First;                 {текущим становится  первый элемент}
For k:=1 to n-1 do
  a:=a^.Next;              {перебираем элементы списка до (n)-го}
 b^.Next:=a^.Next;   {1}
 a^Next:=b;                {меняем ссылки элементов }  {2}
```

.....

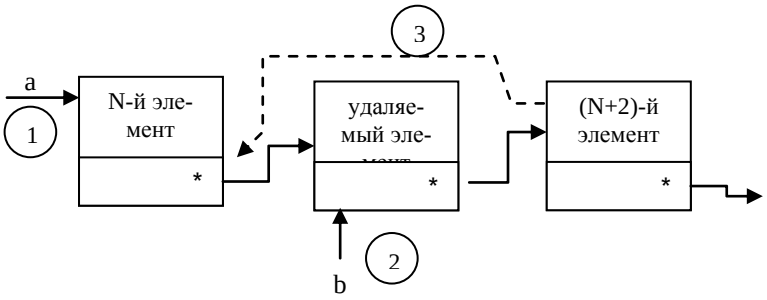
Пример. В созданном линейном списке удалить первый элемент.

.....

```
a:=First;                  {текущим становится  первый элемент}
First := a^.Next;          {First указывает теперь на  второй  элемент}
Dispose(a);                {удаляем первый элемент}
```

.....

Пример. В созданном линейном списке удалить элемент после элемента, на который указывает указатель a. N- номер элемента, после которого удаляется элемент.




```
.........
a:= First;                {текущим становится  первый элемент}
For k:=1 to n-1 do
  a:=a^.Next;             {перебираем элементы списка до (n)-го} {1}
  b:=a^.Next;             {указывает на удаляемый элемент }    {2}
  a^.Next :=b^.Next;      {меняем ссылку элемента }     {3}
  Dispose(b);             {удаляем из динамической памяти элемент}
.........
```

Приложение 1

ФОНД ОЦЕНОЧНЫХ СРЕДСТВ ДЛЯ ПРОВЕДЕНИЯ ПРОМЕЖУТОЧНОЙ АТТЕСТАЦИИ ОБУЧАЮЩИХСЯ ПО ПРАКТИКЕ

1. Описание фонда оценочных средств (паспорт)

№ компе-тен-ции	Элемент компетенции	Этапы/ тема исследований	ФОС
ОК-2	готовность к кооперации с коллегами, работе в коллективе, знание принципов и методы организации и управления малыми коллективами	**1**.Подготовительный этап	Вопросы к зачету:№ 1.1 Отчет по практике Дневник по практике
ОПК-1	владение широкой общей подготовкой (базовыми знаниями) для решения практических задач в области информационных систем и технологий	**2**.Исследовательский этап 2.1 Обработка файловых структур данных 2.2 Разработка программ модульной структуры 2.3 Динамическая память	Вопросы к зачету:№ 2.1-2.12 Отчет по практике Дневник по практике Вопросы к зачету:№ 2.13-2.23 Отчет по практике Дневник по практике Вопросы к зачету:№ 2.24-2.34 Отчет по практике Дневник по практике
ОПК-5	способность использовать современные компьютерные технологии поиска информации для решения поставленной задачи, критического анализа этой информации и обоснования принятых идей и подходов к решению	**1**.Подготовительный этап **3.**Обработка и анализ полученной информации (материала)	Вопросы к зачету:№ 3.1 Отчет по практике Дневник по практике Вопросы к зачету:№ 3.1 Отчет по практике Дневник по практике

ПК-23	готовность участвовать в постановке и проведении экспериментальных исследований.	**1**.Подготовительный этап	Вопросы к зачету:№ 4.1 Отчет по практике Дневник по практике
ПК-24	способность обосновывать правильность выбранной модели, сопоставляя результаты экспериментальных данных и полученных решений.	**3.**Обработка и анализ полученной информации (материала)	Вопросы к зачету:№ 5.1, 5.2 Отчет по практике Дневник по практике

2.Вопросы к зачету с оценкой

№ п/п	Компетенции		ВОПРОСЫ К ЗАЧЕТУ	№ и наименование раздела
	Код	Определение		
1	2	3	4	5
1.	**ОК-2**	готовность к кооперации с коллегами, работе в коллективе, знание принципов и методов организации и управления малыми коллективами	**1.** Какие существуют методы организации и управления малыми коллективами?	1.Подготовительный этап
2.	**ОПК-1**	владение широкой общей подготовкой (базовыми знаниями) для решения практических задач в области информационных систем и технологий	**1.** Что называется файлом? В чем отличие файла от массива? **2.** Что называется записью? Как описывается структура записи? **3.** Как осуществляется доступ к полям записи? **4.** Допустимы ли различные типы данных для элементов одной записи? **5.** Какие файловые типы доступны в Pascal? **6.** Охарактеризуйте внутреннюю структуру файлов различного типа. **7.** Сформулируйте основные этапы работы с файлом любого типа. **8.** Процедуры чтения и записи для файлов различного типа. **9.** Дайте понятие текущего указателя файла.	2.Исследовательский этап 2.1 Обработка файловых структур данных

			10. Приведите перечень процедур и функций, реализующих специальные операции над файлами. **11.** Как устанавливается размер буфера обмена для текстовых файлов? **12.** Каковы преимущества использования файлов? **13.** В чем заключается метод модульного программирования? **14.** Сформулируйте определения подпрограммы-функции и подпрограммы-процедуры. В чем их отличие? **15.** Покажите структуру подпрограмм. В каком разделе программы они описываются? **16.** Укажите когда целесообразно использовать подпрограммы, какие удобства они предоставляют пользователю. **17.** Укажите как осуществляется обращение к подпрограмме-процедуре, подпрограмме-функции. **18.** Могут ли подпрограммы применяться без параметров? **19.** В чем отличие модуля от под программы? **20.** Перечислите основные разделы модуля. Что содержат эти разделы? **21.** Назовите стандартные модули Pascal. Дайте их краткую характеристику. **22.** Какие требования необходимо выполнить при компиляции программы с подключенными модулями? **23.** Чем отличается видимость объектов модуля?	2.2 Разработка программ модульной структуры
			24. Что такое динамическая память? **25.** Чем отличаются статические переменные от динамических? **26.** Как определяется тип динамической переменной? **27.** Как установить требуемый размер динамической памяти? **28.** Как порождаются и уничтожаются динамические переменные?	2.3 Динамическая память

			29. Какие типы указателей используются при работе с динамической памятью? **30.** Перечислите стандартные процедуры и функции для работы с динамической памятью. **31.** Какова технология создания списка любого вида в динамической памяти? **32.** Какова технология добавления элемента связанного списка в динамической памяти? **33.** Какова технология удаления элемента связанного списка в динамической памяти? **34.** Какова технология перебора элементов связанного списка в динамической памяти?	
3.	**ОПК-5**	способность использовать современные компьютерные технологии поиска информации для решения поставленной задачи, критического анализа этой информации и обоснования принятых идей и подходов к решению	**1.**Какие компьютерные технологии поиска информации существуют? **2.**Технология анализа информации.	1.Подготовительный этап 3. Обработка и анализ полученной информации (материала)
4.	**ПК-23**	готовность участвовать в постановке и проведении экспериментальных исследований.	**1.** Какие методики проведения экспериментальных исследований существуют?	1.Подготовительный этап
5.	**ПК-24**	способность обосновывать правильность выбранной модели, сопоставляя результаты экспериментальных данных и полученных решений.	**1.** Методология определения целей и задач проведения экспериментальных исследований. **2.** Современные инструментальные средства планирования экспериментов и анализа их результатов	3. Обработка и анализ полученной информации (материала)

3. Описание показателей и критериев оценивания компетенций

Показатели	Оценка	Критерии
Знать **ОК-2:** − методы организации и управления малыми коллективами; **ОПК-1:** − теоретические основы общей подготовки для решения практических задач в области информационных систем и технологий; **ОПК-5:** − основные информационно- коммуникационные технологии поиска информации; **ПК-23:** −методики проведения экспериментальных исследований. **ПК-24:** – методологию определения целей и задач проведения экспериментальных исследований. Уметь **ОК-2:** - работать в коллективе и применять различные методы организации и управления малыми коллективами; **ОПК-1:** − применять различные информационные системы и технологии для решения практических задач в производственной сфере; **ОПК-5:** − использовать возможности информационно-вычислительных сетей для решения практических задач; **ПК-23:** –планировать постановку и проведение экспериментальных исследований. **ПК-24:** – проводить экспериментальные исследования; Владеть **ОК-2**: – методиками организации и управления малыми коллективами; **ОПК-1:** –базовыми знаниями для решения практических задач в области информационных систем и технологий; **ОПК-5:** –навыками поиска информации для решения поставленной задачи и обоснования принятых идей и подходов к решению вычислительных задач; **ПК-23:**	**отлично**	Выставляется практиканту в том случае, если он выполнил весь объем работы, предусмотренный программой практики, продемонстрировав: – соответствие содержания отчета программе прохождения практики (отчет собран в полном объеме); - правильное выполнение практических задач в области информационных систем и технологий; – студент демонстрирует системность и глубину знаний, полученных при прохождении практики; – стилистически грамотно, логически правильно излагает ответы на вопросы; – дает исчерпывающие ответы отражающие наличие базовых знаний в области информационных систем и технологий; – владеет методикой проведения экспериментальных исследований; –владеет навыками поиска информации для решения поставленной задачи и обоснования принятых идей и подходов к решению вычислительных задач.
	хорошо	Выставляется практиканту в том случае, если он выполнил весь объем работы, предусмотренный программой практики, продемонстрировав: – соответствие содержания отчета программе прохождения практики (отчет собран в полном объеме); – не везде прослеживается структурированность (четкость, нумерация страниц, подробное оглавление отчета); - правильное выполнение практических задач в области информационных систем и технологий; – не нарушены сроки сдачи

−навыками участия в постановке и проведении экспериментальных исследований; **ПК-24:** −современными инструментальными средствами планирования экспериментов и анализа их результатов.		отчета; – студент демонстрирует достаточную полноту знаний в объеме программы практики, при наличии лишь несущественных неточностей в изложении содержания основных и дополнительных ответов; – владеет необходимой для ответа терминологией; – дает не достаточно исчерпывающие ответы отражающие наличие базовых знаний в области информационных систем и технологий; – владеет методикой проведения экспериментальных исследований; – владеет навыками поиска информации для решения поставленной задачи.
	удовлетворительно	Выставляется практиканту в том случае, если он выполнил весь объем работы, предусмотренный программой практики, продемонстрировав: – соответствие содержания отчета программе прохождения практики (отчет собран в полном объеме); – не везде прослеживается структурированность (четкость, нумерация страниц, подробное оглавление отчета); – в оформлении отчета прослеживается небрежность; - не достаточно правильное выполнение практических задач в области информационных систем и технологий; – нарушены сроки сдачи отчета; – использует специальную терминологию, но могут быть допущены 1-2 ошибки в определении основных понятий, которые студент затрудняется исправить самостоятельно; – способен самостоятельно, но не глубоко, анализировать материал, раскрывает сущность решаемой проблемы только при наводящих вопросах преподавателя;

		– владеет методикой проведения экспериментальных исследований не на достаточном уровне.
	неудовлетворительно	Выставляется практиканту в случае невыполнения им требований, предусмотренных программой практики, а именно: – соответствие содержания отчета программе прохождения практики (отчет собран не в полном объеме); – нарушена структурированность (четкость, нумерация страниц, подробное оглавление отчета); – в оформлении отчета прослеживается небрежность; – индивидуальное задание не раскрыто; – нарушены сроки сдачи отчета; – студент демонстрирует фрагментарные знания в рамках программы практики; – не владеет минимально необходимой терминологией; – допускает грубые логические ошибки, отвечая на вопросы преподавателя, которые не может исправить самостоятельно. – не владеет методикой проведения экспериментальных исследований; – не владеет навыками поиска информации для решения поставленной задачи.

Приложение 2

АННОТАЦИЯ
рабочей программы учебной
(практики по получению первичных профессиональных умений и навыков, в том числе первичных умений и навыков научно-исследовательской деятельности) №2

1. Цель и задачи практики

Цель прохождения практики:

– закрепление и углубление теоретических знаний, полученных при обучении;

– подготовка к осознанному и углубленному изучению общепрофессиональных и специальных дисциплин;

– формирование компетенций, связанных с готовностью к кооперации с коллегами при работе в коллективе; получением навыков организации и управления малыми коллективами; применением различных информационные систем и технологий для решения практических задач в производственной сфере; способностью осуществлять организацию рабочих мест, их техническое оснащение, размещение компьютерного оборудования.

Задачи практики:

– получение первичных профессиональных умений и навыков;

– закрепление теоретических знаний, умений и навыков, полученных обучающимися в процессе аудиторных занятий;

– расширение профессионального кругозора обучающихся;

– изучение опыта работы, соответствующей основным видам будущей профессиональной деятельности обучающихся по направлению 09.03.02 Информационные системы и технологии.

– проведение поиска и систематизации научной информации в определенных областях знания с использованием современных информационных технологий;

– выработка способности и умения анализировать и представлять полученные в ходе исследования результаты в виде законченных отчетов.

2. Структура практики

2.1 Общая трудоемкость практики составляет 108 часов, 3 зачетных единицы, 2 недели.

2.2 Основные разделы (этапы) практики:

1 – Подготовительный этап.

2 – Исследовательский этап.

3 – Обработка и анализ полученной информации.

4 – Подготовка отчета по практике.

3. Планируемые результаты обучения (перечень компетенций)

Процесс прохождения практики направлен на формирование следующих компетенций:

ОК-2 – готовность к кооперации с коллегами, работе в коллективе, знание принципов и методы организации и управления малыми коллективами;

ОПК-1 – владение широкой общей подготовкой (базовыми знаниями) для решения практических задач в области информационных систем и технологий;

ОПК-5 - способность использовать современные компьютерные технологии поиска информации для решения поставленной задачи, критического анализа этой информации и обоснования принятых идей и подходов к решению;

ПК-23 – готовность участвовать в постановке и проведении экспериментальных исследований;

ПК-24 - способностью обосновывать правильность выбранной модели, сопоставляя результаты экспериментальных данных и полученных решений.

4. Вид промежуточной аттестации: зачет с оценкой

Приложение 3

Протокол о дополнениях и изменениях в рабочей программе
на 20___-20___ учебный год

1. В рабочую программу по практике вносятся следующие дополнения:

_____
_____

2. В рабочую программу по практике вносятся следующие изменения:

_____
_____
_____

Протокол заседания кафедры №____ от «___» _____ 20 __ г.,
(разработчик)

Заведующий кафедрой _____ _____
(подпись) *(Ф.И.О.)*

Программа составлена в соответствии с федеральным государственным образовательным стандартом высшего образования по направлению подготовки 09.03.02 Информационные системы и технологии от «12» марта 2015 г. № 219

для набора 2015 года: и учебным планом ФГБОУ ВО «БрГУ» для очной формы обучения от «03» июля 2018 г. № 413.

для набора 2016 года: и учебным планом ФГБОУ ВО «БрГУ» для очной формы обучения от «16» сентября 2016 г. № 622, заочной формы обучения от «16» сентября 2016 г. № 622

для набора 2017 года: и учебным планом ФГБОУ ВО «БрГУ» для очной формы обучения от «06» марта 2017 г. № 125, заочной формы обучения от «06» марта 2017 г. № 125.

для набора 2018 года и учебным планом ФГБОУ ВО «БрГУ» для очной формы обучения от «12» марта 2018 г. № 130, заочной формы обучения от «12» марта 2018 г. № 130.

Программу составил:

Полячкова М.А., ст. преподаватель каф. ИиПМ _____

Рабочая программа рассмотрена и утверждена на заседании кафедры ИиПМ
от «19» декабря 2018 г., протокол № 5

И.о. заведующего кафедрой ИиПМ _____ А.С. Толстиков

СОГЛАСОВАНО:
И.о.заведующий кафедрой ИиПМ _____ А.С. Толстиков

Рабочая программа одобрена методической комиссией ЕНФ
от «20» декабря 2018 г., протокол № 4

Председатель методической комиссии факультета _____ М.А. Варданян

Начальник
учебно-методического управления _____ Г.П. Нежевец

Регистрационный №_____