ФЕДЕРАЛЬННОЕ АГЕНСТВО ПО ОБРАЗОВАНИЮ
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ УПРАЛЕНИЯ И ПРЕДПРИНИМАТЕЛЬСТВА

КАФЕДРА ИНФОРОМАЦИОННОГО МЕНЕДЖМЕНТА

Ломакин В.В.

ПРОГРАММИРОВАНИЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

учебное пособие

Белгород 2010г.

1. Организационно-методический раздел (пояснительная записка)

1.1. Цель курса

Целью курса «Программирование и программное обеспечение информационных технологий» является:

- формирование у студентов теоретических знаний и практических навыков, представляющих собой основу для успешного осуществления проектирования, разработки, реализации алгоритмов и программ при решении практических задач;
- приобретение знаний в области организации рациональной эксплуатации операционных систем и сред различной топологии;
- развитие познавательных интересов, интеллектуальных и творческих способностей в области проектирования и реализации программных средств;
- приобретение знаний для использования прикладных программных средств при решении задач информатизации предприятий.

1.2. Задачи курса

Задачи изучения дисциплины:

- изучить современное состояние уровня и направлений развития системных и прикладных программных средств;
- дать целостное представление об алгоритмизации и программировании как сферах научной и практической деятельности в современных условиях;
- сформировать знания и умения, составляющие основу для описания последовательности решения задач в виде алгоритмов и структур данных;
- обучить актуальным методам программирования;
- дать представление о современном состоянии методов и средств программирования и тенденциях их развития;
- привить навыки решения математических, технических и инженерных задач с помощью алгоритмов и программных средств высокого уровня, как в учебном процессе, так и в практической деятельности.
- изучить основные технологии проектирования и разработки программного обеспечения;
- приобрести практические навыки в использовании сред разработки программ на языке высокого уровня;
- изучить стандарты в области программного обеспечения информационных технологий.

1.3. Место курса в профессиональной подготовке выпускника

Дисциплина «Программирование и программное обеспечение информационных технологий» обеспечивает базовую подготовку для изучения следующих дисциплин учебного плана: «Инструментальные средства моделирования сложных систем», «Бизнес-аналитика», «Базы данных и базы знаний», «Системная интеграция и управление приложениями корпоративной информационной системы». Являясь базовым курсом по освоению приемов составления алгоритмов, технологий проектирования и разработки программ, дисциплина обеспечивает выработку навыков использования программных средств широкого класса для решения практических задач в сфере информационного менеджмента.

1.4. Требования к уровню освоения содержания курса

В результате изучения курса «Программирование и программное обеспечение информационных технологий» студенты должны:

- знать основные понятия и принципы формально-алгоритмического описания объектов и процессов предметной области разработки, иметь четкое представление о технических и программных средствах реализации информационных процессов, структурировании данных и представлении различных видов информации в ЭВМ;
- уметь самостоятельно составлять алгоритмы решения задач вычислительного и логического характера, применять существующие методы проектирования, разрабатывать и реализовывать программы на процедурно-ориентированном языке программирования высокого уровня
- выбирать и применять программное обеспечение корпоративной интеграции при решении задач внедрения информационных технологий на предприятии.

2. Учебно-тематический план

№ п/п	Раздел дисциплины	Всего часов в трудоемкости	Аудиторные занятия (часов)				СР
			В том числе				
			Всего	Л	С	ЛР	
1.	Понятие о программных средствах	2	2	2			
2.	Технология программирования	6	6	2		4	
3.	Типизация и структуризация программных данных	2	2	2			

4.	Линейные операторы языка программирования высокого уровня	2	2	2			
5.	Циклический вычислительный процесс	8	8	2		6	
6.	Ветвление и логические последовательности выбора	11	4	2		2	7
7.	Сложные структуры данных	10	6	2		4	4
8.	Строковый и логический типы данных	6	4	2		2	2
9.	Модульное программирование	8	4	2		2	4
10.	Файлы, работа с файлами на языке высокого уровня	6	4	2		2	2
11.	Формализация, алгоритмы и методы решения задач на ЭВМ	8	2	2			6
12.	Объектно-ориентированное программирование	16	8	4		4	8
13.	Документирование программ	2	2	2			
14.	Среды визуального программирования	16	6	2		4	10
15.	Гипертекстовый язык разметки.	14	4	2		2	10
16.	Системное и прикладное программное обеспечение.	12	4	2		2	8
17.	Программное обеспечение корпоративной интеграции	12	4	2		2	8
Итого		141	2	36		36	69

3. Содержание дисциплины

Тема 1. Понятие о программных средствах.

Понятие о программных средствах, жизненный цикл программных средств. Понятие о трансляции программ. Общая характеристика и классификация современных программных средств. Системные программные средства. Тестовые и диагностические программы. Антивирусные программы.

Тема 2. Технология программирования.

Структурный подход к программированию. Базовые структуры программирования. Элементы графических схем алгоритмов и программ. Языки программирования. Основные современные концепции и подходы к

построению программного обеспечения. Среды визуального программирования. Общая характеристика языка программирования "Паскаль".

Тема 3. Типизация и структуризация программных данных.

Понятие о типах данных. Иерархия типов данных «Паскаля». Разделы программы на языке "Паскаль". Понятие о синтаксической диаграмме.

Тема 4. Линейные операторы языка программирования высокого уровня.

Оператор процедуры, оператор присваивания, составной оператор. Правила вычисления выражений.

Тема 5. Циклический вычислительный процесс.

Операторы циклов While, For, Repeat-Until. Примеры проектирования алгоритмов и составления программ..

Тема 6. Ветвление и логические последовательности выбора.

Оператор условия и правила его вычисления. Понятие о массиве. Оператор варианта (Case).

Тема 7. Сложные структуры данных.

Структурированные типы данных, понятие о записном типе. Оператор присоединения. Примеры использования записных типов. Множественный тип. Списки, деревья, сети, их представление на языке программирования Паскаль.

Тема 8. Строковый и логический типы данных.

Строковый тип и его применение в написании программ. Логический тип данных. Основные соотношения булевой алгебры.

Тема 9. Модульное программирование.

Модульная структура программы. Понятие о подпрограмме. Правила определения процедур и функций. Формальные и фактические параметры. Рекурсия, итерация. Опережающее описание процедур и функций.

Тема 10. Файлы.

Потоки ввода-вывода информации. Понятие о файле. Особенности работы с файлами в языке высокого уровня. Текстовые файлы.

Тема 11. Ссылочные типы данных.

Понятие о статических и динамических переменных. Динамические структуры данных. Пример использования динамических переменных.

Тема 12. Формализация, алгоритмы и методы решения задач на ЭВМ.

Абстракция данных, методы анализа алгоритмов. Эвристика, формализм. Классы алгоритмов, методы частных целей, сортировка и поиск. Формальное определение языков программирования, синтаксис, семантика языка. Формы представления алгоритмов (язык граф-схем алгоритмов, машины Поста, Тьюринга).

Тема 13. Объектно-ориентированное программирование

Понятие об ООП. Основные свойства ООП. Определение объектов. Расширение объектов. Виртуальные правила и полиморфизм. Совместимость типов обьектов. Динамическое распределение памяти под объекты.

Тема 14. Документирование программ

Стандарты в области программного обеспечения. Документирование программных средств. Сопровождение и эксплуатация программных средств. Методика оценки затрат на разработку программных средств.

Тема 15. Гипертекстовый язык разметки.

Принципы HTML и XML-технологий, их применение для решения практических задач в сфере информационного менеджмента.

Тема 16. Системное и прикладное программное обеспечение.

Системное программное обеспечение. Операционные системы. Однопользовательские однозадачные системы (MS-DOS). Однопользовательские многозадачные системы. Многопользовательские системы. Операционная система с открытым исходным кодом. Прикладное программное обеспечение. Командно-файловые процессоры. Общая характеристика прикладных программных средств. Прикладные программные пакеты. Почтовые программы. Web-браузеры. Программы групповой работы.

Тема 17. Программное обеспечение корпоративной интеграции.

Программное обеспечение корпоративной интеграции: корпоративное, промежуточное программное обеспечение и программное обеспечение корпоративной интеграции.

4. Рекомендуемая (основная) литература

1. Немнюгин, С.А. Turbo Pascal: практикум: Учебное пособие для студентов вузов, обучающихся по направлению подготовки дипломированных специалистов "Информатика и вычислительная техника" / Немнюгин С.А.; МО РФ. - СПб.: Питер, 2002 - 253 с.

2. Давыдов, В.Г. Программирование и основы алгоритмизации: учебное пособие/ В.Г. Давыдов.- М.: Высшая школа, 2003.- 448 с.

3 Программирование на языке Паскаль: Учебное пособие/ Под ред. О.Ф. Усковой.- СПб.: Питер, 2003.- 333 с.-(Учебник для вузов)

4 Брусенцева, В.С. Алгоритмизация и программирование на языке Паскаль: Учебное пособие/ В.С. Брусенцева.- 2-е изд., испр. и доп..- Белгород: Изд-во БелГТАСМ, 2000.- 95 с.

5 Харина, Л.В. Турбо Паскаль. Программирование: Учебное пособие/ Л.В. Харина, Е.Н. Заскалина, под ред. Р.А. Андриановой.- Челябинск: ЮУрГУ, 2002.- 77 с.

6 Истомин, Е.П. Программирование на алгоритмических языках высокого уровня: Учебник/ Е.П. Истомин, С.Ю. Неклюдов.- СПб.: Михайлова В.А., 2003.- 718 с.

7 Иванова, Г.С. Объектно-ориентированное программирование: учебник/ Г.С. Иванова, Т.Н. Ничушкина, Е.К. Пугачев.- Изд. 2-е, перераб. и доп..- М.: Изд-во МГТУ им. Н.Э. Баумана, 2003.- 367 с..-(Информатика в техническом университете).

8. Данова Н.С. Электронный учебник по информатике./ Н.С. Данова, В.А. Лихачев // Электронный ресурс. Режим доступа http:\\www.bsu.edu.ru.

5. Рекомендуемая (дополнительная) литература

1. Климова, Л.М. Pascal 7.0. Практическое программирование. Решение типовых задач: Учебное пособие/ Л.М. Климова.- 4-е изд., доп..- М.: КУДИЦ-ОБРАЗ, 2003.- 524 с..-(Библиотека профессионала)
2. Новичков, В.С. Алгоритмизация и программирование на Турбо Паскале: учебное пообие/ В.С. Новичков, Н.И. Парфилова, А.Н. Пылькин.- М.: Горячая линия - Телеком, 2005.- 462 с.
3. Хьюз, Дж. Структурный подход к программированию / Дж. Хьюз, Дж. Мичтом. - М. : Мир, 1980, - 280 с.
4. Йенсен, К. Паскаль. Руководство для пользователя и описание языка/ К. Йенсен, Н.Вирт.- М. : Финансы и статистика, 1982.
5. Шелест, В.Д. Программирование/ В.Д. Шелест.- СПб.: BHV, 2001.- 584 с.

Раздел I. Программирование и разработка прикладного программного обеспечения

В этом разделе рассматривается краткая классификация языков программирования и практически изучается учебный язык програмирования Pascal.

ТЕМА 1. Языки программирования.

Язы́к программи́рования — формальная знаковая система, предназначенная для записи программ. Программа обычно представляет собой некоторый алгоритм в форме, понятной для исполнителя (например, компьютера). Язык программирования определяет набор лексических, синтаксических и семантических правил, используемых при составлении компьютерной программы. Он позволяет программисту точно определить то, на какие события будет реагировать компьютер, как будут храниться и передаваться данные, а также какие именно действия следует выполнять над этими данными при различных обстоятельствах.

Со времени создания первых программируемых машин человечество придумало уже более восьми с половиной тысяч языков программирования. Каждый год их число пополняется новыми. Некоторыми языками умеет пользоваться только небольшое число их собственных разработчиков, другие становятся известны миллионам людей. Профессиональные программисты иногда применяют в своей работе более десятка разнообразных языков программирования.

Создатели языков по-разному толкуют понятие язык программирования. К наиболее распространенным утверждениям, признаваемым большинством разработчиков, относятся следующие:

Функция: язык программирования предназначен для написания компьютерных программ, которые применяются для передачи компьютеру инструкций по выполнению того или иного вычислительного процесса и организации управления отдельными устройствами.

Задача: язык программирования отличается от естественных языков тем, что предназначен для передачи команд и данных от человека компьютеру, в то время, как естественные языки используются для общения людей между собой. В принципе, можно обобщить определение «языков программирования» — это способ передачи команд, приказов, четкого руководства к действию; тогда как человеческие языки служат также для обмена информацией.

Исполнение: язык программирования может использовать специальные конструкции для определения и манипулирования структурами данных и управления процессом вычислений.

Классификация языков программирования

Можно выделить следующие классы языков программирования: Функциональные; Процедурные (Императивные); Стековые; Векторного программирования; Аспектно-ориентированные; Декларативные; Динамические; Прототипные; Объектно-ориентированные; Рефлексивные; Логического программирования; Сценарные (Скриптовые); прочие.

В языках **функционального программирования** основным конструктивным элементом является математическое понятие функции. Существует различия в понимании функции в математике и функции в программировании, в следствии чего нельзя отнести Си-подобные языки к функциональным, использующим менее строгое понятие. Функция в математике не может изменить вызывающее её окружение и запомнить результаты своей работы, а только предоставляет результат вычисления функции. Программирование с использованием математического понятия функции вызывает некоторые трудности, поэтому функциональные языки, в той или иной степени предоставляют и императивные возможности, что ухудшает дизайн программы (например возможность безболезненных дальнейших изменений). Дополнительное отличие от императивных языков программирования заключается в декларативности описаний функций. Тексты программ на функциональных языках программирования описывают «как решить задачу», но не предписывают последовательность действий для решения. Первым, спроектированным функциональным языком стал Лисп. Вариант данного языка широко используется в системе автоматизированного проектирования AutoCAD и называется AutoLISP

В качестве основных свойств функциональных языков программирования обычно рассматриваются следующие:

краткость и простота;
строгая типизация;
модульность;
функции - объекты вычисления;
чистота (отсутствие побочных эффектов);
отложенные вычисления.

Процедурное (императивное) программирование является отражением архитектуры традиционных ЭВМ, которая была предложена фон Нейманом в 40-х годах. Теоретической моделью процедурного программирования служит алгоритмическая система под названием «машина Тьюринга».

Программа на процедурном языке программирования состоит из последовательности операторов (инструкций), задающих процедуру решения задачи. Основным является оператор присваивания, служащий для изменения содержимого областей памяти. Концепция памяти как хранилища

значений, содержимое которого может обновляться операторами программы, является фундаментальной в императивном программировании.

Выполнение программы сводится к последовательному выполнению операторов с целью преобразования исходного состояния памяти, то есть значений исходных данных, в заключительное, то есть в результаты. Таким образом, с точки зрения программиста имеются программа и память, причем первая последовательно обновляет содержимое последней.

Примеры :Ада (язык общего назначения); Бейсик (версии начиная с Quick Basic до появления Visual Basic); Си; КОБОЛ; Фортран; Модула-2; Паскаль; ПЛ/1.

Стековый язык программирования (англ. stack-oriented programming language) — это язык программирования, в котором для передачи параметров используется машинная модель стека. Этому описанию соответствует несколько языков, в первую очередь Forth и PostScript, а также многие ассемблерные языки (использующие эту модель на низком уровне — Java, C#). При использовании стека, в качестве основного канала передачи параметров между словами, элементы языка, естественным образом, образуют фразы (последовательное сцепление). Это свойство сближает данные языки с естественными языками.

Выполнение программы в стековом языке программирования представляет собой операции на одном или нескольких стеках, которые могут иметь различное предназначение. Вследствие этого программные конструкции других языков программирования должны быть изменены, прежде чем они могут быть использованы в стековом языке. Стековые языки программирования используют так называемую «обратную польскую» нотацию (англ. RPN, reverse polish notation) или постфиксную нотацию, в которой аргументы или параметры команды должны быть записаны перед самой командой. Например, в обратной польской нотации операция сложения записывается как «2 3 +», а не «+ 2 3» (префиксная или «польская» нотация) или «2 + 3» (инфиксная нотация). Это позволяет использовать, в полной мере, стековые языки при ограниченных аппаратных ресурсах памяти в контроллерах встроенных систем.

Аспектно-ориентированное программирование (АОП) — парадигма программирования, основанная на идее разделения функциональности, особенно сквозной функциональности, для улучшения разбиения программы на модули.

Методология аспектно-ориентированного программирования была предложена группой инженеров исследовательского центра Xerox PARC под руководством Грегора Кикзалеса (Gregor Kiczales). Ими же был разработан первый, и наиболее успешный до сих пор, контекстно-ориентированный язык программирования AspectJ.

Существующие парадигмы программирования, такие как процедурное программирование и объектно-ориентированное программирование, предоставляют некоторые способы для разделения и выделения функциональности, например, функции, объекты, классы, пакеты, но

некоторую функциональность с помощью предложенных методов невозможно выделить в отдельные сущности. Такую функциональность называют сквозной, так как её реализация разбросана по различным модулям программы. Сквозная функциональность приводит к рассредоточенному и запутанному коду. Запутанным называется такой код, в котором одновременно реализована различная функциональность.

Все языки АОП предоставляют способы для выделения сквозной функциональности в отдельную сущность. Различие между ними заключается в удобстве, безопасности и области применения средств, которые они предоставляют. Наиболее популярный на данный момент язык АОП — AspectJ. Используемые в нем понятия распространились на большинство языков АОП.

Декларати́вные языки́ программи́рования - это языки программирования высокого уровня, в которых программистом не задается пошаговый алгоритм решения задачи ("как" решить задачу), а некоторым образом описывается, "что" требуется получить в качестве результата. Механизм обработки сопоставление по образцу декларативных утверждений уже реализован в устройстве языка. Типичным примером таких языков являются языки логического программирования (языки, основанные на системе правил).

В программах на языках логического программирования соответствующие действия выполняются только при наличии необходимого разрешающего условия.

Наиболее распространённым языком логического программирования является язык Пролог.

Динамический язык позволяет определять типы данных и осуществлять синтаксический анализ и компиляцию «на лету», непосредственно на этапе выполнения. Динамические языки больше подходят для быстрой разработки приложений.

К динамическим языкам относятся: Python, PHP, Ruby, JavaScript.

Прототипное программирование - стиль объектно-ориентированного программирования, при котором отсутствует понятие класса, а повторное использование (наследование) производится путём клонирования существующего экземпляра объекта — прототипа.

Каноническим примером прототип-ориентированного языка является язык Self. В дальнейшем этот стиль программирования начал обретать популярность и был положен в основу таких языков программирования, как JavaScript, REBOL и др.

Объектно-ориентированный язык программирования (ОО-язык) — язык, построенный на принципах объектно-ориентированного программирования.

В основе концепции объектно-ориентированного программирования лежит понятие объекта - некоей субстанции,которая объединяет в себе поля (данные) и методы (выполняемые объектом действия).

Например, объект "человек" может иметь поля "имя","фамилия" и иметь методы "есть" и "спать".Соответственно, мы можем использовать в программе операторы Человек.Имя:="Иван" и Человек.Есть(пища).

Особенности. В современных ОО языках используются методы:

Наследование. Создание нового класса объектов путем добавления новых элементов (методов). В данный момент ОО языки позволяют выполнять множественное наследование, т. е. объединять в одном классе возможности нескольких других классов.

Инкапсуляция. Сокрытие деталей реализации, которое (при грамотном использовании) позволяет вносить изменения в части программы безболезненно для других её частей, что существенно упрощает сопровождение и модификацию ПО.

Полиморфизм. При полиморфизме некоторые части (методы) родительского класса заменяются новыми, реализующими специфические для данного потомка действия. Таким образом, интерфейс классов остаётся прежним, а реализация методов с одинаковым названием и набором параметров различается. С понятием «Полиморфизм» тесно связано понятие «Позднего связывания».

Типизация. Позволяет устранить многие ошибки на момент компиляции, операции проводятся только над объектами подходящего типа.

Неполный список объектно-ориентированных языков программирования: Java; C#; C++; Objective-C; Object Pascal (Delphi); VB.NET; Perl; PHP; Python; JavaScript; Ruby; Smalltalk; Ada. Кроме ОО-языков общего назначения существуют и узкоспециализированные ОО-языки, например 1С («один-эс»).

Логи́ческое программи́рование - парадигма программирования, а также раздел дискретной математики, изучающий методы и возможности этой парадигмы, основанной на выводе новых фактов из данных фактов согласно заданным логическим правилам. Логическое программирование основано на теории математической логики. Самым известным языком логического программирования является Prolog, являющийся по своей сути универсальной машиной вывода, работающей в предположении замкнутости мира фактов.

Первым языком логического программирования был язык Planner, в котором была заложена возможность автоматического вывода результата из данных и заданных правил перебора вариантов (совокупность которых называлась планом). Затем был разработан язык Prolog, который не требовал плана перебора вариантов и был, в этом смысле, упрощением языка Planner.

Скри́птовый язы́к (англ. scripting language, в русскоязычной литературе принято название язык сценариев) — язык программирования, разработанный для записи «сценариев», последовательностей операций, которые пользователь может выполнять на компьютере. Простые скриптовые языки раньше часто называли языками пакетной обработки (batch languages или job control languages). Сценарии обычно интерпретируются, а не

компилируются (хотя часто сценарии компилируются каждый раз перед запуском).

В прикладной программе, сценарий (скрипт) — это программа, которая автоматизирует некоторую задачу, которую без сценария пользователь делал бы вручную, используя интерфейс программы.

1.1. Среды визуального программирования

Визуальное программирование - способ создания приложений путём манипулирования графическими объектами вместо написания программных кодов в текстовом виде.

Языки визуального программирования могут быть дополнительно классифицированы в зависимости от типа и степени визуального выражения, на следующие типы:

- языки на основе объектов, когда визуальная среда программирования предоставляет графические или символьные элементы, которыми можно манипулировать интерактивным образом в соответствии с некоторыми правилами;

- языки на основе форм, когда программирование осуществляется помещением на специальные формы объектов и настройкой их свойств и поведения. Примеры: Delphi и C++ Builder фирмы Borland.

- языки схем, основанные на идее «фигур и линий», где фигуры (прямоугольники, овалы и т. п.) рассматриваются как субъекты и соединяются линиями (стрелками, дугами и др.), которые представляют собой отношения. Пример: UML.

(Интегрированная) среда разработки программного обеспечения (англ. IDE, Integrated development environment) — система программных средств, используемая программистами для разработки программного обеспечения.

Обычно среда разработки включает в себя текстовый редактор, компилятор и/или интерпретатор, средства автоматизации сборки и отладчик. Иногда также содержит средства для интеграции с системами управления версиями и разнообразные инструменты для упрощения конструирования графического интерфейса пользователя. Многие современные среды разработки также включают браузер классов, инспектор объектов и диаграмму иерархии классов — для использования при объектно-ориентированной разработке ПО.

Примеры сред разработки - Turbo Pascal, Borland C++, Borland Delphi, QNX Momentics IDE, и др.

Частный случай ИСР — среды визуальной разработки, которые включают в себя возможность визуального редактирования интерфейса программы.

Среда визуальной разработки — среда разработки программного обеспечения, в которой наиболее распространенные блоки программного кода представлены в виде графических объектов. Применяются в основном

для создания прикладных программ и разработки графического интерфейса пользователя (GUI).

Преимущества: быстрота разработки, лёгкость освоения, стандартизация внешнего вида программ.

Недостатки: как правило, привязка к конкретной среде разработки связаное с проблематичностью перехода на другую среду разработки; затруднённое использование нестандартных компонентов; наличие недокументированных особенностей компонент.

Следует учитывать что некоторые визуальные среды разработки имеют собственный формат хранения проекта и при переходе на другую среду может возникнуть непереносимость свойств проекта и некоторых частей проекта, таких, как собственные библиотеки используемой среды разработки.

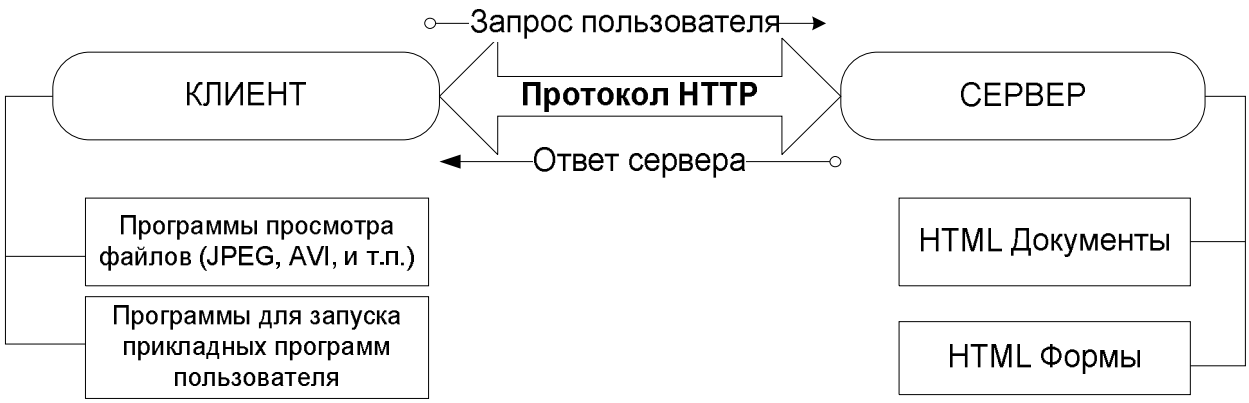
Так же следует учитывать некоторые изменения которые могут вносится в язык программирования, конкретный пример - среда разработки Delphi (по сути это уже новый язык программирования). Среду разработки как и язык программирования следует выбирать на этапе проектирования ПО. Правильно спроектированное ПО должно учитывать развитие и внедрение новых технологий, поэтому перенос разработки такого ПО в другую среду разработки не должен представлять трудностей.

1.2. Гипертекстовый язык разметки HTML

В марте 1989 г. Тим Бернерс-Ли (Tim Berners-Lee) из CERN предложил руководству этого международного европейского научного центра концепцию новой распределенной информационной системы, которую назвал World Wide Web. Свои соображения он изложил в проекте "Гипертекст для ЦЕРН". В 1990 году эти предложения были приняты и проект стартовал. Так началось развитие одной из наиболее популярных современных информационных технологий Internet.

Архитектура WWW-технологии

WWW построена по схеме "клиент-сервер". На рисунке показано, как разделены функции в этой схеме.



Программа-клиент выполняет функции интерфейса пользователя и обеспечивает доступ практически ко всем информационным ресурсам Internet. Фактически, клиент - это интерпретатор HTML. И как типичный интерпретатор, клиент в зависимости от команд (разметки) выполняет различные функции. В круг этих функций входит не только размещение текста на экране, но обмен информацией с сервером по мере анализа полученного HTML-текста, что наиболее наглядно происходит при отображении встроенных в текст графических образов. При анализе URL-спецификации или по командам сервера клиент запускает дополнительные внешние программы для работы с документами в форматах, отличных от HTML, например GIF, JPEG, MPEG и т. п.

База данных HTML-документов -- это часть файловой системы, которая содержит текстовые файлы в формате HTML и связанные с ними графику и другие ресурсы. Особое внимание хотелось бы обратить на документы, содержащие элементы экранных форм. Эти документы реально обеспечивают доступ к внешнему программному обеспечению.

Основные компоненты технологии World Wide Web

Основными компонентами WWW являются:

- язык гипертекстовой разметки документов HTML (HyperText Markup Lan-guage);

- универсальный способ адресации ресурсов в сети URL (Universal Resource Locator - универсальная форма адресации информационных ресурсов);

- протокол обмена гипертекстовой информацией HTTP (HyperText Transfer Protocol).

- универсальный интерфейс шлюзов CGI (Common Gateway Interface).

Самым простым способом создания любого документа является его набивка в текстовом редакторе. Обычно гипертекстовые системы имеют специальные программные средства построения гипертекстовых связей. В HTML гипертекстовые ссылки встроены в тело документа и хранятся как его часть. В WWW документы -- это обычные ASCII- файлы.

Общая структура документа

Документ HTML представляет из себя набор вложенных элементов. Элементы представляют из себя контейнеры, в которых размещаются: текст ; графика; гипертекстовые ссылки; инструкции управления отображением.

Каждый контейнер начинается последовательностью типа: < имя_элемента список_атрибутов > и заканчивается последовательностью типа: < /имя_элемента >.

Первая последовательность называется тагом начала элемента, а вторая последовательность называется тагом конца элемента. Инструкции управления изображением также являются тагами - тагами начала. Исторически сложилось так, что ряд элементов потеряли в процессе развития языка свои таги конца и, таким образом, превратились в инструкции управления отображением. Сам документ является также элементом, который может иметь две формы: форму обычного документа; форму фрейма

Обычный документ представляет из себя два контейнера HEAD и BODY:

< HTML >
< HEAD >
.....
< /HEAD >
< BODY атрибуты_тела_документа >
.....
< /BODY >
< /HTML >

Управление отображением

Здесь сведены в таблицу наиболее часто используемые элементы.

Форма записи элемента	Назначение
< P >	Начать новый параграф
< BR >	Принудительный перевод строки
< HR >	Горизонтальная черта
Списки	
< UL > ... < /UL >	Ненумерованный список
< OL > ... < /OL >	Нумерованный список
< DL > ... < /DL >	Список определений
< LI >	Элемент нумереванного или ненумерованного списка
< DT >	Термин, определяемый в списке определений
< DD >	определение термина в списке определений
Выделение текста	
< B &gt...< /B >	Жирный шрифт
< I > ... < /I >	Курсив
< U > ... < /U >	Подчеркивание
< PRE > ... < /PRE >	Отмена форматирования
< FONT SIZE=атрибут_размера COLOR=атрибут_цвета > ... < /FONT >	Управление шрифтом
Заголовки	
< Hx > ... < /Hx >	Уровень заголовка. Символ "x" принемает значение от 1-7

Встроенная графика

Встраивание графики в текст происходит при помощи использования тага IMG:

<IMG SRC=адрес_картинки ALIGN=атрибут_выравнивания BORDER=ширина_рамки ISMAP USEMAP=имя_стека_ссылок HSPACE=горизонтальный_отступ VSPACE=вертикальный отступ >

Атрибуты тага IMG можно свести в следующую таблицу:

Атрибуты		
Атрибут	Значения	Действие
ALIGN	LEFT	Значение LEFT позволяет организовать обтекание текста справа от картинки
	RIGHT	Обтекание текста слева от картинки
	TOP, BOTTOM, CENTER	Выравнивают картинку относительно текущей строки текста.
BORDER		Значение этого атрибута определяет толщину рамки вокруг

	картинки
SRC	Задает адрес файла картинки
ISMAP	Картинка - стек гипертекстовых ссылок программы imagemap
USEMAP	Стек гипертекстовых ссылок, который не требует использование программы imagemap
HSPACE	Горизонтальный зазор между картинкой и техтом, ее обтекающим
VSPACE	Вертикальный зазор между картинкой и текстом

Гипертекстовые ссылки

Гипертекстовая ссылка имеет вид:

< A HREF=адрес_ресурса TARGET=имя_окна > идентификатор ссылки < /A >

Представленная выше ссылка - это обычная текстовая гипертекстовая ссылка, но бывают еще стеки гипертекстовых графических ссылок. В этом случае для навигации используется графическая картинка:

< A HREF=адрес_ресурса TARGET=имя_окна > < IMG SRC=картинка ISMAP USEMAP=карта > идентификатор ссылки < /A >

В случае атрибута ISMAP используется программа imagemap, а в случае USEMAP используется карта ссылок описанная в елементе MAP. Элемент MAP имеет вид:

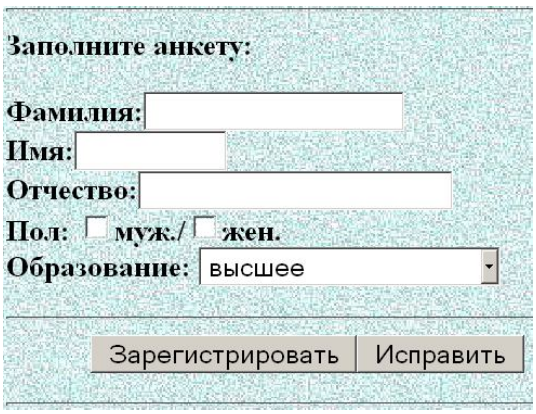
< MAP NAME=map > < AREA SHAPE="RECT" COORDS="x1,y1,x2,y2" HREF=ссылка > .....< /MAP >

Формы

Формы предназначены для передачи параметров и данным прикладным программам. Синтаксис формы имеет следующий вид:

< FORM METHOD=метод ACTION=адрес_программы > элементы формы < /FORM > Элементы формы -- это поля ввода INPUT, кнопки, меню и области ввода. Ниже приведен пример формы.

Вид формы в окне навигатора



HTML- код формы

```
<HR> <FORM METHOD=POST> Заполните анкету:
<P>
Фамилия:<INPUT NAME=fam SIZE=20
MAXLENGTH=20><BR>
Имя:<INPUT NAME=name SIZE=10
MAXLENGHT=10><BR>
Отчество:<INPUT NAME=f_name SIZE=25
MAXLENGTH=25><BR>
Пол: <INPUT NAME=sex1
TYPE=CHECKBOX>муж./<INPUT NAME=sex2
TYPE=CHECKBOX>жен.<BR>
Образование:
<SELECT NAME=edu>
<OPTION SELECTED>высшее
<OPTION>среднее специальное
<OPTION>среднее
<OPTION>начальное
</SELECT> <HR>
<CENTER><INPUT NAME=submit
VALUE=Зарегистрировать TYPE=submit><INPUT
NAME=reset TYPE=reset
```

VALUE=Исправить></CENTER>
</FORM> <HR>

Источники:
1. Храмцов П. Лабиринт Internet. Электронинформ, 1996. Режим доступа: http://lib.ru/LABIRINT/content.htm

2. Технология программирования

Развитие ПО ЭВМ привело к необходимости соблюдения определенного порядка программирования - это необходимо для обеспечения читаемости программ с целью их использования другими программистами или при написании программ коллективом программистов.

2.1. Базовые структуры программирования.

Существуют несколько основных технологий программирования. Одной из самых распространенных является структурный подход к программированию.

Структурный подход

Структурный подход в разработке программ состоит из трех частей:

1) Нисходящая разработка (метод последовательных уточнений)При нисходящей разработке - проектирование, и непосредственно программирование ведутся сверху вниз. Программирование сверху вниз приводит к тому, что программа всегда находится в рабочем состоянии. Написание программы всегда начинается с основной программы, после чего задача программно разбивается на подзадачи, определяя конкретные подпрограммы.

При таком подходе в процессе реализации программы остаются нереализованные части, вместо них вставляются так называемые программные «заглушки» В простейшем случае это оператор вывода или сформированная вручную структура, являющаяся результатом. Практика программирования показала, что написание программ по принципу сверху вниз быстрее и эффективнее позволяет получить результат. Концепция большинства языков программирования ориентирована именно на нисходящую разработку программ.

2) Структурное кодирование (программирование) предполагает запись алгоритмов и выражение логической структуры программы с помощью комбинаций трех базовых структур (см. рис.).

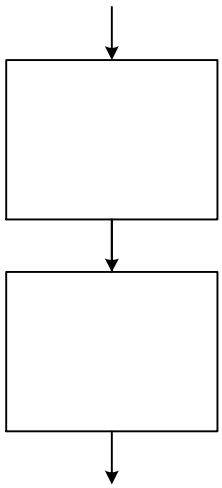
Основную часть циклов, применяемых в программировании составляют циклы «до» и «пока». Использовать данные структуры можно в зависимости от языка программирования писать программы без оператора безусловного перехода(GO TO)

3) Сквозной структурный контроль. Важным результатом использования базовых структур, является то, что поток управления в программах будет направлен сверху вниз – это облегчает их чтение. Однако, в процессе написания сложных программ, особенно несколькими программистами, возникает необходимость состыковки частей программ.

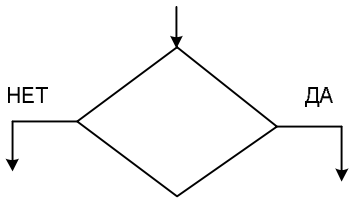
Основная идея сквозного структурного контроля регулярные встречи исполнителей с целью взаимной проверки работы, как на стадии разработки, так и на стадии программирования.

Структурный контроль необходим, для того чтобы обнаружить и исправить ошибки как можно раньше, пока их стоимость минимальна, а последствия наименее значительны.

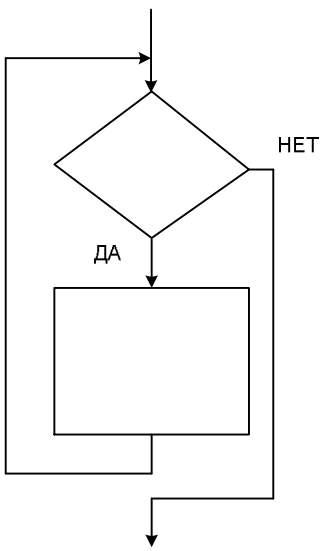
Следование



Развилка

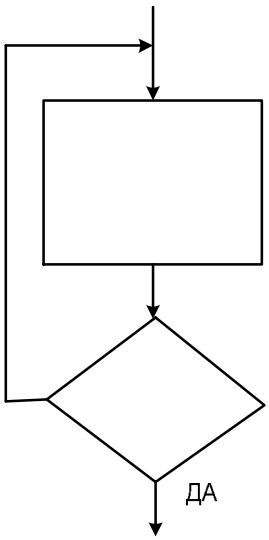


Цикл «Пока»



Тело цикла выполняется, пока условие истинно, завершается – когда ложно.

Цикл «До»



Цикл выполняется до тех пор, когда условие становится истинным

Рис. Базовые структуры программирования.

Выделяются отдельные элементы графических схем алгоритмов и программ (определяется государсвенными стандартами ГОСТ 19.003-80, ГОСТ 19.701-90):

1) Процесс;
2) Решение;
3) Предопределенный процесс;
4) Ввод-вывод;
5) Документ;
6) Пуск-остановка и проч.

На рисунке представлены элементы блок-схем алгоритмов с рекомендованными пропорциями в соответствии со стандартом.

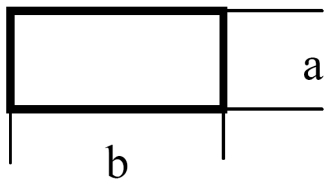
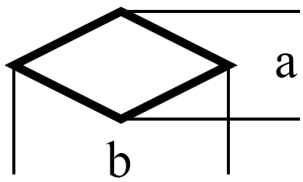
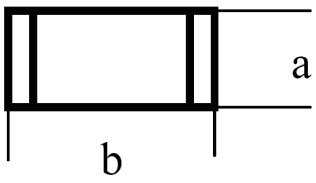
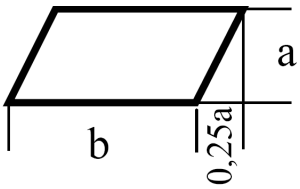
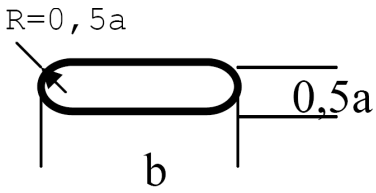
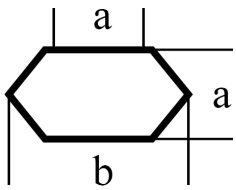
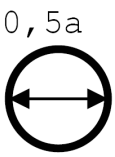
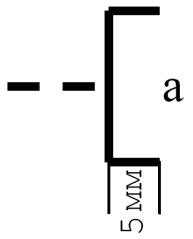
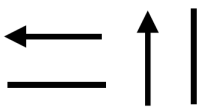
№	Название	Обозначение	№	Название	Обозначение
1	Процесс		6	Решение (проверка условия)	
2	Предопределенный процесс (вызов процедуры, модуля)		7	Ввод – вывод	
3	Начало, конец алгортма (пуск/остановка)		8	Модификация (блок изменения переменной цикла)	
4	Соединитель		9	Комментарий	
5	Линии перехода		10	Пропорции сторон	a = 10; 15; 20 . . . мм; b = 1,5a или 2a

Рис. Условные графические обозначения для составления блок-схемы (ГОСТ 19.701-90).

Блоки схемы имеют скозную нумерацию сверху – вниз и слева – направо. За основные направления линий переходов приняты направления слева – направо и сверху – вниз.

Операция присваивания

Чаще всего обозначается символами ‘=’ или ‘:=’. Выполняется справа налево, значение выражения находится справа вычисляется, затем присваивается переменной находящейся слева, например,

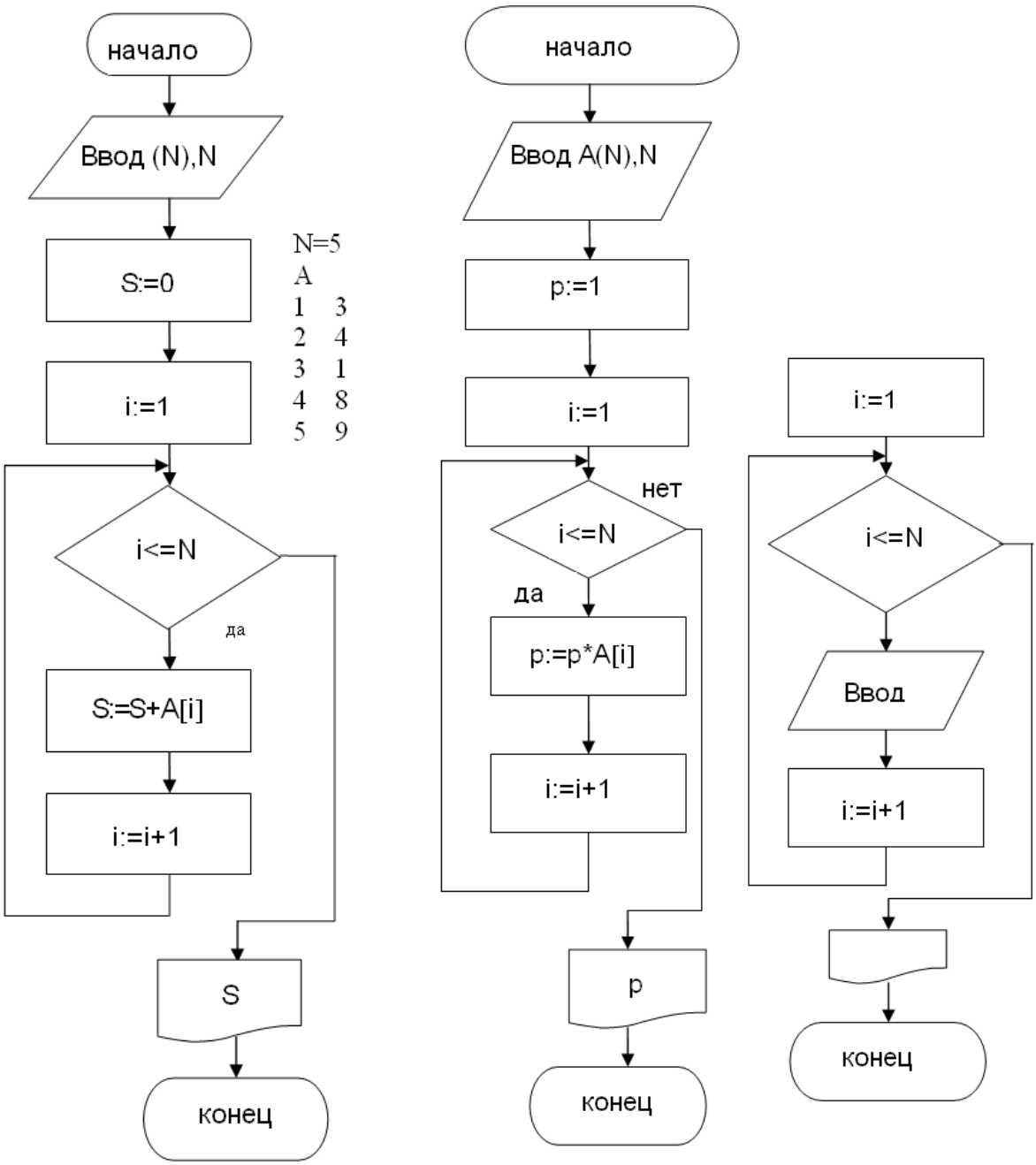
a:=5, i:=a+1, i:=i+1, i:=7.

Примеры алгоритмов суммирования, нахождения произведения, максимума и минимума.

Рассмотрим примеры указанных алгоритмов.

Нахождение суммы элементов массива

Произведение элементов массива

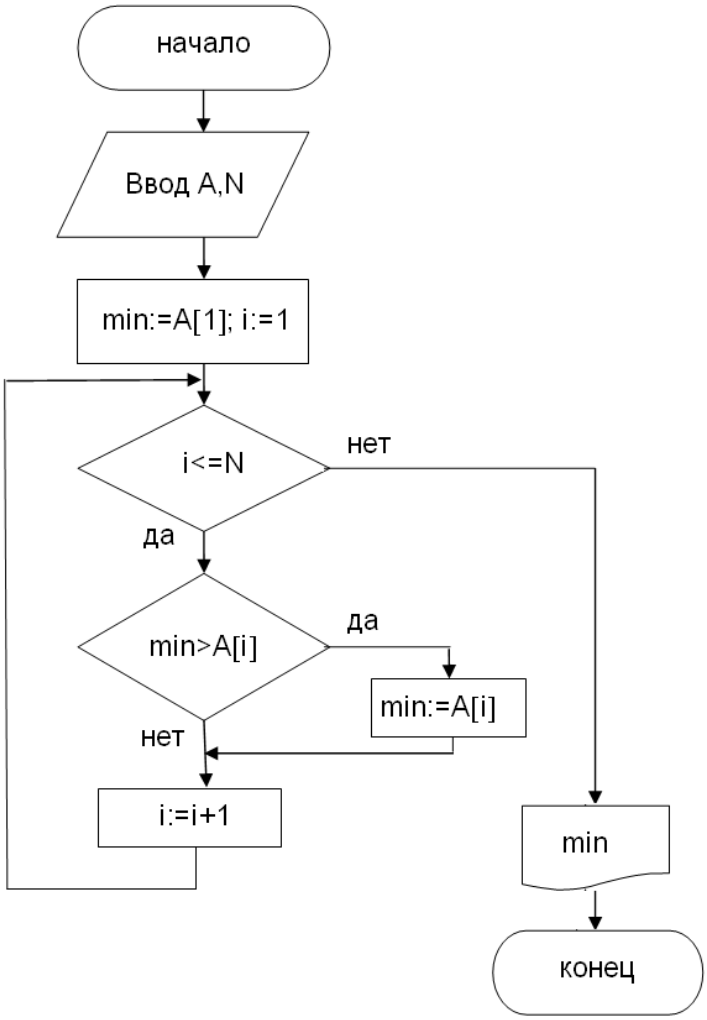


Замечание: В блок-схемах не отражено, что элементы массива вводятся так же с помощью цикла и что максимальное число элементов массива в языке программирования фиксировано.

Пример. Определение максимума, минимума.

Найти минимальный элемент массива. Блок-схема алгоритма может выглядеть так:

Нахождение суммы элементов массива



2.2. Основы программирования на языках высокого уровня на примере языка программирования Pascal

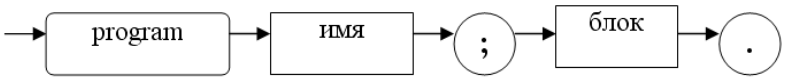
Был создан в 1968 году Никлаусом Виртом, как язык обучения программированию систематическим образом.

Первый транслятор появился в 1970 году. Существуют стандарты ISO,ANSI. Существует много реализаций языка Pascal, наиболее известна в настоящее время Delphi, основанная на языке Pascal, при этом язык в ней значительно расширен.

2.3. Структура программы

Pascal является структурированным языком, в котором каждый объект имеет свой тип, а написание программы, структурно подчинено определенным правилам.

Синтаксическая диаграмма (диаграма Н.Вирта) для программы.



Каждый путь по синтаксической диаграмме определяет синтаксически корректную программу. Каждый прямоугольник содержат имена, указывающие на диаграмму используемую для расшифровки соответствующего «значения». Терминальные символы, т.е. те, которые реально используются при написании программы, помещаются в окружности или овалы.

Для для имени

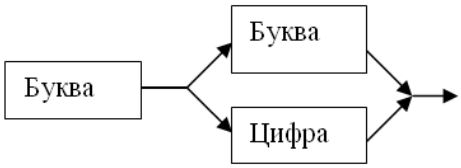
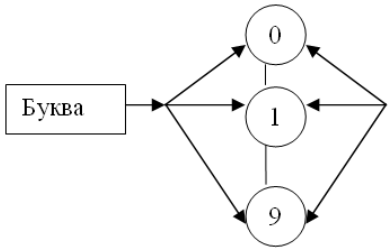


Диаграмма для цифр



Пример имени программы: Program1

Имя - последовательность букв и цифр, начинающаяся с буквы (Для букв от A до Z и маленьких, русских букв нет)

В языке Pascal регистр букв при написании имен не имеет значения, однако, крайне желательно придерживаться **следующих правил:**

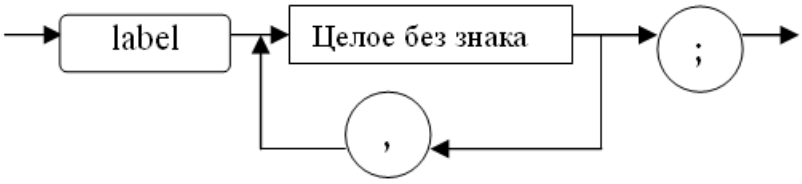
Большими буквами именуются константы, типы, все остальное маленькими буквами.

Прямоугольный блок включает в себя следующие разделы, которые обязательно должны следовать в указанном порядке:

1-раздел определения меток;
2-раздел определения констант;
3-раздел определения типов;
4-раздел описания переменных;
5- раздел описания процедур и функций;
6- раздел операторов.

Раздел определения меток.

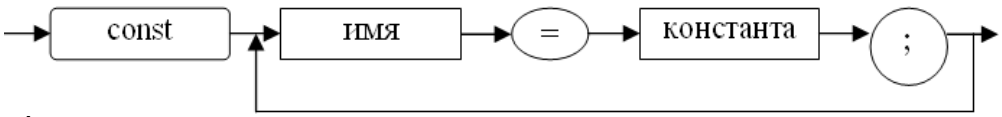
Метка-целое без знака. В реализации в Pascal допускается имя, когда метка помечает оператор, после нее ставится двоеточие. Раздел выглядит следующим образом:



Структурное программирование подразумевает написание программ без использования меток, считается что это улучшает читаемость программ. Метки следует использовать в исключительных случаях.

Раздел определения констант(const)

Вводит имя, как синоним некоторой константы. Вид раздела:

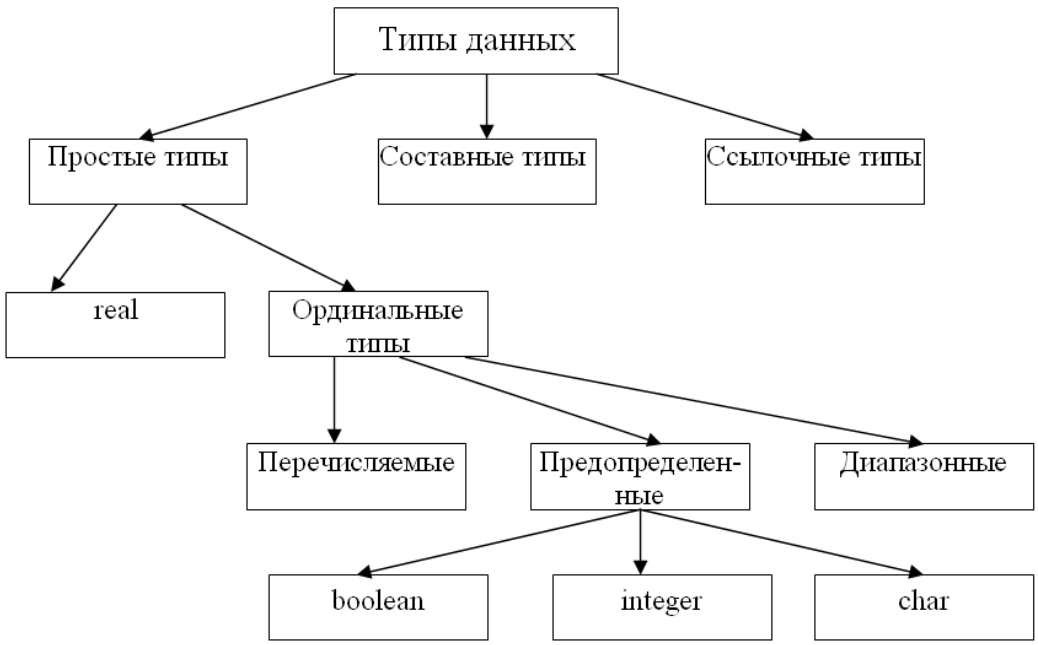



```
Const Mintemp= -273;
            Light speed=300000;
            BUKVA='A'..'Z';

type letter = 'A' … 'Z';
            var x:letter; ch:char;
            begin
```

Раздел определения типа

Иерархия типов языка Pascal представлена на рис.



Простые типы не обладают внутренней структурой.

Тип real-вещественные значения, имеющие порядок и мантиссу.

Тип Boolean-определяет два значения false и true.

Тип char- определяет символьные значения в соответствии с таблицей ASCII.

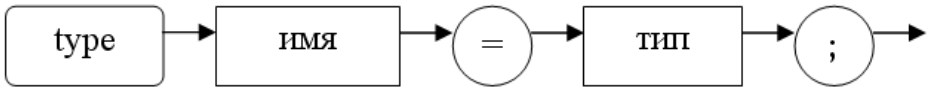
Тип integer – целые значения, точные, но в ограниченном диапазоне(-32768 до 32767).

В среде Delphi типы real и integer имеют двойную точность, одинарную и сокращенный формат.

Ординальные типы еще называют скалярными и порядковыми.

Раздел типов

Вид раздела:

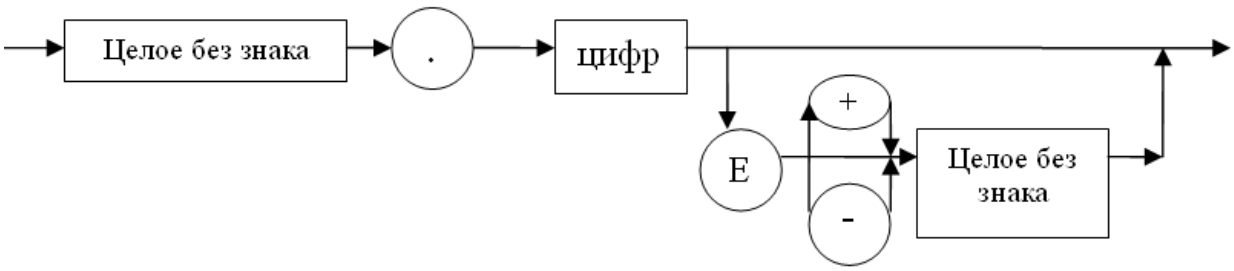


Пример:

```
Type
        RANGE =     -100..200;
        Smallchar   =     'a'.. 'z';
```

Числа

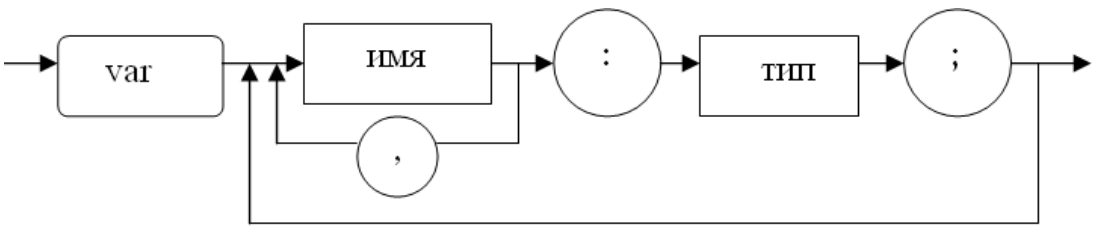
Вещественные числа могут представляться в формате с фиксированной и плавающей точкой. Диаграмма описания числа без знака



Например:
15.3578
0.03
15.351E-15=15.351*10-15
1.1534E+5

Раздел описания переменной

Вид раздела:



Пример:

```
var     b, c        :     real;
        ount, i     :     integer;
        flag        :     Boolean;
        ch          :     char;
```

Каждая переменная, используемая в программе, должна быть описана в разделе описания переменных.

Раздел описания процедур и функций

При решении задач во многих случаях программу разбивают на несколько блоков, каждый блок воспринимается как отдельная программа. Такое разбиение, на относительно небольшие блоки дает в результате понятную структуру программы.

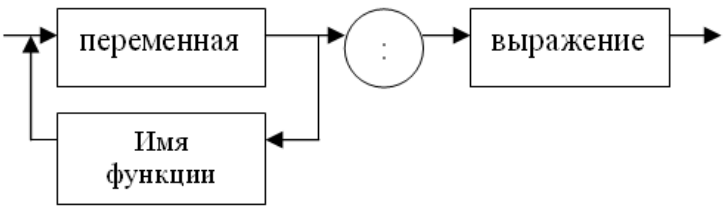
Блоки еще называют подпрограммами. В Pascal разбиение на блоки осуществляется с помощью процедур и функций.

Описание процедуры или функции имеет вид, аналогичный виду программы, т.е. заголовок, за которым идет блок.

Раздел операторов

begin- начало раздела операторов
end.- конец раздела операторов

Оператор присваивания



Не следует путать операцию присваивания (:=) с операцией равенства (=).

Выражение состоит из операции и операндов. Операндом может быть константа, переменная, или обозначение функции.

Вид выражения однозначно определяет правила его вычисления, т.е. действия выполняются слева на право, с соблюдением следующего старшинства в порядке убывания:

not (логическое отрицание);

*, / , div, mod, and. (div-целочисленное деление, значение не округляется: например,5 div 3=1 5/3=1.6…

mod-остаток от деления: например, 5mod3=2);

+, -, or

=,< >,>,<,<=,>=, in(оператор вхождения во множество)

Существует понятие предопределения функций. Эти функции реализованые заранее в языке программирования.

Например: trunk(R), R- вещественное значение, результат - целая часть.

Например: trunk(2,78)=2

round(R)-округление до целого, например, round(5,76)=6

Пусть x - любое целое или вещественное выражение, тогда результат abs(sqr(x)), имеет такой же тип что и x, результат основной функции real:

abs(x)=|x|;

sqr(x)=x^2;

sin(x),cos(x);

exp(x)=e^x;

sqrt(x)=√x>=0

Например:a:=sqrt(x).

Оператор процедуры

Активизировать процедуру, представляющую собой подпрограмму, можно осуществляя определенную последовательность действий.

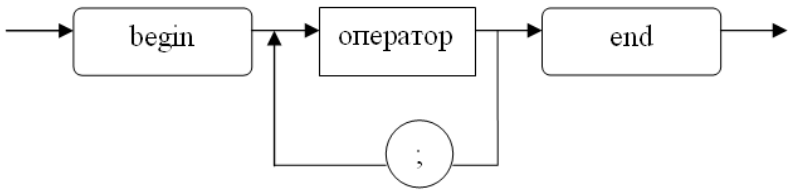
Для консольного ввода – вывода используется read, readln/write, writeln.

Пример программы с консольным вводом и выводом на экран:

```
Program Summa;
var c1, c2, rez    :     real;
begin
            readln(c1,c2);
            rez:=c1+c2;
            Writeln('результат=',rez);
end.
```

Составной оператор

Предусматривает выполнение входящих в его состав операторов компонент, в порядке их написания. Служебные слова begin и and играют роль операторных скобок.



Раздел операторов представляет собой один составной оператор.

Комментарии начинаются с символа { или (*, заканчиваются } или *) , но не внутри строки символов.

Комментарий может содержать любые символы в программе условно заменен на пробел. Комментарий вводят с целью разъяснения сути того, что происходит в программе. Комментарий должен отражать смысл того, что написано: например,

i:=i+1; {увеличение на единицу} (пример неверного комментария)

{число человек}

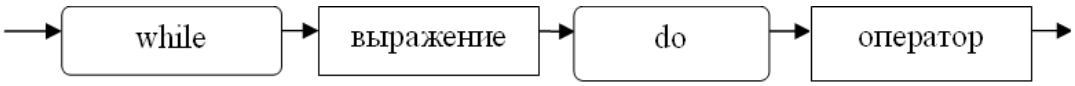
Пустой оператор

В Pascal (;) используется как разделитель между операторами, т.е. точка с запятой(;) не входит в оператор и не заканчивает его. В примере программы (;) между последним оператором и ключевым словом end, означает, что между ними находится пустой оператор, т.е не выполняется никаких действий.

2.4. Оператор повторения(цикла)

Циклы предусматривают выполнение операторов входящих в них несколько раз.

Оператор цикла с предусловием (типа «пока»)



While-ключевое слово.

Выражение, управляющее повторениями должно быть логического типа. Если оно истинно(дает значение true)-оператор выполняется, если ложно(значение false)-выполнение всего оператора цикла завершается (логическое если можно сказать да или нет).

Например:

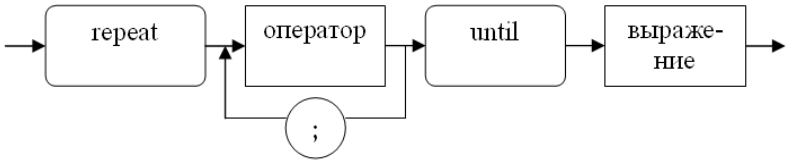
```
-i+1
+a=1        +a - true
-a+b
+a<1
```

Пример:

```
Program while example;  (* Вычисление N-частичной суммы гармонического ряда
H(N)=1+1/2+1/3+1/4+…+1/N *)
 var   N     :     integer;
       H     :     real;
begin
       Readln (N);
       H: =0;
       While N>0 do
             begin
                   H: =H+1/N;
                   N: =N-1;
             end;
       Writeln ('Сумма=', H)
end.
```

Причина изменения элементов ряда с самого маленького числа, в том что существует понятие разрядной сетки ЭВМ. Число хранится в виде мантисса/порядок. Если начать измерение с 1, то результат может получиться достаточно большим и элемент 1/N или близкие к нему не вместятся в мантиссу.

Оператор цикла с постусловием (типа «до»)

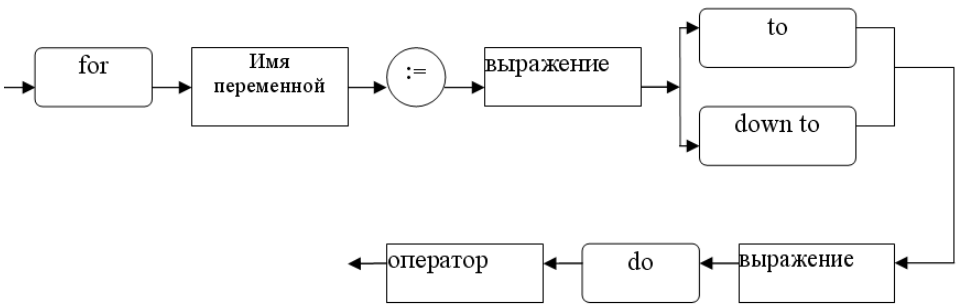


Пример:

```
Program repeat example; (*Вычисление N- частичной суммы гармонического ряда
H(N)=1+1/2+1/3+1/4+…+1/N*)
var   N     :     integer;
      H     :     real;
begin
      Readln (N);  H: =0;
      repeat
            H: =H+1/N;
            N: =N-1;
      until
      N: =0;
      Writeln (‘Сумма=’, H)
end.
```

Выражение должно быть логического типа. Оно управляет повторением. Цикл выполняется до тех пор, пока выражение становится истинным. Слова begin и end внутри цикла с постусловием использовать не нужно. Роль операторных скобок выполняют команды repeat,until.

Оператор цикла с параметром



Переменная, следующая за ключевым словом for, называется управляющей переменной(параметром цикла). Параметр цикла должен относится к ординальному типу и быть описан в том же блоке, где появляется сам оператор цикла.

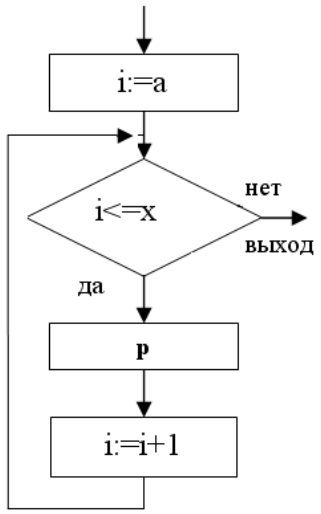
Начальное и конечное значения вычисляются единожды. В операторе-компоненте цикла параметр не должен изменяться. При нормальном выходе из цикла, управляющая переменная считается неопределенной.

Пример:

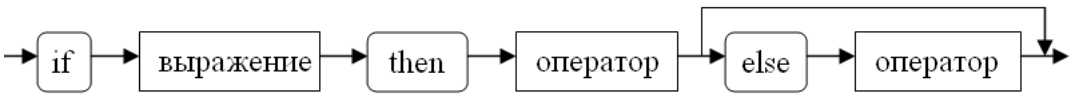
Блок-схема реализации оператора цикла
for i:=a to x do p

```
Program forExample;
var
      i, n  :     integer;
      h     :     real;
```

```
begin
 read(n);
 h:=0;
 for i:=n down to 1
 h:=h+1/I;
 writeln('сумма=', h);
end.
```



2.5. Условный оператор



Выражение должно относится к типу boolean, после then (;), понимается как пустой оператор, пере else(;) не ставится.

Пример:

If A<B then; A:=3

Условный оператор управляет пустым оператором, а присваивание выполняется всегда.

Синтаксическая двусмысленность порожденная конструкцией:

if выражение1 then if выражение2 then оператор1 else оператор2 . Разрешается путем ее интерпретации, как эквивалентной конструкции, чтобы решить - надо поставить begin.

Пример:

if выражение1 then
begin
if выражение2 then оператор1
else оператор2
end;

Пусть имеется задача найти значение функции,имеющей вид:

$$Z = \begin{cases} |x|, & x < 1; \\ e^x, & 1 \le x \le 5; \\ \sqrt{x}, & x \ge 5. \end{cases}$$

Тогда программа для ее расчета может быть такой:

```
Program Fune;
      var   x, z: real;
begin
      readln(x);
```

```
      if x<1
            then x:=abs(x)
      else
            if x>=5
                  then z:= SQRT(x)
            else z:=exp(x)
      writeln('значение функции =', z);
end.
```

2.6. Диапазонные типы

Этот тип можно определить указав некоторый диапазон значений из любого другого предварительно определенного ординального типа. Этот тип называется базовым.

Диапазон определяется указанием наименьшего и наибольшего постоянного значения, входящего в диапазон.

Пример:

```
var
      A: 1..10;
      B: 0..40;          базовый тип для A,B,C- integer
      C: 30..100;
```

2.7. Массивы

Любой массив состоит из фиксированного числа компонент. Это число определяется при описании массива. Все компоненты относятся к одному типу – его называют типом компонент. Каждая компонента может быть явно обозначена с помощью имени массива за которым в квадратных скобках следует индекс. Индексы можно вычислять. Их тип называется типом индексов, массивный тип относится к основным типам.

Схема определения нового массивного типа:

```
type
A=array [T1] of [T2];
```

A – имя массивного типа,

T1 – тип индекса(обязательно ординальный, кроме integer);

T2 – тип компонент(может быть типом массива)

array(массив) of (что-то)

Пример описания массивов:

```
type
      mas = array [0..32767] of integer;
      massive = array [1..100] of real;
var
      A     :     mas;          описание переменных
      B     :     massive;            данного типа
```

В соответствии с этим описанием, создаются переменные A и B:

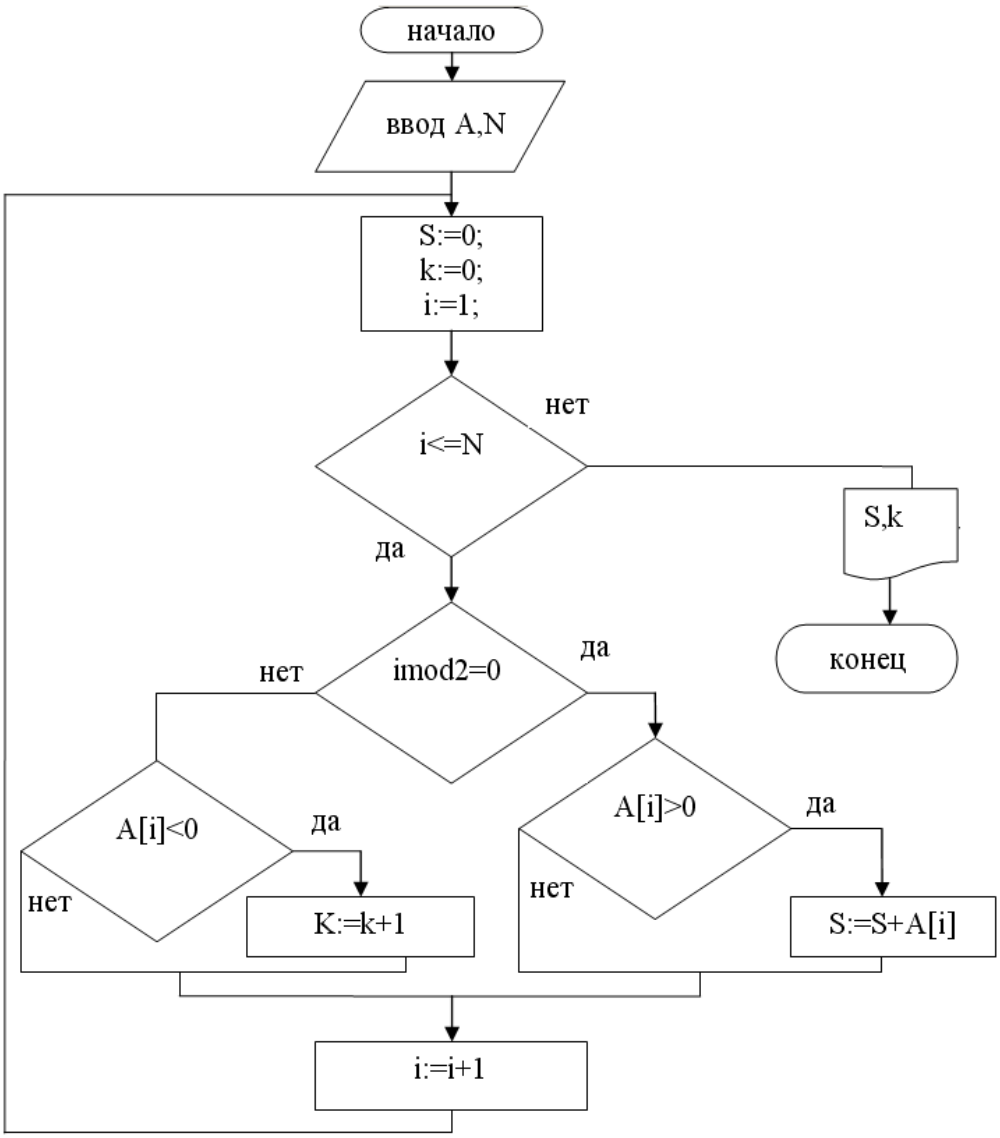
```
A = array [0..32767] of integer;
B = array [1..100] of real;
```

Здесь информация о диапазоне индексов и типах компонент указывается в разделе описания переменных.

Ввод и вывод значения массива всегда выполняется в цикле.

Пример: Найти максимальные и минимальные элементы a.

```
Program MinMax;
 Const MaxSize=20
  type  ListSize=1..MaxSize;
var
 i:=ListSize;
 min, max, k : integer;
 A: array [ListSize] of integer;
 begin
 ReadLn (‘введите массив’);
for i=1 to MaxSize do
      begin
            Write(‘введите A[‘i’]=’ );
            Readln(A[i]);
      end;
min:=A[1]; max:=min; {ищем максимальные и минимальные элементы }
for i:=2 to MaxSize do
      begin
            k:=A[i];
            if k> max then max:=k
            else if k<min then min:=k;
      end;
Writeln(‘min=’, min);
Writeln(‘max=’, max);
end.
```



k- количество;

S- сумма.

```
Program Mas;
const N=20
type  diap=1..N;
      mas=array [??????] of real;
var   S     :     real;
      i, k  :     integer;
      d     :     mas;
begin
      WriteLn('введите элементы массива');
      for i:= 1 to N do          {ввод }
      readln(a[i]);              {массива}
      for i:=1 to N do
            if imod2=0 then begin  if a[i]>0 then S:=S+a[i] end;
            else if a[i]<0 then k:=k+1;
      Writeln('сумма=', S, 'количество=',k);
end.
```

Многомерный массив на примере двумерного

Пусть задан переменная А следующим образом:

Var A : array [1..4] of array [1..3] of integer;

Эта переменная содержит четыре элемента, каждый из которых в свою очередь состоит из трех элементов – целых чисел.

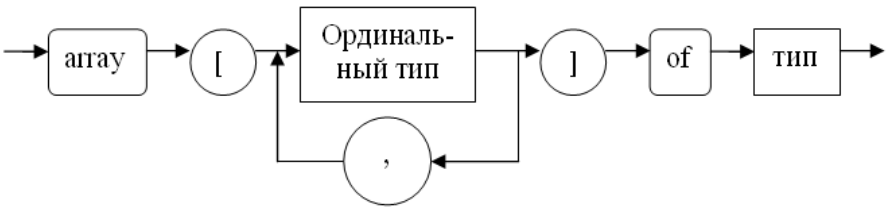
Такую переменную можно представить в виде таблицы (двумерного массива):

array [1..3]	array [1..4]			
	1	2	3	4
1	A[1,1]	A[2,1]	A[3,1]	A[4,1]
2	A[1,2]	A[2,2]	A[3,2]	A[4,2]
3	A[1,3]	A[2,3]	A[3,3]	A[4,3]

Доступ к элементам переменной А (запись и чтение значений элементов массива) может быть реализован так:

```
var   A     :     array [1..4] of array [1..3] of integer;
      i, j  :     integer;
begin
      i:=3; j:=2; {выбрали третий столбец, вторую строку}
      A[i,j]:= 5; {записали в ячейку на пересечении третьего столбеца и второй строки целое число 5}
      Write(A[i,j]); {вывели на экран ячейку таблицы на пересечении третьего столбеца и второй строки, т.е. целое число 5}
      end.
```

Общая схема многомерного описания массива



Если массивы A и B одного типа, то присваивание (A:=B) возможно если у них одинаковые типы компонент и типы индексов.

Например:

```
var
a, c: array [1..20] of real;
x: array  [1..10] of real;
b: array [1.20] of real;
```

Пример: Пусть в заданный текст входят только цифры. Вывести число в два раза превосходящее заданное. Программа может выглядеть так:

```
Program Num;
      Const N=4;
type range=1..N;
 var
      mas   : array [‘0’.. ‘9’] of integer;
      str   : array [range] of char;
      i     : range;
      sum   : integer;
begin (*ввод строки символов*)
for i:=1 to N do
read (str [i]);
mas [’0’]:=0; mas[‘1’]:=1;
……..
mas[‘8’]:=8; mas[‘9’]:= 9;
sum:=0;
for i:=1 to N do
sum:=sum*10+ mas[str[i]];
sum:=sum*2;
writeln(sum);
end.
```

2.8. Логические функции

Логические функции выполняются над данными имеющими тип Boolean. Например: A<1, A=5, false, true.

Определяется для каждой логической функции таблицей истинности, в которой изложены все возможные комбинации на входе, и соответствующие им комбинации на выходе. Значению логического нуля соответствует - false, логической единицы – true.

1) Логическая дизъюнкция (сложение (‘или’))

Обозначается: x:= A or B

A	B	x
false	false	false
false	true	true
true	false	true
true	true	true

Применительно к языкам программирования

2) Логическая конъюнкция(умножение (‘и’))

Обозначается: x:=A and B

A	B	x
false	false	false
false	true	false
true	false	false
true	true	false

В случае логического ‘или’ функция истинна, когда истинен хотя бы один из операндов. В случае логического ‘и’ функция истинна, когда истинны оба операнда.

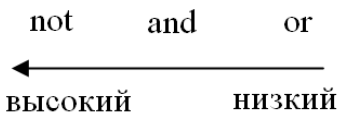
3) Логическая инверсия (отрицание (‘не’))

Обозначается: x:=not A

A	x
false	true
true	false

Фунция принимает значение, противоположное значению операнда.

Приоритет выполнения логических операций следующий:



Запишем истинные выражения при выполнении условия, и ложные в противном случае:

А) 0<x<1; (x>0) and (x<1)

Б) x-максимальное из трех чисел x, y, z

(x>y) and (x>z)

В) x – меньше по крайней мере одного из чисел n,y,z

(x<n) or (x<y) or (x<z)

Г) символ c – цифра

(c>=’0’) and (c<=’9’)

Д) символ c - не является цифрой

Not ((c>=’0’) and (c<=’9’))

Т.к. приоритет логических операций выше операций сравнения в языке Pascal, необходимо ставить скобки в операциях сравнения, находящихся между логическими операциями.

основные соотношения для логических функций булевой алгебры(Буль):

1)Правило снятия двойного отрицания: not not A=A

2) Свойство коммутативности:

A and B = B and A;

A or B = B or A;

3) Правило де Моргана

A and B=not (not A or not B);

A or B= not(not A and not B);

Not ((c>=’0’) and (c<=’9’))= not not (not (c>=’0’) or not (c<=;9;))= not (c>=’0’) or not (c<=’9’)= c<’0’ or c’9’

Основные соотношения для логических функций применяется при упрощении логических выражений.

2.9. Записи (record)

Запись состоит из фиксированного числа компонент называемых полями. Поля могут быть различных типов. Между массивами и записями имеются отличия, если все компоненты массива должны быть одного и того же типа, то записи могут содержать компоненты различных типов, различаются также механизмы выбора компонент, в случае массива применяется вычисляемый выбор компонент, т.е. с помощью переменной с индексами, т.к.записи содержат компоненты различных типов вычисляемый выбор для них не приемлем, вместо этого каждой компоненте даётся имя, которое затем используется для выбора поля.

В языке Pascal нет стандартного типа для комплексных чисел. Покажем как обойти это препятствие.

```
type
Complex=record
            re, im:real;
      end;
```

Аналогично можно определить тип даты рождения

```
type
DATE=record
      day:1..31;
      month:1..12;
      year:1900..2020
end.
```

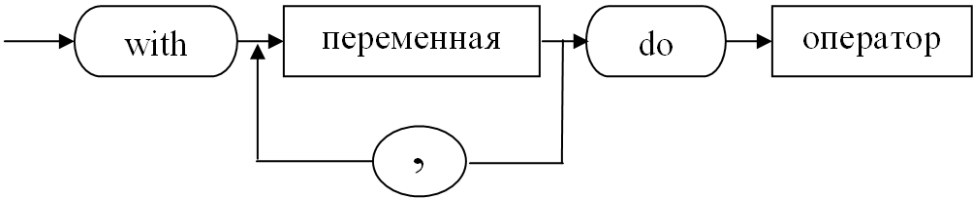
Пример программы вычисляющей сумму двух комплексных чисел.

```
Program ComplexExemple;
Type complex=record
      Re, Im:     real;
      end;
var x,y:complex;
begin
writeln (‘введите число’);
readln (x.re, x.im);
readln (y.re, y.im);
writeln (‘сумма =’, x.re+y.re,+i’,x.im+y.im);
end.
```

Запись вводится покомпонентно, чтобы обращаться к полям, необходимо указать идентификатор переменная – запись, за которым через точку указывается имя поля.

Оператор присоединения

Для обращения к полям часто используют одинаковые и/или очень большие имена переменных. Для сокращения записи составных имён используется оператор **with.**




```
x.im:=1; x.re:=2;
with x do
begin
      im:=1; re:=2;
end;
```

```
with A do
  with B do   ⟶  with A, B, C do
    with C do
```

Задача. Найти в группе тех студентов, которые имеют дни рождения в заданном месяце и году и рост от 175 до 185 см.

```
Program Group;
Const N=25;                                  {число студентов}
type person=record
            date: record
            day:1..31;
            mon:1..12;
            year:1950..2000;
            surname: string;
            hight:150..220;
      end;
var
      student      : array [1..N] of person;
      I      :     1..N;
      k,p    :     Boolean;
begin                        {ввод информации о студенте}
      For i:=1 to N do
      with student [i] do
      begin writeln (’введите данные’, i ,’-го студента’)
            readln (surname);
            with date do readln (day, mon, year);
            write (’рост’)
            readln (hight)
      end;
      k:=false;    {студент с указанными данными отсутствует}
      For i:=1, to N stud do
      with student [i] do
            begin p:=false;      {нет нужной даты рождения}
            with date do
                  begin
                        if (mon=3) and (year=1990)
                        then p:=true;
                  if p and (hight>=175) and (hight<=185) then
                        begin
                              writeln (surname);
                              k:=true
                        end;
            end;
      if k=false then writeln (’студент не найден’);
```

```
end.
```

При описании типа- **запись** имеются особенности, сначала желатльно определить соответствующий тип. А затем переменную имеющую данный тип, например

```
type person=record
        fam:string;
        mark:1..5;
        end;
var
p:person;
```

Фактически определение типа – это указание каким образом выделять память под соответствующие переменные. Поэтому обращение with person do - ошибка, with p do- правильно.

2.10. Строковый тип

Строка – это последовательность символов заключённая в апостроф.

Строки состоящие из 1-го символа представляют собой константы стандартного типа char. В стандартном Pascal строки состоящие из N символов (N>1) считаются константами типа:

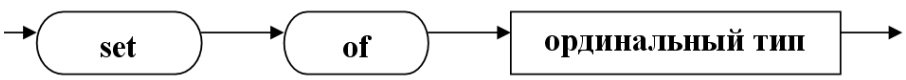
Packed array [1..N] of char;

Данные строки называются **строками постоянной длины** и в сфере Delphi практически не используются. Строки постоянной длины можно сравнивать покомпонентно, если они одинаковой длины. При сравнении строк попарно фактически сравниваются коды ASCII символов.

2.11. Множественные типы

Множество в математике – это произвольный набор объектов любой природы, понимаемый, как единое целое. На количество разновидность объектов не накладывается никаких ограничений. В языке Pascal допускается ограниченное число типов для элементов множества. Это может быть любой постой тип, за исключением вещественного (real) и в чистом виде целого (integer). Последнее ограничение связано с ограниченным числом максимально допустимого количества элементов множества.

Синтаксическая диаграмма для множественного типа имеет вид:



type letter=set of ‘A’..‘Z’

Конструктор множества

Конструктор множества задаёт совокупность конкретных элементов из которых состоит данное множество, например

```
type bigletter=set of 'A'..'Z';
       smallletter=set of 'a'..'z';
       cifra=0..of;
var   b:bigletter;
       s:smallletter;
       c:cifra;
       c:=[ ];    пустое множество
       b:['A'..'F'];
       s:['a', 'c', 'e', 'g']
```

Операции над множеством

Если x- переменная множества, а F- множественное выражение, то присваивание x:= F; i:=i+1;

Если все элементы F относятся к базовому типу x.

Если A и B- множества совместимых типов, то

A+B- объединение множества;

A*B- пересечение множества

A-B- разность множеств, т.е.множество элементов A не входящих в B

К множественным операциям применимы следующие операции (пусть e- ординальное выражение базового типа e в A вхождение во множество. Если e входит в A, то true):

A=B- равенство множеств

A<>B- неравенство множеств

A<=B- true, если A подмножество B

A>=B- true, если B подмножество A

'a' в s=true

S:=s+['n']; (к множеству s добавить 'n' так неправильно!)

Пусть переменная F содержит множество всех букв, g- маленькие буквы (на нижнем уровне), h- гласные

f-h- согласные

g*h- маленькие гласные буквы

g+h- маленькие и большие гласные

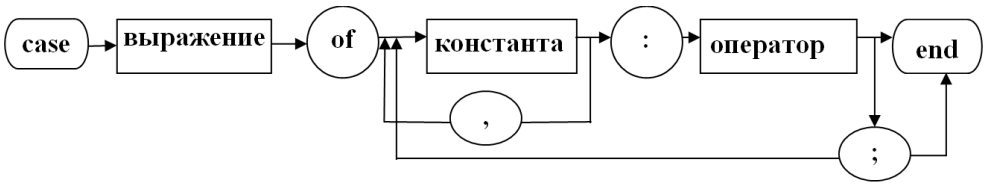
Пример. Прочитать последовательность цифр без знака и преобразовать её в число.

```
Program Example;
var ch:char;
      digist:set of '0'..'9';
      number:integer;
begin  digist:=['0'..'9'];
           read (ch);
           number:=0;
           while ch in digist do;
            begin  number:=number*10+ord(ch)-ord('0');
                       read (ch);
```

```
                end;
        writeln ('число=', number)
…
```

2.12. Оператор варианта (выбора)

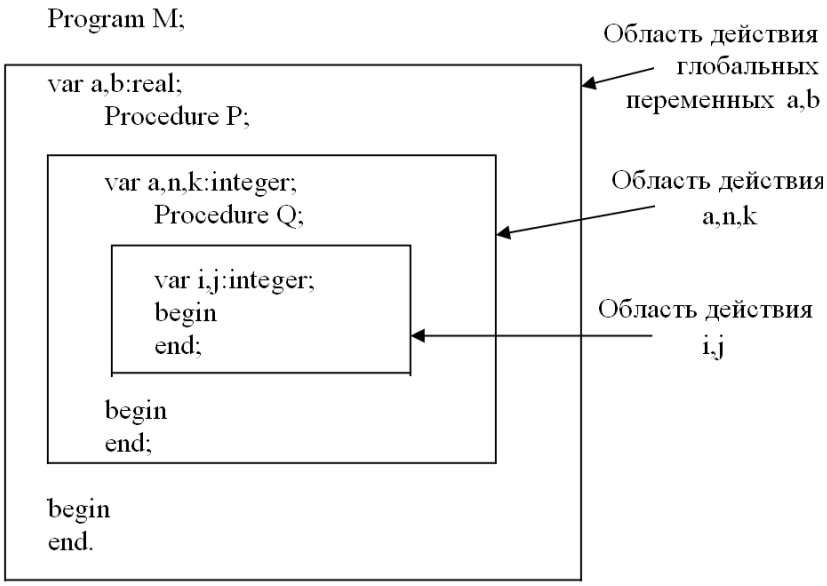
Этот оператор состоит из выражения и списка операторов, с каждым из которых сопоставлено одно или несколько постоянных значений, относящихся к типу селекторов. Тип селектора должен быть ординальным. Каждая из постоянных значений должно быть сопоставлено самое большее с одним оператором. Оператор варианта выбирает для выполнения оператор сопоставленный с текущим значением селектора. По окончании выполнения оператора, который выбран, управление передаётся в конец всего оператора варианта или к альтернативному варианту определяемому ключевым словом else.




```
i:integer
case i of
0:y:=1;
1,2,3:begin
                y:=0;
                a:=1;
                end;
4,5:y:=3;
else y:=1;
end.
```

2.13. Область действия и описания процедур и функций

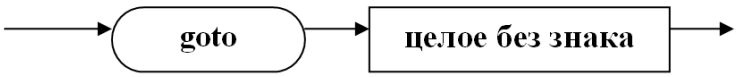
Описание каждой процедуры и функции по структуре похоже на программу, т.е. состоит из заголовков и блоков, следовательно описание процедур и функций могут вкладываться в описание других процедур и функций. Все описания внутри процедуры или функции имеют смысл только в тексте программы составляющим блок данной процедуры или функции, такой фрагмент называется **областью действия описанных имён и других объектов.**



Поскольку блоки могут быть вложенными один в другой, то вложенными могут быть и области действия. Объекты описываемые в основной программе, т.е. не локализованные в определённой процедуре или функции называются **глобальными** и доступны в любом месте программы. Объекты описываемые в процедуре или функции называются **локальными объектами** данной процедуры или функции. Возможно переопределение объектов во вложенных процедурах, но этого не рекомендуется делать.

Оператор перехода (безусловного перехода)

Оператор перехода указывает, что работа программы должна продолжаться с того места, где находится метка.



В Turbo Delphi используется имя (вот так: goto a1;)

Каждая метка: а) должна появляться в описании метки label перед её использованием в блоке; б) должна стоять только перед одним оператором входящим в состав оператора блока; в) в качестве области определения имеет область охватывающую весь текст за исключением вложенных блоков переописывающих данную метку. Переход внутрь сложного оператора из вне его запрещено.

Примеры неправильных переходов.

а)
```
for i:=1 to 10 do
   begin
          15:..
   end;
   goto 15;
```

б)
```
if A>0 then goto 3;
   if B<0 then 3;
```

в)
```
Procedure P;
   label 100;
   Procedure Q;
   begin
         100:..
     end;
   begin
    goto 100;
    ..............
    end.
```

г)
```
Program M;
   label 18,3;
   begin
         18:A:=0;
         ……..
          3:B:=1;
          18:C=
'A';
   end.
```

Задача. Найти в символьном массиве 1-ый элемент равный «А» и выдать его порядковый номер, при отсутствии элемента выдать сообщение.

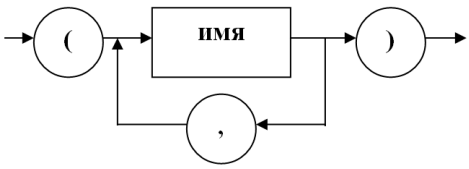
‘BCDA’..‘E’

```
Program Example;
label 125;
type dim=1..15;
var stz: array [dim] of char;
        i: dim;
begin for i:=1 to 15 do real (stz[i]);
         for i:=1 to 15 do if stz [i]='A' then
         begin
           writeln ('N=', i);       {N=4, т.к. А на 4 месте}
           goto 125    125:;
         end;
         writeln ('буква не найдена');
end.
```

В языке Pascal операторы перехода следует применять в исключительных случаях, когда приходится нарушать естественную структуру алгоритма.

2.14. Перечисляемые типы

Описание перечисляемого типа задаёт упорядоченное множество значений перечисляя имена констант обозначающих эти значения. Ординальный (порядковый) номер первый из перечисляемых констант равен 0, следующий 1 и т.д.




```
type color= (white, red, black, green);
       day of week= (mon, tue, wed, thu, fri, sat, sun);
       free= (sat, sun);
```

В Pascal определён автоматически перечисляемый тип Boolean= (false, true). Это означает, что false<true, что mon<tue. Ко всем перечисляемым типам применимы операции отношения, но оба операнда должны быть одного типа. Для аргументов относящихся к ординальным типам применимы следующие предопределённые функции:

succ (x) следующая за x
succ (red)=blue;
pred (x); предыдущее значение
ord (x); порядковый номер
ord (mon)=0, (порядковый номер начинающийся с 0)

Следует учитывать, что перечисляемый тип определяется с помощью констант имеющих символьные имена, поэтому значение переменных перечисляемого типа нельзя вводить с помощью оператора ввода, например

var c:color;

~~…….read (c);~~ ошибка!

Значение перечисляемого типа можно задать с помощью оператора присваивания

c:=green;

~~‘green’~~ не верно!

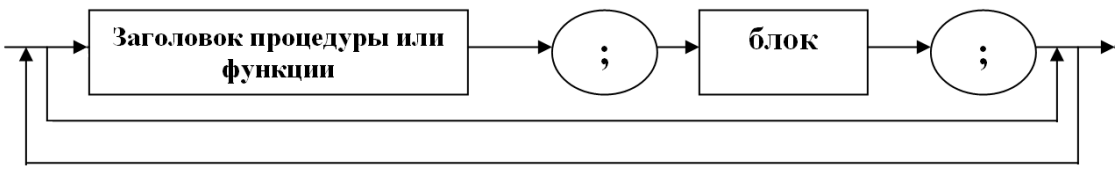
Пример. Определить название месяца следующего за заданным месяцем M и время года, к которому относится.

```
Program Pefine;
type mouth= (yan, feb, mar,apr, may…);
       year=(win, spr, sum, aut);
var m:mouth;
      y:year;
      num:1..10;
begin……
        m:=may;
        if m=dec then m:=yan; else m:=succ (in);
        case m of
        yan:writeln (‘январь’);
        ……..
        Dec:writeln (‘декабрь’);
        end;
        num:=ord (m)+1;
        case num of
        1,2,12:y:=win;
        3,4,5:y:=spr;
        6,7,8:y:=sum;
        9,10,11:y:=aut;
        end;
        case y of
       win:writeln (‘зима’);
       ……..
       aut:writeln (‘осень’);
       end;
end.
```

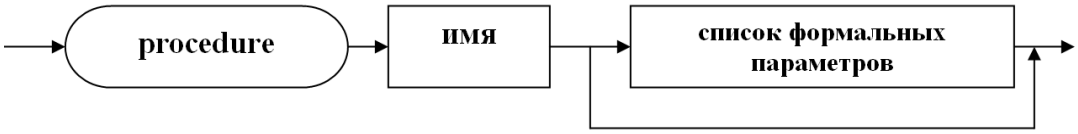
Перечисляемый тип задаёт константа с помощью алгоритма описываемого в программе, устанавливается логическое соотношение между перечисляемыми типами, при этом манипуляция осуществляется непосредственно с типами. Действия с перечисляемым типом, их результаты можно использовать для локализации программных продуктов. Таким образом, организация интерфейсов на разных языках.

2.15. Процедуры и функции

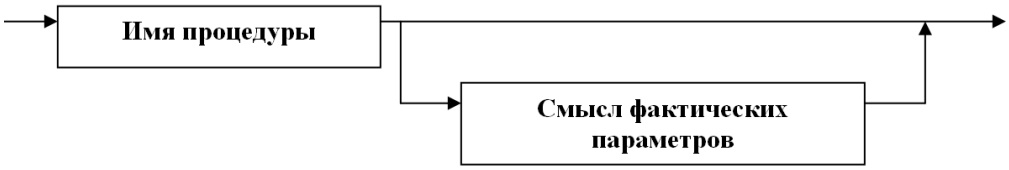
Представляют собой отдельные модули, оформляемые аналогично программе.



Синтаксическая диаграмма для заголовка процедуры.



Формально – это то и там, где соблюдена форма. Содержание меняет форму и наоборот. Оператор процедуры активизирует её работу.



Пример

Min (f,g,h)

f(x)=√x;

g(x)=sin?(x);

h(x)=e?*cos(x);

[0.5] с шагом 0.1

Решение

```
Program minimum;
var x,m,f,g,h,:real;
      procedure min;
      var x:real; {- локальная переменная процедуры min}
       begin if (f<g)and (f<h) then x:=f
                 else if (h<f) and (h<g) then x:=h
                        else x:=g;
                  m:=x;
       end;
begin  x:=0;
          while x<5 do
           begin f:=sqrt (x);
                     g:=sqr (sin(x));
                     h:=exp (x)*cos(x);
                     min;
writeln ('x=',x,'min=',m);
          x:=x+0.1
          end;
end.
```

Обмен данными между процедурой и программой осуществляется:

1) С помощью глобальных переменных, например программа min (указана выше). Данный способ не рекомендуется с целью обеспечения безопасности программ, т.к. могут возникать побочные эффекты, связанные с непредсказуемым изменением переменных. Применение данного случая допустимо, но только в редких случаях.

2) С помощью параметров. Он считается основным способом.

Списки параметров:

1.**формальные**, задаются при описании процедуры. В списке параметров даются имена каждому параметру, затем указывается их тип.

Пример: Procedure P (a,b,c:real; i:integer);

2.**фактические**, подставляются в операторе процедуры вместо формальных параметров.

Пример: P (1.3, 1.75, 3.4, 18);

Параметры бывают 4х видов:

1) Параметры-значения, фактические параметры должны быть выражениями, в простейшем случае это переменные или константы.

2) Параметры-переменные, фактическим параметром должна быть переменная, а перед формальным ставится ключевое слово var. Параметры-значения используются для передачи данных в процедуру. Параметры-переменные для передачи и обменом значениями между процедурой и основной программой.

3) Параметры-процедуры;

4) Параметры-функции.

Пример:

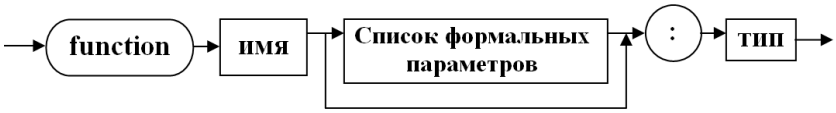
```
Program Parametr;
var a,b:integer;
procedure Add (x:integer, var y:integer);
begin x:=x+1;
      y:=y+1;
      writeln (x,y) ;
 end;
begin a:=0; b:=0;
      Add (a,b);
      writeln (a,b);
end.
```

```
 1  1     (выдастся)
 0  1
 ответ
```

При вызове процедуры, если мы имеем дело с параметром - значения, то передаваемое значение копируется в отдельную область памяти, которая и используется в процедуре, например параметр a. Т.е. при выходе из процедуры выделенная память перестаёт использоваться и значение переданного параметра не изменится. В случае использования в операторе процедуры параметра переменной в процедуру фактически передаётся сама переменная, т.е. её значение в основной программе также изменится.

Функции

Функция - часть программы, определяющая явным образом одно ординальное, вещественное или ссылочное значение, поэтому функцию, возможно, использовать в выражении в качестве операнда. Функция во многом схожа с процедурой и её описание имеет вид



Пример. Вычислить значение функции sh(x)*tg(x)+sh(sin(x)) на интервале от 0 до 1 с шагом 0.01.

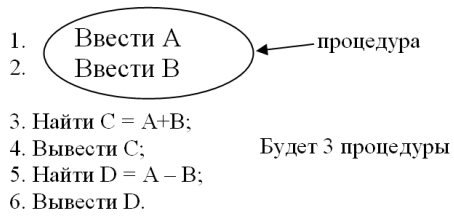
Решение:

```
Program F.primer;
var x,y:real;
function sh(z:real):real;
```

```
var e:real;
begin
      e:=exp(z);
      sh:=(e-1/e)/2; {Передача значений в основную программу осуществляется с
помощью его присваивания имени функции.}
end;
function my-tg(z:real):real;
      begin my-tg:=sin(z)/cos(z);
end;
begin x:=0;
      write x<=1 do
      begin y:=sh(x)*my-tg(x)+sh(sin(x));
            writeln (x=',x', y=',y);
            x:=x+0.01;
      end;
end.
```

Обычно в виде процедур и функций оформляют относительно часто повторяемые в программе действия или сложные алгоритмы, чтобы повысить читаемость программ и упростить их откладку. Для разбиения на процедуры и функции полезно составлять словесные написания алгоритмов.

Пример. Дано матрица A и B. Получить C и D.



Таким образом, после словесного описания алгоритма нужно реализовать 3 процедуры: вести, найти, вывести.

```
Program PRIMER;
const N=4; M=5; {число строк и столбцов}
type ranstr=1..N;
      rancol=     1..M;
      mas   =     array[ranstr;rancol] of real;
var   I           :=    ranstr;
      j           :=    rancol;
      A,B,C,D     :     mas;
Procedure inpt (var Matrix:mas);
var   str   :     ranstr;
      col   :     rancol;
begin for str:=1 to N do
            for col:=1 to M do read Matrix [str,col];
end;
Procedure sumsub (b:Boolean, A1:mas, A2:mas, var sum:mas);
var   str   :     ranstr;
      col   :     rancol;
      key   :     -1..1;
begin
      if b then key:=1 else key:=-1;
      for str:=1 to N do
            for col:=1 to M do sum [str,col]:=A1 [str,col]+A2 [str,col]*key;
end;
Procedure outpt (M:mas);
var   str   :     ranstr;
      col   :     rancol;
begin
      for str:=1 to N do
      begin
            for col:=1 to M do
            write (M[str,col]);
            writeln;
```

```
      end;
end;
begin
writeln (‘введите 1-ую матрицу’); inpt (A);
writeln (‘введите 2-ую матрицу’); inpt (B);
sumsub (true,A,B,C);
writeln (‘’); outpt (C);
sumsub (false,A,B,D);
writeln (‘’); outpt (D);
end.
```

При описании символов в качестве формальных параметров, процедур и функций, необходимо указывать имя типа. Описание самого массива непосредственно указывать нельзя. В практической программе нужно соблюдать осторожность при использовании массивов в качестве параметра. Это связано с особенностями выделения памяти для параметров при вызове процедур и функций. При активации процедур или функций для каждого параметра значения выделяется место в памяти, текущее значение параметра копируется туда, при выходе память освобождается. Обращение к параметру-значения происходит быстрее и есть защита от несанкционированного изменения данных. Но в случае параметра составного типа, например, массив, нужно соблюдать осторожность, т.к. операция копирования относительно медленная и может потребоваться достаточно большое место памяти.

Рекурсивные процедуры и функции.

Использование процедуры или функции в тексте этой же процедуры или функции предполагает её рекурсивное выполнение, не следует путать рекурсию с вложенным определением процедуры или функции!, т.е. когда одна процедура определена в теле другой. **Рекурсия -** это вызов процедурой или функцией самой себя прямо или косвенно.

$\underbrace{f(f(x))}_{n\text{ раз}}$, где $e(x)/(1+\sin(x))$, $f=x^2$

$x=4$

$x=4^2$ обращается несколько раз

$x=16^2$

```
Program Recurs;
var z:real; n:integer;
function F (x:real,p:intiger):real;
var y:real;
begin
     y:=exp(x)/(1+sin(x));
     if p>1 then F:F(y,p-1)
     else F:=y;
     end;    {z=1.5;}
begin  writeln (‘введите x, n’);      {n=4}
          readln (z,n);
          writeln (‘F=’, F(z,h));
end.
```

При рекурсивном вызове необходимо позаботится об условии выхода из рекурсии (когда p=1-это будет выход из рекурсии). В противном случае можно столкнуться с бесконечным вызовом рекурсивных процедур или

функций и переполнением памяти. Эта ситуация аналогична зацикливанию. При рекурсивном вызове подпрограммы создаётся её дополнительная копия в памяти. При успешном выходе из рекурсии, память выделенная под рекурсивные вызовы освобождается.

Параметры процедуры. Параметры функции.

В каждой реализации Паскаля предусмотрены стандартные процедуры, которые нельзя передавать в качестве фактических процедурных или функциональных параметров.

Опережающее описание процедур и функций.

Используется в случае, когда подпрограмма P вызывает Q, а Q вызывает P.

```
Procedure Q (x:real); Forward;
Procedure P (y:real);
begin….
        Q (y);
       ……
end;
Procedure Q;
begin
        P(x);
end;
```

Побочные эффекты - ситуация, когда программа выполняет действия помимо тех, что предполагает программист и о которых программист не знает. Существование побочных эффектов объективно, но ситуации, когда они возникают, следует избегать. В частности к побочным эффектам может привести использование параметров совместно с глобальными переменными в процедурах и функциях. Более безопасным с точки зрения побочных эффектов является использование параметров.

```
 Program SideEffect;
var a,z:integer;
      function change (x:integer):integer;
       begin z:z-x;
                change:=SQR(x);
         end;
begin z:=10; a:=change(z);
 writeln (a,z);
         z:=10; a:=change(10)*change(z);
writeln (a,z);
         z:=10; a:=change(z)*change(10);
writeln (a,z);
end.
```

Ответы (что может получиться)

1) 100,10
2) 10 000,10
3) 10 000,10

100,0 ← побочный эффект

В данном примере «неожиданные» результаты вызваны тем, что глобальные переменные используются совместно с параметрами в функции.

Строковый тип в Turbo Delphi

Строковый тип данных определяет последовательность символов произвольной длины от 0 до 255. Если нужно уменьшить длину строки, то указывают в её описании.

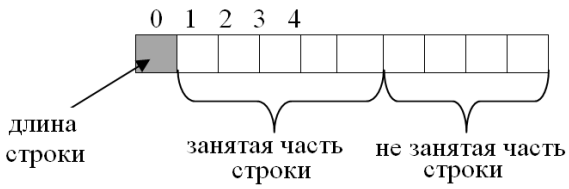
```
type fam=string[50];
       group=string[7];
var
    s;string;
```

Достоинство типа string в том, что строка может иметь переменную длину, при этом ввод/вывод можно осуществить с помощью операторов ввода/вывода, кроме того допустимы операции присваивания и объединения.

Пример.

```
var S1,S2:string;
      S1:='моя группа';
      S2:='№ группы';
 Write (S1+ ' '+S2)
```

Реализация механизма строковых типов.



Т.е. фактически используются только части памяти отводимой под строку. Кроме этого определены операции сравнения <, <=, >, >=, <>. При сравнении действуют следующие правила: 1) более короткая строка меньше; 2) если длины равны, то производится поэлементное сравнение

Пример: ABCD<ABDC

В Turbo Delphi существуют функции для работы со строками:

1) lenght-длина строки ();

2) copy-возвращает подстроку из заданной строки;

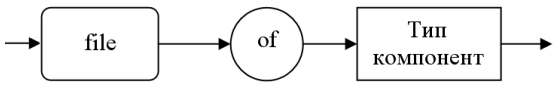
3) delete-выполняет удаление подстроки и заданной строки начиная с определённой позиции;

4) insert-вставляет строку 1 в строку 2 с определённой позиции;

5) pos-возвращает позицию начала заданной подстроки в строке.

2.16. Файловый тип

В Паскале слово файл относится к последовательности компонент, при этом все компоненты одного типа. Файл содержит конечное, но не фиксированное ранее число компонент. Если файл не содержит компонент его называют пустым. В стандартном Паскале файлы отличались последовательным доступом компонентов. Преимущества: для оперирования с таким файлом нужно всего две операции. Недостатки: неудобство в работе. В современных версиях Паскаля используется прямой доступ. Синтаксическая диаграмма для файла имеет вид:




```
var  f: file of integer;
```

Описание каждой переменной- файла f автоматически вводит некоторую буферную переменную, обозначаемую f↑. Эта буферная переменная имеет тип определённый для компонент файла.

Для работы с файлами предусмотрены процедуры и функции:

eof (F) – конец файла, если F не определено – это ошибка. Обращение eof(F) даёт true, если F находится в режиме формирования, записи или до обращения F был установлен на последнюю компоненту последовательности.

Reset(F) – готовит файл для просмотра (чтения), помещая указатель на начало файла. Если файл не пустой, то eof(F) = false.

Rewrite(F) – готовит файл для формирования (записи). Значение (F) заменяется на пустой файл eof(F) при этом = true.

При работе с файлами имеется следующая особенность: если F-файловая переменная, то Read (F, x) означает присвоение значению x значения текущей компоненты и переход к следующей компоненте.

Общие схемы для работы с файлами:

1) просмотр файла F

```
Reset(F
While not eof(F) do
 begin Read(F,…);
end.
```

2) создание файла F

```
Rewrite(F);
while  условие создания файла do
begin…
write (F,…)
end;
```

Условием создания файла может являться отсутствие нажатия определённой клавиши, отсутствие определённого события.

В Turbo Delphi существует механизм связи между файловой переменной и именем файла ASSIGN (F, имя файла).

Close (F) – закрытие файла. Применение даёт гарантию, что все операции с файлом гарантированно завершены на диске. В процессе работы многие операции над информацией содержащейся в файле осуществляются в оперативной памяти.

Задача: написать программу, которая подсчитывает среднее арифметическое всех элементов файла и выдаёт количество элементов меньших этого значения.

```
Program Files;
Var   k     :     integer;
x, s  :     real;
f     :     file of real;
begin {подсчёт среднего арифметического}
ASSIGN(F, my-file.dat);
Reset(F);
```

```
k:=0; s:=0;
repeat
read(f, x); k:= k+1; s:= s+x;
until eof(F);
s:= s/k; k:= 0;
reset(F);
read(f,x);
if x< s then k:= k+1;
until  eof (F);
Writeln ('число =',k);
end.
```

Файл типа real – это типизированный файл, для которого не существует стандартных программ обработки. Поэтому программу создания файла типа real необходимо реализовать самостоятельно, приведём фрагмент:

```
Program Filecreate;
…
begin
ASSIGN (F, my file.dat);
Rewrite (F);
Repeat;
Readln (x);          {- чтение с клавиатуры}
Write(F, x);         {- запись в файл}
until
…                    {- условие прекращения ввода}
close (F);
…
```

Понятие формата файла подразумевает интерпретацию данных содержащихся в нём, как имеющих определённый тип.

Текстовые файлы – это файлы компонентами которых являются последовательности символов разделённые на строки произвольной длины с помощью концевых маркеров.

Для описания таких файлов используется предопределённый тип text; маркер конца строки может распознаваться и формироваться специальными текстовыми процедурами.

Маркер конца строки чаще всего представляется символами с кодами 1316 и 1516 (перевод строки; возврат строки). Иногда концевой маркер представлен одним символом.

Writeln (F) – заканчивает строку текстового файла F, т. е. записывает концевой маркер.

Readln (F) – пропускает все символы до начала следующей строки текстового файла.

Eoln (F) – логическая функция, определяющая достигнут ли конец текущей строки в файле F.

Схемы работы с файловыми:

F: text; - формирование файла F line – логическая переменная = true, когда нужно закончить строку, fl = true, когда нужно закончить весь текст.

```
…
Rewrite (F);
repeat
repeat
write (F, c);
until line;
…..
Writeln (F);
Until fl;
```

Чтение текстового файла

```
T: text;
…
Reset (T)
while not eof (T) do
Begin while not eoln(T) do
Begin Read (T,c)
end;
Readln (T)
End;
```

Копирование текстового файла in в текстовый файл out, с сохранением строчной структуры in c: char;

```
Reset (in) ; Rewrite (out);
While not eof in do
begin
while not eoln (in) do
begin
Read (in, c);
Write (out, c)
end;
Readln (in);
Writeln (out);
end;
```

Форматированный вывод текстовых данных. Процедуры Read и Writeln могут осуществлять вывод данных в определённом формате, задаваемым числом позиций. Фактически задаётся минимальный размер поля и если он не указан, присваивается значение по умолчанию, в соответствии с типом переменных.

Если число символов больше числа заданных позиций, то оно выдаётся целиком.

```
Write (a:8);
```

Число выдаётся в формате E

```
Write (a:10:2)
a:=0.1573
Write (a:10)1.573E – 001
```

2.17. Логическая и физическая организация данных.

Логическая организация – это внешнее представление данных, без учёта того, как они организованны реально, на конкретном физическом носителе.

Соответственно операции над данными могут носить логический и физический характер. Например : логическое удаление. В файле запись существует, но в ней имеется признак логического удаления. Если признак true – запись удалена, если false – нет.

Помеченные для удаления записи не выводятся при просмотре, т. е. пользователь их не видит.

Физическое удаление – это фактическое удаление записи при этом обычно оно происходит следующим образом: файл переписывается в другой без удалённых записей, исходный файл переименовывается в другой без удалённых записей, исходный файл переименовывается или удаляется, затем имя исходного файла меняется на имя существовавшего до этого файла.

Принцип виртуальности предполагает абстрактную реализацию одного устройства (явления) с помощью другого устройства (явления).

Динамические структуры данных – те, которые во время выполнения программы могут увеличиваться или уменьшаться в размере (логически).

Стек (линейный список с одной точкой доступа - вершиной), также называется структурой LIFO.

Пример: проверить правильность баланса скобок в выражении, записанном в соответствии с правилами синтаксиса Паскаля.

Наиболее простой алгоритм – это арифметический подсчёт числа открывающих и закрывающих скобок, но данный алгоритм неверен, т.к. нужно соблюдать соответствие разных типов скобок и учитывать, что каждой открывающей скобке соответствует закрывающей, но не наоборот.

((( ))) – верно

))) (((- неверно – (([( )])

В данном случае для верного решения задачи используется алгоритм с применением стека. Пока не конец выражения помещаем в стек открывающую скобку, если на входе закрывающая, то смотрим находится ли в стеке соответствующая открывающей. Если нет – ошибка.

Если по окончании разбора стек пустой, скобочное выражение записано правильно.

Пример:

```
Program stack;
const LEVEL=15;
var
s     :     string;
I     :     integer;
ch    :     char;
msopen,
msclose:    array [1..2] of char;
stack :     array [0 LEVEL] of integer;
nst   :     0..LEVEL;    {указатель стека}
flag  : boolean;
k: 1..2;
begin {задаём соответствие открытых и закрытых скобок}
msopen [1] : '( '; msopen [2] := '[';
msclose [1] : ')'; msclose [2] := ']';
Readln (S);
i := 1, ch := S[i]; nst:= 1;
flag := folse {обнаружение ошибок}
while (ch<>',') and flag = false do
if ch = msopen [k] then
begin
stack[nst]:=k; {номер скобки в стек};
nst:= nst+1;
end
else
if ch=msclose [k] then
begin
nst:= nst-1;
if (nst=0) or stack < [nst] <>k then flag:= true
end;
if flag or (nst>1) then {ошибка или в стеке остались скобки}
Writeln ('нарушен баланс скобки')
else  Writeln ('ошибок нет')
end.
```

Обычно работа со стеком производится следующим образом: в стек заносится элемент, затем указатель стека увеличивается.

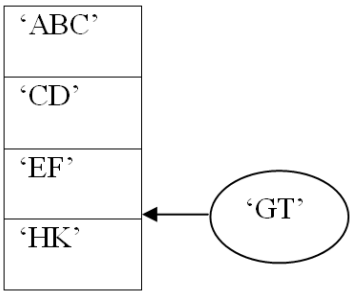
Вынесение элементов стека : указатель стека уменьшается и производится работа с выбранным элементом.

Очередь – способ организации ожидания.

К данным находящимся в очереди можно получить доступ через две точки, данные могут быть добавлены в конец очереди и удалены из её начала. Очередь – это структура типа First in – First out.

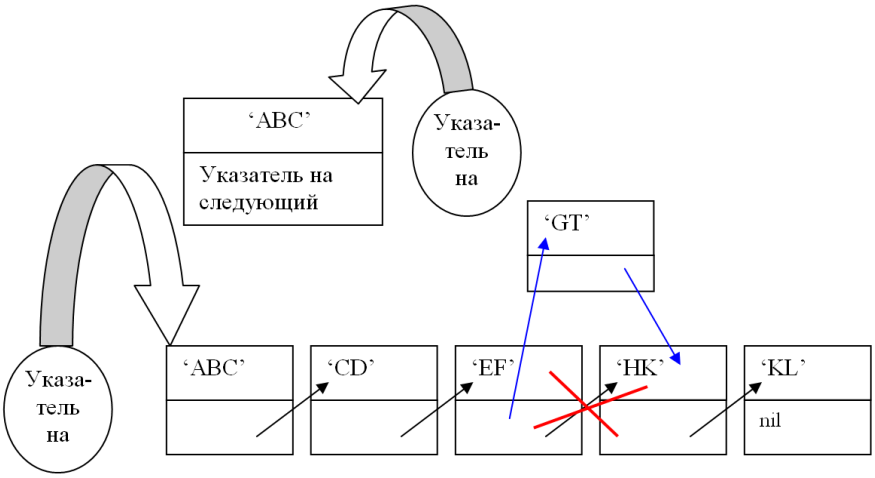
Очереди используются для организации ожидания в мультипрограммных средах.

Пусть необходимо хранить список названий, упорядоченных по алфавиту. Пусть требуется вставить новое название ‘GT’. Если мы используем массив, то, чтобы освободить место для нового названия, необходимо передвинуть элементы массива.



Альтернативным является представление массива с помощью структуры данных, называемой связным списком. Наиболее эффективно связный список реализуется с помощью динамической памяти и указателя.

Связный список можно изобразить так:



Фактически вставка элемента в список – это манипуляции с указателями.

Переход связанного списка, пока не конец списка, берём ссылку на следующий элемент, извлекаем его и т. д.

Удаление узла из связанного списка – это также работа с указателями.

2.18. Ссылочные типы.

Рассмотрим кратко такой важный тп данных как ссылки.

Статистические и динамические переменные.

Память для статистических переменных отводится при загрузке программы в память, т. е. непосредственно перед выполнением. Для определения идентификатора переменной и объёма выделяемой памяти, статистические переменные необходимо описывать. Отведенная память сохраняется за переменной на всё время выполнения модуля, в котором она локализована.

У программиста есть возможность описать статистические переменные, но нет возможности управлять памятью. Статистические переменные описываются в разделе описания переменных.

Для динамических переменных память отводится и уничтожается в процессе выполнения программы.

Динамические переменные не указываются в разделе описания переменных, а порождаются и уничтожаются с помощью предопределённых процедур NEW, DISPOSE.

Для работы с динамической переменной в программе необходимо обеспечить доступ к выделенному участку памяти, это осуществляется с помощью специальных переменных – указателей. Тип переменной – указателя ссылочный. Описание ссылочного типа имеет вид:

ссылочный тип = ↑ идентификатор типа;

Пример :

```
Type
ptr    =     ↑integer; (указатель на целое )
ptreal       =     ↑real; (указатель на вещественное)
var
p      :     ptr; {статитическая переменная, указатель на integer}
a      :     ptreal; {статистическая переменная, указатель на вещественное}
```

p↑ - Динамическая переменная.

Если p= nil, то указатель ни на что не ссылается и обращение p ↑ - ошибка.

Фактически p↑ означает операцию взятия содержимого находящегося по адресу содержащемуся в p.

Пример:

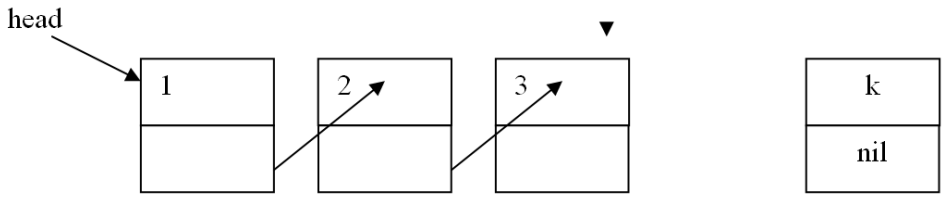
```
var i, y : integer;
begin
        i := 51;
        y := i;
Write (i, y);
End.
```

```
var i, y :  ↑   integer;
begin new(i)
   y:= i;
    y ↑  := 51;
Write (i, y);                {ошибка!}
Write (i↑, y↑ );
End.
```

Задача: дан файл с фамилиями абонентов стоящих в очереди. Вывести фамилии первых n абонентов в алфавитном порядке с указанием номера очереди.

Решение.

1) Читаем данные из файла и формируем связный список. Head – указатель на начало. Номер в очереди соответствует номеру строки с фамилией в файле.



linle

2) Проходим список для буквы ch, пока номер в очереди меньше n и не достигнут конец списка. Если первая буква фамилии совпадает с ch выдаём её.

```
Program Family;
Type str = string[25];
Pointer = ↑abonent;
{Использование ссылки на неопределённый тип до его описания допустимо, т.к. размер указателя не зависит от размера участка памяти на который он ссылается.}
abonent = real;
name: str;
number: integer;
link: pointer;
end;
var string1; str, mainhad:     pointer;
       k, num      :     integer;
       F           :     text;
       Ch    :     char;
Procedure vvod; {вводим фамилии и формируем один элемент связного списка}
var pnt: pointer;
begin new (pnt);
if head = nil then head := pnt
else main↑.link:= pnt;
pnt↑.name:= string1 (фамилия);
pnt↑.namber:= num;
pnt↑.link:= nil {признак последнего элемента};
main:= pnt; {main содержит ссылку на предыдущий элемент}
end.
Procedure out abonent
numb: integer;
nead p: pointer;
var  pnt: pointer;
begin
        while (numb > 0) and (head p<> nil) do
begin
        pnt:= head p;
        head:= head p↑.link; (ссылка на следующий);
        if pnt↑.name[1]= ch then
WriteLn (pnt.Name : 17, pnt. Number:16);

numb:= numb- 1;
end
end;
begin
        ASSIGN ('F; abonent. Dat');
        Reset(F);
         Num: = 1; { № очереди}
         Head:= nil;
         While not eof (F) do
begin
          ReadLn (F, string 1);
         Vvod;
          Num := num+1;
End;
Close (F);
WriteLn ('введите число абонентов');
ReadLn(K);
if K>= num then
WriteLn ('много')
      else  for ch:= 'A' to 'Z' do outal (k; head);
```

```
end.
```

При выдаче фамилий проход связного списка осуществляется для каждой буквы, пока номер очереди больше нуля и указатель не содержит нулевую ссылку.

3. Объектно-ориентированное программирование.

Объекты моделируют свойства мира в котором мы живём, они являются конечной абстракцией данных.

Рассмотрим такой объект, как точка. В традиционном языке программирования с помощью структурированного типа записи описываются координаты точки. Все манипуляции с точкой описываются с помощью подпрограмм отдельно, т. е. структура объекта отделена от действий, которые выполняются им или над ним. Это порождает неудобства, связанные с отсутствием привязки к конкретному источнику информации. В объектно-ориентированном подходе, объект описывается вместе с действиями. Язык ООП характеризуется тремя основными свойствами:

инкапсуляция – это комбинирование записи с процедурами и функциями, которые ей манипулируют, в результате получается новый тип данных – объект, т. е. все структуры и действия помещаются вместе в «единую капсулу».

Реализация процедур и функций, манипулирующих с объектом, помещается отдельно.

наследование – это определение объекта и затем использование его для получения иерархии производных объектов, при этом каждый объект-потомок использует и наследует доступ к коду и данным своих прародителей.

полиморфизм – это предание действию определённого имени, которое используется всеми объектами иерархии, причём каждый объект реализует действие особым подходящим для него образом.

Построим иерархию объектов от местоположения до окружности и оформим её в виде модуля.

```
UNIT POINTS;
Interface
Uses Graph;
type
       location = object;
       x, y: integer;
        Procedure init (init x, init y: integer);
Инициирует местоположение точки.
         function  Get x: integer;
         function  Get y: integer;
end;
          point = object (location);
          visible: Boolean;
          Procedure init (init x, init y: integer);
          Procedure show;
          Procedure hide;
          Function is visible: Boolean;
          Procedure move to (new x, new y: integer);
```

```
End;
Implementation
{Реализация правил объектов location }
Procedure location. init (init x, init y: integer);
begin
          x:= init x;
          y:= init y;
end;
function location. Get x: integer;
begin
        Get x:= x;
End;
function location;
Get y: integer;
begin
        Get y:= y;
End;
{Реализация правил объекта point.}
Procedure Point. Init (init x, init y: integer );
begin
        location. Init (init x, init y: integer );
        visible:=  false;
end;
Procedure Point. Show;
begin
       visible:=  true;
Put pixel (x, y, Get color);
End;
 Procedure Point. Hide;
begin
        visible:=  false;
{рисуем цветом фона – гасим}
 Put pixel (x, y, Get Bk color);
End;
Function point. is visible: boolean;
begin
        is visible:= visible;
end;
Procedure Point. move to  (new x, new y: integer );
begin
        hide;
        x:= new x;
        y:= new y;
        Show;
end;
```

При описании иерархии типов необходимо учитывать, что:

методы, определённые в объективном типе имеют доступ ко всем полям этого типа. Можно обращаться к полям непосредственно, используя точечную нотацию location. x, однако, это считается отходом от объектно–ориентированного стиля, согласно которому все действия с информацией, содержащейся в объективном типе, осуществляется с помощью методов.

Отличие объекта от модуля заключается в следующем: модуль рассматривается как часть программы, оформленная в виде отдельно хранящейся и отдельно компилируемой конструкции. Нельзя определить переменную модульного типа. Определив объективный тип можно создать произвольное число элементов этого типа, имеющих свои методы для манипулирования значениями полей.

На практике определяют объект с максимально возможным числом методов. Если в программе будут использоваться лишь некоторые, то в программный код попадут только те методы, которые реально вызываются

программой. Чтобы использовать объекты и правила, определённые в модуле point's, необходимо использовать этот модуль в своей программе.

```
Program Make points;
Uses points
var new point: point;
begin
        new point. init(18,37);
        new point. Show;
new point. Move to (100, 105);
new point. Hide;
end.
```

Расширение объектов

В Паскале существуют гибкие процедуры write и read. Гибкость заключается в том, что у данных процедур количество и тип параметров могут быть любыми. В стандартном Паскале нет средств, с помощью которых можно написать такие процедуры. В ООП проблема с тем, что одинаковые действия могут содержать разные параметры для различных типов объекта решается по средствам наследования, когда производный тип определён, наследуется правило родительского типа, но они при желании могут быть заменены. Для замены унаследованного правила определяется правило с тем же именем, но другим набором параметров. Продолжим иерархию типов и определим производный объект от объекта point, который будет рисовать на экране окружность.

```
Circle= object (point);
 Radius: integer;
Procedure init (init x, init y, init R: integer);
Procedure hide;
Procedure Expand (by: integer);
Procedure move to (new x, new y: integer);
Procedure Contract (by: integer);
End;
```

Реализация методов для типа circle.

```
Procedure Circle. init (init x, init y, init R: integer);
begin
        Point. init(init x, init y);
        Radius:= init R;
End;
```

Вызов правила, которое мы заменяем (point и init) крайне желателен, так как возможно, что point и init выполняет определённые действия. При внесении изменений в родительское правило, изменения так же коснуться потомков.

Динамическое распределение памяти под объекты

```
var One Circle:   ↑circle;
begin new (OneCircle);
OneCircle↑.init (50, 50, 10);
End.
```

Допускается совмещение этих двух действий в одном вызове New (One circle, init (50, 50, 10)); первый параметр однозначно определяет, для какого объекта берётся

init.Dispose (One circle); - освобождает память.

С процедурой освобождения памяти, выделенной под объекты, связан особый вид методов – деструкторов. Для описания деструктора необходимо добавить, например, в типе Circle строку Destructor Done.

Реализация деструктора может быть такой:

```
  Destructor Done;
begin hide;
end;
```

Использование деструкторов необходимо, когда экземпляр объекта был создан динамически. Завершением работы с объектом рекомендуется завершать следующим образом: Dispose (One Circle Done).

Использование деструктора Done вне Dispose не приведёт к освобождению памяти занимаемый объект.

Необходимость введения деструкторов объясняется следующим образом: пусть Pn Point – указатель на родитель, Pn Circle – указатель на потомок.

New (Pn Circle, init (50, 70, 15)); - создаёт экземпляр объекта – потомка. По правилу присваивания допустимо присвоение Pn Circle Pn Point. После присваивания Pn Point содержит ссылку на область памяти меньшего размера, чем отводится под Circle. В данном случае Dispose (Pn Point) приведёт к освобождению только участка памяти под

Point, а не под Circle. Применение Dispose (Pn Point, Done) приведёт к освобождению именно того участка памяти, который реально занимает объект, на который ссылается Pn Point, это связанно с тем, что деструктор будет вызван для объекта Circle.

Особенности использования Паскаля в качестве инструментального языка программирования.

Типы и области действия

Паскаль является языком программирования с сильной типизацией, т. е. каждый объект имеет тип, определяющий допустимые значения и набор операций. Применение недопустимых операций отслеживается транслятором, т. е. можно обнаружить ошибку ещё до выполнения программы.

```
var     mas  10: array [1..10] of real;
          mas  20: array [1..20] of real;
```

Так как mas 10 и mas 20 имеют разные типы, оказывается невозможным написать универсальную процедуру обработки, например сортировки любого из них. В различных реализациях Паскаля предложены способы устранения этих недостатков, например путём введения типа string для строк. Статистические переменные и возможность их инициализации на этапе трансляции в стандартном Паскале отсутствуют. Если при использовании Паскаля необходимо, чтобы переменная сохраняла своё значение в промежутках между обращениями к некоторой процедуре или функции, переменная должна быть внешней, таким образом, она оказывается доступной для других процедур и функций, и описание находится далеко от

того места, где переменная используется. Необходимость присваивания переменной значений на этапе выполнения удлиняет текст и объектный код. В Turbo Delphi существует возможность инициализации значений переменных в разделе const.

Структура программы:

Паскаль разработан так, чтобы можно было реализовать транслятор с него по однопроходной схеме, но для этого необходимо, чтобы описание любых объектов предшествовало их использованию. Это же относится к процедурам и функциям. Поэтому раздел процедур и функций предшествует основной программе. В свою очередь это противоречит принципу структурного построения программы, когда реализацию необходимо начинать с основной программы. Отсутствие возможности проведения раздельной трансляции на уровне стандарта затрудняет разработку больших программ. В зависимости от реализации, проблема раздельной трансляции решается по-разному. В Turbo Delphi используется модуль init.

В Паскале запрещено использовать в параметрах процедур и функций при их описании, типы данных не являющиеся базовыми. Например:

```
Procedure P(ms: array [1..10] of integer).
Type mas = array [1..10] of integer;
Procedure P(ms: mas);
```

Это не всегда удобно, так как описание типа может оказаться далеко от того места, где он используется. Использование по умолчанию того принципа, что параметры, являющиеся массивами, передаются по значению, приводит к затратам по памяти и скорости выполнения. Тип множества удобен в использовании, но крайне ограничен – это связано с тем, что максимальное число элементов во множестве не превышает определённо, довольно малой величины. В языке Паскаль отсутствуют возможности нарушения механизма сильной типизации. Это означает, что Паскаль непригоден для написания программ типа распределения памяти и систем ввода – вывода, так как эти программы могут возвращать в качестве своих значений и оперировать параметрами и объектами различных типов.

Управление последовательностью действий

В Паскале отсутствует гарантированный порядок выполнения операций OR, AND.

```
If  (i<>0) and ((a/i)>0) then.
```

В Паскале выполнение a/I произойдёт в любом случае, даже если i=0. В стандарте Паскаля отсутствуют операторы прекращения цикла и немедленного перехода к следующей итерации. В реализациях Паскаля этот недостаток устранён. В цикле For

значение управляющей переменной при выходе из цикла считается неопределённым. Это не позволяет определить явным образом, по окончании цикла, выполнен ли цикл полностью или завершён преждевременно.

В стандарте Паскаля в операторе CASE отсутствует альтернатива по умолчанию. В Паскале точка с запятой используется в качестве разделителя

операторов, а не завершителя. Средства работы с файлами, средства ввода – вывода в Паскале не всегда удобны. Отсутствуют средства передачи в программу параметров указательных в командной строке вызывающей команду на выполнение. Отсутствуют средства включения файла # include и макропроцессор # define, но заменой макропроцессора в большинстве случаев может служить раздел const. Большинство из указанных недостатков устранены в существующих трансляторах языка, в Delphi стандарт Паскаля значительно расширен.

3.1. Документирование программ

Так как время сопровождения и эксплуатации программных средств в общем случае превышает время разработки, нецелесообразно экономить время разработки за счёт написания документации. Качество программы во многом определяется качеством документации на неё, в том числе представленной в электронной форме.

Согласно ГОСТ 19.101-77 программы разделяются на виды:

Компонент- это программа, рассматриваемая как единое целое, выполняющая законченную функцию и применяемая самостоятельная или в составе комплекса.

Комплекс – программа, состоящая из двух или более компонентов и/или комплексов, выполняющих взаимосвязанные функции и применяемая самостоятельно или в составе другого комплекса.

К программным документам относят документы, содержащие сведенья необходимые для разработки, изготовления, сопровождения и эксплуатации программы:

- спецификация – это состав программы и документации на неё.
- ведомость держателей подлинника – это перечень предприятий, на которых хранят подлинники программных документов.
- текст программы
- описание программы – это сведенья о логической структуре и функционировании программы.
- программа и методика испытаний – это требования подлежащие проверке при испытании программы, а также порядок и методы их контроля.

Тестовые примеры должны составляться, как правило, лицом, знающим область разработки, но не участвующим непосредственно в разработке программы. Обычно это представители заказчика.

- техническое задание – это назначение и область применения программы, технические, технико-экономические и специальные требования, предъявляемые к программе. Необходимые стадии и сроки разработки, виды испытаний.
- пояснительная записка – это схема алгоритма, общее описание алгоритма и/или функционирования программы, а также обоснование принятых технических и технико-экономических решений.
- эксплуатационные документы:

а) ведомость эксплуатационных документов

б) формуляр – это основные характеристики программы, комплектность и сведенья об эксплуатации программы.

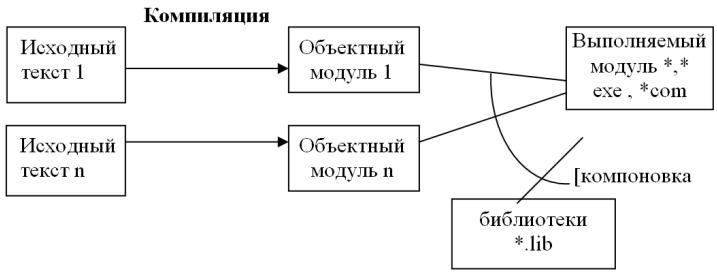
в) описание применения – это сведенья о назначении программы, области применения, применяемых методах, классе решаемых задач, ограничениях для применения минимальной конфигурации технических средств.

г) руководство системного программиста, оператора, описание языка.

д) руководство по техническому обслуживанию – это сведения для применения тестовых и диагностических программ, при обслуживании технических средств.

3.2. Этапы подготовки программ к выполнению

В общем случае программа собирается из некоторого количества блоков. Один из этих блоков тот, с которого начинается выполнение программы, т.е. он выполняет роль основной программы или конструкции запускающей приложения на выполнение.



Исходные тексты могут быть сформулированы на разных языках программирования в универсальных и специализированных текстовых редакторах поддерживающих простой текст. На этапе компиляции исходный текст переводится на язык машинных команд, оформляемый в виде объектного модуля (расширение объектных модулей*.obj). Иногда после компиляции получается текст программы на промежуточном языке, в этом случае требуется ещё один шаг компиляции для получения объектного модуля и с промежуточного языка. В препроцессоре производятся замены. **Независимая компиляция –** компиляция частей программы независимо друг от друга. На этапе компоновки (редактирование связей) происходит объединение объектных модулей с модулями содержащихся в библиотеках, в результате чего получается выполняемый модуль (чаще имеет расширение *.exe). Данная схема подразумевает, что исходные тексты могут быть на разных языках программирования, но для практического использования такой схемы обязательно выполнение условия соответствия форматов объектных модулей (можно встроить программы одну в другую). В настоящее время данная схема чаще всего реализована в виде интегрированных сред, объединяющее действие по формированию

исходного текста, компиляции, компоновки, запуска на выполнение и откладки программы.

3.2.1. Особенности реализации независимой компиляции в Turbo Delphi

Для обеспечения этого механизма введено понятие unit ,структура его:
Unit Uname
Interface
Interface-описание объектов, видимых в других модулях
Implementation-описание внутренних объектов модуля или скрытых
Begin
… - операторы инициализации объектов модуля
End

При использовании одного модуля в другом модуле указывается список uses,

Например: uses func, matrix

Главная цель программирования – получение результата, а не написание программ.

3.3. Основы компиляции

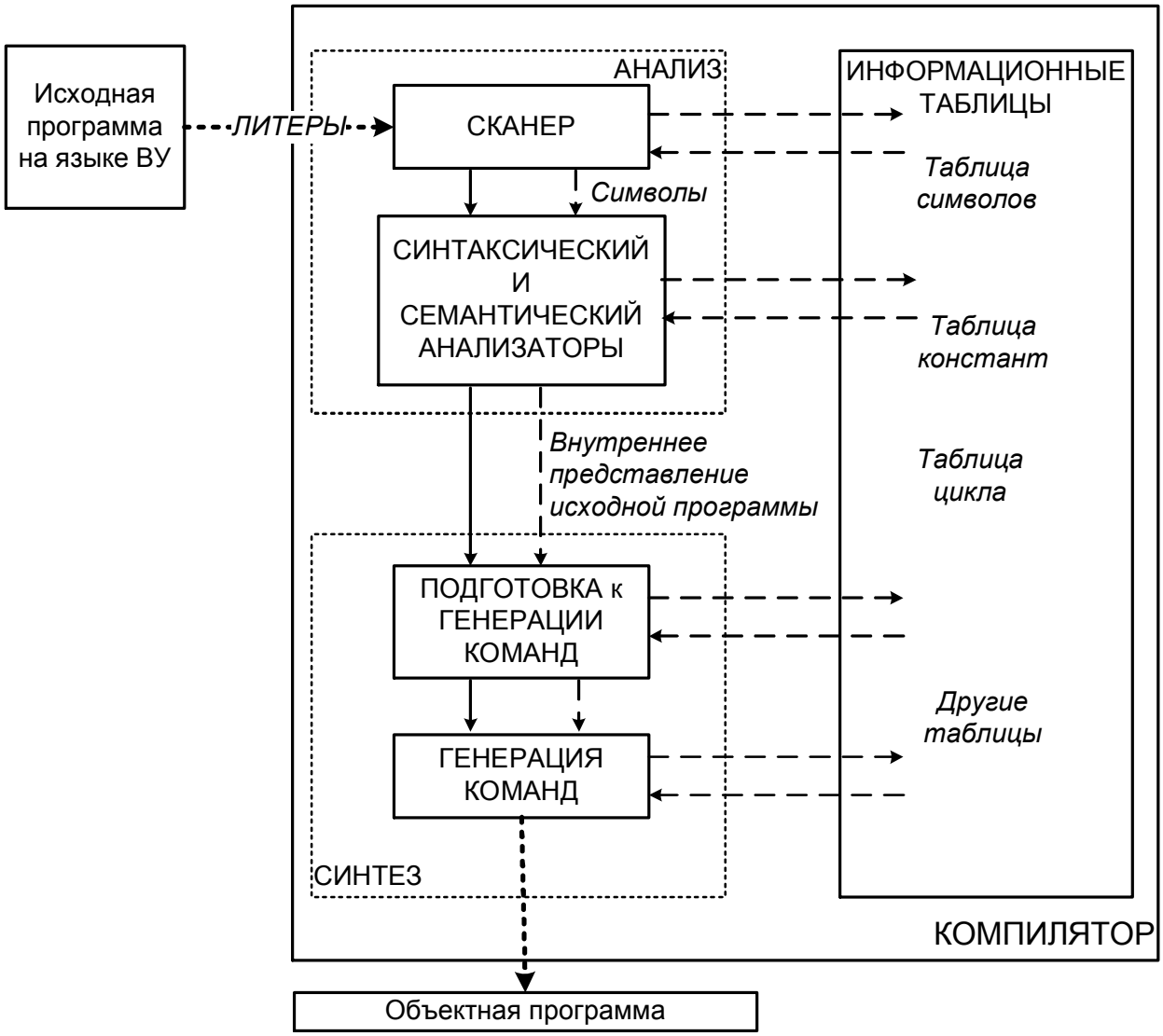
Программы, написанные на языках программирования высокого уровня перед выполнением на ЭВМ должны транслироваться в эквивалентные программы, написанные на машинном коде. Далее рассматривается процесс полуения объектного кода из исходной программы по данным сайта «Структуры и алгоритмы» (режим доступа http://www.structur.h1.ru/biblio.htm).

Транслятор - это программа, которая переводит исходную программу в эквивалентную ей объектную программу.

Если исходный язык является языком высокого уровня, например таким, как Паскаль, C++, и если объектный язык - автокод, то такой транслятор называется компилятором.

Интерпретатор для некоторого исходного языка принимает исходную программу, написанную на этом языке, как входную информацию и выполняет ее. Различие между компилятором и интерпретатором состоит в том, что интерпретатор не порождает объектную программу, которая затем должна выполняться, а непосредственно выполняет ее сам.

Для того чтобы выяснить, как осуществить выполнение инструкций исходной программы, чистый интерпретатор анализирует ее всякий раз, когда она должна быть выполнена.



При программировании интерпретатор обычно разделяют на две фазы. На- первой фазе интерпретатор анализирует всю исходную программу, почти так же, как это делает компилятор, и транслирует ее в некоторое внутреннее представление. На второй фазе это внутреннее представление исходной программы интерпретируется или выполняется. Внутреннее представление исходной программы разрабатывается для того, чтобы свести к минимуму время, необходимое для расшифровки или анализа каждой инструкции при ее выполнении.

Для компиляции компилятор должен выполнить анализ исходной программы,, а затем синтез объектной программы.

Сначала исходная программа разлагается на ее составные части; затем из них строятся части эквивалентной объектной программы. Для этого на этапе анализа компилятор строит несколько таблиц, которые используются затем как при анализе, так и при синтезе.

Информационные таблицы.

При анализе программы из описаний, заголовков процедур, заголовков циклов и т.д. извлекается информация и сохраняется для последующего использования. Эта информация обнаруживается в отдельных точках

программы и организуется так, чтобы к ней можно было обратиться из любой части компилятора. Например, при каждом использовании идентификатора необходимо знать, как был описан этот идентификатор и как он использовался в других местах программы. Что конкретно следует хранить, зависит, конечно, от исходного языка, объектного языка и сложности компилятора. Но в каждом компиляторе в той или иной форме используется таблица символов (иногда ее называют списком идентификаторов или таблицей имен). Это таблица идентификаторов, встречающихся в исходной программе, вместе с их атрибутами. К атрибутам относятся тип идентификатора, его адрес в объектной программе и любая другая информация о нем, которая может понадобиться при генерации объектной программы.

Сканер

Сканер - часть компилятора, иногда также называемая лексическим анализатором просматривает литеры исходной программы слева направо и строит символы программы - целые числа, идентификаторы, служебные слова и т. д. (Символы передаются затем на обработку фактическому анализатору.) На этой стадии может быть исключен комментарий. Сканер также может заносить идентификаторы в таблицу символов и выполнять другую простую работу, которая фактически не требует анализа исходной программы. Он может выполнить большую часть работы по макрогенерации в тех случаях, когда требуется только текстовая подстановка.

Обычно сканер передает символы анализатору во внутренней форме. Каждый разделитель (служебное слово, знак операции или знак пунктуации) будет представлен целым числом. Идентификаторы или константы можно представить парой чисел. Первое число, отличное от любого целого числа, использующегося для представления разделителя, характеризует сам "идентификатор" или "константу"; второе число является адресом или индексом идентификатора или константы в некоторой таблице. Это позволяет в остальных частях компилятора работать эффективно, оперируя с символами фиксированной длины, а не с цепочками литер переменной длины.

Синтаксический и семантический анализаторы

Эти анализаторы выполняют действительно сложную работу по расчленению исходной программы на составные части, формированию ее внутреннего представления и занесению информации в таблицу символов и другие таблицы. При этом также выполняется полный синтаксический и семантический контроль программы.

Синтаксис - способ соединения слов (и их форм) в словосочетания и предложения.

Семантика - это значения единиц языка.

Обычный анализатор представляет собой синтаксически управляемую программу. В действительности стремятся отделить синтаксис от семантики настолько, насколько это возможно. Когда синтаксический анализатор (распознаватель) узнает конструкцию исходного языка, он вызывает

соответствующую семантическую процедуру, или семантическую программу, которая контролирует данную конструкцию с точки зрения семантики и затем запоминает информацию о ней в таблице символов или во внутреннем представлении программы. Например, когда распознается описание переменных, семантическая] программа проверяет идентификаторы, указанные в этом описании, чтобы убедиться в том, что они не были описаны дважды, и заносит их вместе с атрибутами в таблицу символов.

Когда встречается инструкция присваивания вида <переменная>:= <выражение> семантическая программа проверяет переменную и выражение на соответствие типов и затем заносит инструкцию присваивания в программу во внутреннем представлении.

Внутреннее представление исходной программы

Внутреннее представление исходной программы в значительной степени зависит от его дальнейшего использования. Это может быть дерево, отражающее синтаксис исходной программы. Это может быть исходная программа, в так называемой польской записи, список тетрад и т.д.

Подготовка к генерации команд

Перед генерацией команд обычно необходимо некоторым образом обработать и изменить внутреннюю программу. Кроме того, должна быть распределена память под переменные готовой программы. Одним, из важных моментов на этом этапе является оптимизация программы с целью уменьшения времени ее работы.

Генерация команд

По существу, на этом этапе происходит перевод внутреннего представления исходной программы на автокод или на машинный язык. Это, по-видимому, наиболее хлопотная и кропотливая часть компилятора, хотя и наиболее понятная.

В интерпретаторе эта часть компилятора заменяется программой, которая фактически выполняет (или интерпретирует) внутреннее представление исходной программы. Причем само внутреннее представление в этом случае мало чем отличается от того, которое получается при компиляции.

Главные трудности реализации компилятора заключаются в стадиях семантического анализа, программы подготовки генерации и программы генерации команд. Эти три части взаимозависимы, должны в значительной степени разрабатываться совместно и могут коренным образом измениться при переходе с одного объектного языка на другой или с одной машины на другую.

Раздел II. Программное обеспечение

Успешное освоение и дальнейшая практическая работа с использованием информационных технологий предполагает наличие программных технологий, основанных на использовании средств сбора, передачи, обработки, хранения, представления информации в процессе управленческой деятельности.

1. Общая характеристика и классификация программных средств.

Программное обеспечение информационных систем можно условно разбить на три большие группы (см. рис.).

2. Системные и прикладные программные средства.

Системные программные средства предназначены для обеспечения деятельности компьютерных систем как таковых. В их составе выделяют:

тестовые и диагностические программы;

антивирусные программы;

операционные системы;

командно-файловые процессоры (оболочки).

Тестовые и диагностические программы.

Тестовые и диагностические программы предназначены для проверки работоспособности отдельных узлов компьютера и компонентов программно-файловых систем и, возможно, устранения выявленных неисправностей.

Антивирусные программы.

Антивирусные программы предназначены для выявления и, возможно, устранения вирусных программ, нарушающих нормальную работу вычислительной системы.

Операционные системы.

Операционные системы являются основными системными программными комплексами, выполняющими следующие основные функции:

тестирование работоспособности вычислительной системы и ее настройку при первоначальном включении;

обеспечение синхронного и эффективного взаимодействия всех аппаратных и программных компонентов вычислительной системы в процессе ее функционирования;

обеспечение эффективного взаимодействия пользователя с вычислительной системой.

Операционные системы классифицируются следующим образом:

однопользовательские однозадачные системы (MS-DOS);

однопользовательские многозадачные системы (OS/2, Windows);

многопользовательские системы (системы семейства UNIX, Windows, Linux).

Командно-файловые процессоры

Командно-файловые процессоры (оболочки) предназначены для организации системы взаимодействия пользователя с вычислительной системой на принципах, отличных от реализуемых операционной системой, с целью облегчения его работы или предоставления дополнительных возможностей (например, Norton Commander или Windows).

Прикладные программные средства обеспечения управленческой деятельности классифицируются следующим образом:

системы подготовки текстовых документов;

системы обработки финансово-экономической информации;

системы управления базами данных;

личные информационные системы;

системы подготовки презентаций;

системы управления проектами;

экспертные системы и системы поддержки принятия решений;

системы интеллектуального проектирования и совершенствования систем управления;

прочие системы.

Системы подготовки текстовых документов предназначены для организации технологии изготовления управленческих документов и различных информационных материалов текстового характера. Они включают в себя:

текстовые редакторы;

текстовые процессоры;

настольные издательские системы.

Системы обработки финансово-экономической информации предназначены для обработки числовых данных, характеризующих различные производственно-экономические и финансовые явления и объекты, и составления соответствующих управленческих документов и информационно-аналитических материалов. Они включают в себя:

универсальные табличные процессоры;

специализированные бухгалтерские программы;

специализированные банковские программы (внутрибанковских и межбанковских расчетов);

специализированные программы финансово-экономического

анализа и планирования.

Системы управления базами данных предназначены для создания, хранения и манипулирования массивами данных большого объема. Разные системы этого класса различаются способами организации хранения данных и обработки запросов на поиск информации, а также характером хранящихся в базе данных.

Личные информационные системы предназначены для информационного обслуживания рабочего места управленческого работника и по существу выполняют функции секретаря.

Системы подготовки презентаций предназначены для квалифицированной подготовки графических и текстовых материалов, используемых в целях демонстрации на презентациях, деловых переговорах, конференциях. Для современных технологий подготовки презентаций характерно подключение к традиционным графике и тексту таких форм информации, как видео- и аудиоинформация, что позволяет говорить о реализации гипермедиатехнологий.

Системы управления проектами предназначены для планирования и управления ресурсами различных видов (материальными, техническими, финансовыми, кадровыми, информационными) при реализации сложных научно-исследовательских и проектно-строительных работ.

Экспертные системы и системы поддержки принятия решений предназначены для реализации технологий информационного обеспечения процессов принятия управленческих решений на основе применения экономико-математического моделирования и принципов искусственного интеллекта.

Системы интеллектуального проектирования и совершенствования систем управления предназначены для использования так называемых CASE-технологий (Computer Aid System Engineering), ориентированных на автоматизированную разработку проектных решений по созданию и совершенствованию систем организационного управления.

2.1.Базовые информационные технологии обеспечения управления ***информационными ресурсами***

Как уже указывалось, информационная технология'-предполагает наличие трех основных компонентов: комплекса технических средств, системы управления им и организационно-методического обеспечения. При этом в составе комплекса технических средств доминирующим компонентом, определяющим основные возможности технологии, является вычислительная система.

Но любая вычислительная система не может рассматриваться только как набор определенных технических элементов, поскольку функционирует и решает задачи пользователя лишь под управлением соответствующих программ, входящих в состав второго компонента информационной технологии. Поэтому в данном разделе будут рассматриваться информационные технологии в соответствии с используемыми средствами прикладного программного обеспечения.

Рассмотренные в 2 прикладные программные средства, с одной стороны, не покрывают все без исключения потребности управления информационными ресурсами в силу чрезвычайной динамичности этой области управленческой деятельности, а с другой стороны, не являются равнозначными в смысле реального использования в подавляющем большинстве случаев.

Следствием этого положения вещей стало появление так называемых офисных пакетов комплексных программных систем, обеспечивающих реализацию основных, или базовых, технологий управления информационными ресурсами, к которым относят:

технологии подготовки текстовых документов;

технологии обработки финансово-экономической информации и подготовки табличных документов;

технологии управления базами данных;

технологии личных информационных систем.

1. Технологии подготовки текстовых документов

Обработка текстов как направление развития техники возникло в первом десятилетии XX в. с появлением механической пишущей машинки. Затем более полувека пишущая машинка оставалась единственным общедоступным средством получения печатного текста на бумаге. Очевидно, что при печатании на пишущей машинке наиболее трудоемким является процесс внесения изменений в текст, когда в лучшем случае с помощью ножниц и клея создается новый вариант документа, который затем весь перепечатывается заново для получения чистового варианта. В процессе печати опечатки замазываются или исправляются подчисткой и повторным впечатыванием. Наибольшие усовершенствования пишущей машинки, такие, как возможность печати часто повторяющихся текстов с помощью механического читающего устройства с закодированными перфорацией знаками, принципиально не изменили процесса подготовки текста.

1.1. Развитие систем и технологий подготовки текстовых документов

Первый революционный шаг в области обработки текстов был сделан фирмой IBM в 1964 г., когда она выпустила систему под названием MT/ST (Magnetic Tape/Selectric Typewriter), представлявшую собой пишущую машинку с записывающим устройством, которое позволяло записывать вводимый с пишущей машинки текст на кассету с магнитной лентой, после чего можно было найти в тексте нужное место, вставить коррекцию, удалить часть текста или повторить часть текста без повторного ввода с клавиатуры. Позже магнитная лента была заменена магнитными картами, каждая из которых содержала страницу текста и была удобнее, чем магнитная лента, для хранения и поиска текста.

В начале 70-х годов фирмы Lexitron и 3M разработали текстовые системы с видеодисплеями, позволявшие видеть вводимый с клавиатуры текст на экране и вносить изменения, тотчас же отображавшиеся на экране. В 1973 г. текстовые системы уже снабжались устройствами записи текста на гибких дисках, позволявшими к любой части текста иметь прямой доступ, а не последовательный, как на магнитной ленте. Скорость работы существенно повысилась.

Первые электронные текстовые программы были громоздкими и дорогими, однако с появлением микропроцессора и персональных компьютеров на их основе они стали широко доступными. В 1980-е годы было разработано множество текстовых программ для различных персональных компьютеров, отличающихся как функциональными возможностями, так и интерфейсом с пользователями. В последние несколько лет получили распространение текстовые программы с такими возможностями, что их можно считать настольными издательскими системами, позволяющими выполнять не только ввод и редактирование

текста, но и верстку в интерактивном режиме сложного текста с иллюстрациями.

1.2. Выбор системы подготовки текстовых документов

Существующие в настоящее время системы подготовки текстовых документов значительно отличаются друг от друга характеристиками, возможностями по вводу и редактированию текста, его форматированию и выводу на печать, а также по степени сложности освоения пользователем.

Выбор конкретного программного продукта для обработки текста является весьма ответственным моментом. Разнообразные системы подготовки текстов позволяют эффективно использовать компьютер тем специалистам, которые связаны с информационными технологиями. Процесс выбора обусловлен многими факторами, но прежде всего необходимо использовать принцип разумной достаточности. Многообразие пакетов программ позволяет остановиться на том, который обеспечит максимальное удовлетворение потребностей подготовки текстов программ или документов в условиях наилучшего использования ограничивающего фактора, особенно при подготовке текстов больших объемов.

Наиболее важной для практического пользователя характеристикой программы этого класса могла бы выступить область профессиональной деятельности, для которой программный продукт удобен в применении. Инструментальные средства подготовки текстовых документов используются для набора текстов программ, документов различной степени сложности, научных статей, книг и т.д. Ограничивающим фактором может быть квалификация пользователя.

Наиболее актуальным при описании процесса редактирования является понятие интерфейса пользователя, к которому в первую очередь относятся язык общения с текстовым процессором, а также устройства ввода-вывода (для ПК — это клавиатура, манипулятор типа «мышь», экран дисплея и принтер). Современные системы подготовки текстовых документов в большинстве своем обладают дружественным пользовательским интерфейсом.

Однако раз аботчики программ обработки текстов учитывают тот факт, что у каждого пользователя свой стиль работы над документом (что удобно для одной группы пользователей, для другой является помехой). Поэтому наиболее привлекательными для разработчика документа выглядят те программные среды, в которых возможна настройка интерфейса под свои вкусы и потребности. С точки зрения удобства для пользователя одним из важнейших свойств текстовых процессоров является полное соответствие твердой копии образу документа на экране. Такая характеристика по-английски называется WYSIWYG (What You See Is What You Get — что Вы видите, то и получите). Не последнюю роль при выборе текстового процессора играют объем занимаемой памяти (особенно при его

использовании в составе систем автоматизированного проектирования) и цена.

1.3. Классификация систем подготовки текстовых документов

Существующие в настоящее время компьютерные системы подготовки текстовых документов (СПТД) можно классифицировать по объему функциональных возможностей или по назначению для применения:

текстовые редакторы;

текстовые процессоры;

настольные издательские системы.

1.4. Текстовые редакторы

Редактор текстов (text editor) обеспечивает ввод, изменение и сохранение любого символьного текста, но предназначен в основном для подготовки текстов, которые в конечном итоге потребляются программами, поскольку текст программы не требует форматирования, т.е. автоматического преобразования расположения элементов текста, изменения шрифта и т.п. Программный текст исторически первым стал обрабатываться с помощью компьютера.

Набор операций текстовых редакторов определяет особенности построчной записи текстов на языках программирования, хотя этот набор и весьма широк.

Результатом работы текстового редактора является файл, в котором все знаки являются знаками кода ASCII (ASCII — American Standard Code for Information Interchange— Американский стандартный код для обмена информацией) и не содержат знаки, интерпретация которых специфична для данного редактора.

Различаясь способами управления и набором сервисных возможностей, текстовые редакторы в том или ином виде позволяют:

- набирать текст на экране, используя до 200 символов;
- исправлять ошибочные символы в режиме замены;
- вставлять и удалять группы символов (слова) в пределах строки, не переводя неизменившуюся часть строки, а сдвигая ее влево/вправо целиком в режиме вставки;
- удалять одну или несколько строк, копировать их или перемещать в другое место текста;
- раздвигать строки существующего текста, чтобы вставить туда новый фрагмент;
- вставлять группы строк из других текстов;
- обнаруживать все вхождения определенной группы символов (контекста);
- заменять один контекст другим, возможно разной длины;

- сохранять набранный текст для последующих коррекций;

- печатать текст на разных типах принтеров стандартными программами печати одним шрифтом в пределах документа.

Легко видеть, что использование для подготовки и печати документа редактора текстов на качественном уровне соответствует использованию пишущей машинки, разве что более производительному из-за легкости повторения печати с хранимой в электронной памяти заготовки и возможности как исправления опечаток, так и частичной переработки текста путем вставки или исключения фрагментов.

Подобно пишущей машинке, редактор текстов не выделяет особо символ пробел, т.е. ему безразлично машинное представление этого символа — его код.

Таким образом, использование редактора текстов для подготовки печатных документов - это использование программного продукта не по прямому назначению.

1.5. Текстовые процессоры

Когда предметной областью пользователя является подготовка текстов на естественных языках для печати и печать этих документов, набор операций редактора должен быть существенно расширен и программный продукт переходит в новое качество — систему подготовки текстов — продукт, которому соответствует англоязычный термин word processor. Программы для обработки документов ориентированы на работу с текстами, имеющими структуру документа, т.е. состоящими из абзацев, страниц и разделов.

Среди систем подготовки текстов на естественных языках можно выделить три больших класса, но с достаточно размытыми границами: форматеры, текстовые процессоры и настольные издательства. Исходя из внутримашинной структуры подготавливаемого документа, можно было бы предложить следующий подход к классификации систем подготовки текстов. Форматер — система подготовки текстов, которая не использует для внутреннего представления текста никаких кодов, кроме стандартных: конец строки, перевод каретки, конец страницы. Текстовый процессор — система подготовки текстов, которая во внутреннем представлении снабжает собственно текст специальными кодами — разметкой. В основном редакторы и текстовые процессоры различаются по назначению: первые создают файлы, которые используются затем компиляторами или форматерами, вторые — предназначены для подготовки текстов для последующей печати на бумаге; форма представления текста имеет большое значение.

Текстовые процессоры имеют специальные функции, которые предназначены для облегчения ввода текста и представления его в напечатанном виде. Среди этих функций можно выделить следующие:

ввод текста под контролем функций форматирования, обеспечивающих немедленное изменение вида страницы текста на экране и расположение слов

на ней, давая приближенноепредставление о действительном расположении текста на бумаге после печати (такая возможность обычно обеспечивается после предварительной настройки текстового процессора на принтер, на котором предполагается печатать текст);

возможность предварительного описания структуры будущего документа с помощью специального языка; в этом описании задаются такие параметры, как величина абзацных отступов,

тип и размер шрифта для различных элементов текста, расположение заголовков, междустрочные расстояния, число колонок текста, расположение и способ нумерации сносок (в конце текста или на той же странице) и т.п.; чтобы воспользоваться этим описанием, при вводе текста обычно нужно последовательно нажать на определенные клавиши, чтобы сообщить текстовому процессору, какой элемент текста вводите (заголовок, или стандартный параграф, или сноску); комбинацию клавишей для указания каждого элемента текста выбирает пользователь;

возможность автоматической проверки орфографии и получения подсказки при выборе синонимов;

возможность ввода и редактирования таблиц и формул с отображением их на экране в том виде, в каком они будут напечатаны;

возможность объединения документов в процессе подготовки текста к печати;

возможность автоматического составления оглавления и алфавитного справочника.

Большинство текстовых процессоров имеет средства настройки на конфигурацию оборудования компьютера, в частности на тип графического адаптера и видеомонитора. Поэтому возможности представления текста на экране сильно зависят от разрешающей способности видеомонитора.

Проблема совместимости различных текстовых процессоров.

Практически все текстовые процессоры имеют уникальную структуру данных для представления текста, что объясняется необходимостью включения в текст дополнительной информации, описывающей структуру документа, шрифты и тому подобное, поскольку каждое слово или даже символ могут иметь свои особые характеристики. Поэтому текст, подготовленный с помощью одного текстового процессора, как правило, не может быть прочитан другими текстовыми процессорами и, следовательно, не может быть отредактирован и напечатан. В целях совместимости текстовых документов при переносе их из среды одного текстового процессора в другой существует особый вид программного обеспечения — конверторы, обеспечивающие получение выходного файла в формате текстового процессора получателя документа.

Существующие в настоящее время текстовые процессоры значительно отличаются друг от друга характеристиками, возможностями по вводу и редактированию текста, его форматированию и выводу на печать, а также по степени сложности освоения пользователем. Достаточно условно эти инструментальные средства могут быть разделены на две категории.

Виды текстовых процессоров. К первой категории можно отнести текстовые процессоры, позволяющие подготовить и напечатать сложные и большие по объему документы, включая книги.

Текстовые процессоры второй категории имеют существенно меньшие возможности, однако проще в использовании, работают быстрее и требуют меньше оперативной памяти, существенно ниже по стоимости. В список текстовых процессоров первой категории можно включить Word.

1.6. Настольные издательские системы

Настольные издательства готовят тексты по правилам полиграфии и с типографским качеством. Подобно тому как текстовые процессоры не являются «развитием» форматеров, настольные издательства не являются более совершенным продолжением текстовых процессоров, так как у них совсем иная предметная область.

Пакеты программ настольного издательства {desktop publishing, пакеты DTP или НИС) по сути являются инструментом верстальщика. Предназначены программы этого класса не столько для создания больших документов, сколько для реализации различного рода полиграфических эффектов. То есть программа настольного издательства позволяет легко манипулировать текстом, менять форматы страниц, размер отступов, дает возможность комбинировать различные шрифты, работать с материалом до получения полного удовлетворения от внешнего вида как отдельных страниц (полос издания), так и всего издания.

По ряду функциональных возможностей пакеты НИС аналогичны лучшим текстовым процессорам, и граница, разделяющая их, становится все незаметнее.

Так, оба типа программ позволяют размещать на одной странице текст и иллюстрации, формировать текст в несколько колонок, могут быть использованы для редактирования текста и манипулирования текстовыми блоками. Причем в текстовых процессорах тот же эффект может достигаться значительно более простым и быстрым путем. Это связано с тем, что они являются существенно более простым инструментом, чем пакеты НИС.

Но пакеты НИС отличаются от текстовых процессоров еще двумя важными характеристиками. Во-первых, пакеты НИС имеют более широкие возможности управления подготовкой текста. Вовторых, подготовленные в пакете НИС материалы выглядят изданиями высшего уровня качества, а не просто как изящные распечатки.

Если текстовые процессоры ориентированы на работу со словами и абзацами, то пакеты НИС позволяют легко и элегантно манипулировать текстом до уровня отдельных символов в слове.

Чтобы облегчить процесс подготовки страницы, в пакетах НИС предусмотрена возможность вывода на экран монитора точной копии того, что будет распечатано на принтере, — напомним, что эта характеристика

называется по-английски WYSIWYG. В пакете DTP можно просмотреть в увеличенном виде любой подозрительный участок страницы.

Все пакеты имеют характеристики, отсутствующие в абсолютном большинстве текстовых процессоров, например сжатие и растяжение строк, вращение текста и изменение расстояний междустрочками и абзацами с очень маленьким шагом приращения и т.д.

Внешний файл, подготовленный текстовым процессором, можно распечатать только этим же текстовым процессором. Как правило, печать может быть выполнена на принтере любого типа, в том числе и на лазерном. Тексты, подготовленные настольными издательствами, распечатываются только на лазерных принтерах.

Среди систем подготовки текстовых документов в этом классе можно также предложить деление на две подгруппы: настольные издательства профессионального уровня и издательские системы начального уровня. Системы первой подгруппы предназначены для работы над изданиями документов со сложной структурой или типа иллюстрированного журнала К системам профессионального уровня можно отнести QuarkXPress, FrameMaker, PageMaker. Однако освоение дорогих и сложных в эксплуатации «настольных типографий» обычно требует значительных временных затрат, поэтому вряд ли их целесообразно использовать тем специалистам, которым по роду занятий лишь изредка требуется красиво и довольно быстро подготовить документацию, письмо или объявление.

Системы второй подгруппы обычно не предназначаются для получения промышленной полиграфической продукции. Пользователи данного класса НИС для решения своих задач, как правило, применяют другие программы, а НИС используют эпизодически, например при создании информационного бюллетеня или формировании поздравительной открытки для тиражирования в небольшой фирме. Все пакеты данной категории ориентируются на новичка и пользователя, который отдает издательской деятельности лишь часть своего рабочего времени. Наиболее распространены в этой группе Microsoft Publisher, Pageplus for Windows.

Самый очевидный параметр оценки уровня НИС — ее цена.

Цена некоторых систем профессионального уровня в десяток раз превышает цену минимальных программ. Однако это еще не все,— пользователь должен оценить, что ему реально требуется, насколько полно программа соответствует его нуждам, сколь быстро он может ее освоить и достаточно ли она удобна в работе.

2. Технологии обработки финансовоэкономической информации и подготовки табличных документов

Технологии обработки финансово-экономической информации базируются на следующих прикладных программных продуктах:

универсальные табличные процессоры;

специализированные программы бухгалтерских расчетов;

специализированные банковские программы;

специализированные программы финансово-экономического анализа и бизнес-планирования.

Перечисленные специализированные программы имеют область применения, ограниченную их возможностями, и в дальнейшем не рассматриваются, так как универсальные табличные процессоры в рамках своих возможностей могут выполнять функции специализированных программ.

2.1. Общая характеристика программных средств подготовки табличных документов

При решении различных экономических, финансовых и других задач в управленческой деятельности приходится представлять и обрабатывать информацию в табличной форме в виде разного рода таблиц, бланков ведомостей, форм, списков. Создание и обработка табличных документов на основе использования средств вычислительной техники первоначально реализовывались двумя способами:

данные размещаются в таблице на бумаге, а их обработка производится с помощью электронного калькулятора;

данные размещаются в памяти компьютера, а для их обработки создается и используется программа на одном из языков программирования.

Первый способ хорош тем, что он рассчитан на самого рядового пользователя, нагляден, легко проверяется, но расчеты выполняются очень медленно.

Второй способ, естественно, во много раз быстрее, но есть свой недостаток — он не подходит для неподготовленного пользователя, требует работы квалифицированного программиста. Сам заказчик не будет непосредственно выполнять разработку и отладку таких программ.

Средством разрешения этого противоречия при проведении расчетов над данными, представленными в табличной форме, явились пакеты прикладных программ для работы с электронными таблицами, так называемые табличные процессоры, или SPREAD SHEETS, которые в простой и естественной форме соединяют преимущества обоих способов и благодаря этому получили широкое распространение.

Организация подготовки табличных документов основана на двух основных категориях:

форма представления данных на экране монитора в виде таблицы практически неограниченного размера (собственно электронная таблица как объект обработки);

программа (или пакет программ) для обработки таких данных (собственно табличный процессор как инструмент обработки).

Диапазон возможных применений современных табличных процессоров невероятно широк: они позволяют не только безошибочно проводить операции над числами в столбцах и строках, но и строить на

основе табличных данных диаграммы, производить сложный финансово-экономический анализ, автоматизировать различные сферы бухгалтерской и экономической деятельности.

Табличный процессор является обязательной составляющей любого интегрированного пакета или офисной системы. Очевидно, что возможностей у такой составляющей несколько меньше, но она обеспечивает решение таловых задач.

С появлением операционной системы Windows были разработаны версии самых популярных табличных процессоров, ориентированных на работу в этой среде:

Следует отметить, что безусловным лидером (по объему продаж, а следовательно, по популярности, которой она пользуется у пользователей) среди программ этого класса является система Excel.

2.2. Функциональные возможности табличных процессоров

Современные табличные процессоры имеют очень широкие функциональные и вспомогательные возможности, обеспечивающие удобную и эффективную работу пользователя. Перечислим основные такие возможности, общие для всех систем этого класса.

1. Контекстная подсказка (Help). Вызывается из контекстного меню или нажатием соответствующей кнопки в пиктографическом меню.

2. Справочная система. Организована в виде гипертекста и позволяет легко и быстро осуществлять поиск нужной темы.

3. Многовариантность выполнения операций. Практически все операции могут быть выполнены одним из трех-четырех способов, пользователь выбирает наиболее удобный.

4. Контекстное меню (Shortcut menu). Разворачивается по щелчку правой кнопки «мыши» на объекте. Речь идет, например, о месте таблицы, где в данный момент хочет работать пользователь. Наиболее часто используемые функции обработки, доступные в данной ситуации, собраны в контекстном меню.

5. Пиктографическое меню. Наиболее часто используемым командам соответствуют пиктограммы, расположенные под строкой меню. Они образуют пиктографическое меню. Вследствие щелчка мышью на пиктограмме выполняется связанная с ней команда. Пиктографические меню могут быть составлены индивидуально.

6. Рабочие группы или рабочие папки (Workbook). Документы можно объединять в рабочие папки, так что они могут рассматриваться как единое целое, если речь идет о копировании, загрузке, изменении или других процедурах. В нижней части электронной таблицы расположен алфавитный указатель (регистр), который обеспечивает доступ к рабочим листам.

Пользователь может задавать название листам в папке (вместо алфавитного указателя), что делает наглядным содержимое регистра, а значит, облегчает поиск и переход от документа к документу.

7. Средства для оформления и модификации экрана и таблиц. Внешний вид рабочего окна и прочих элементов экранного интерфейса может быть определен в соответствии с требованиями пользователя, что позволяет сделать работу максимально удобной. Среди таких возможностей — разбиение экрана на несколько окон, фиксация заголовков, строк и столбцов и т.д.

8. Средства оформления и вывода на печать таблиц. Для удобства пользователя предусмотрены все функции, обеспечивающие печать таблиц, — такие, как выбор размера страницы, разбиение на страницы, установка размера полей страниц, оформление колонтитулов, а также предварительный просмотр получившейся страницы.

9. Средства оформления рабочих листов. Современные системы предоставляют широкие возможности по форматированию таблиц, такие, как: выбор шрифта и стиля, выравнивание данных внутри клетки, возможность выбора цвета фона клетки и шрифта, возможность изменения высоты строк и ширины колонок, черчение рамок различного вида, возможность задания формата данных внутри клетки (например, числовой, текстовый, финансовый, дата и т.д.), а также автоформатирование — в систему уже встроены различные возможности оформления таблиц и пользователь может выбрать наиболее подходящий формат из уже имеющихся.

10. Шаблоны. Табличные процессоры, как и текстовые, позволяют создавать шаблоны рабочих листов, которые применяются для создания бланков писем и факсов, различных калькуляций. Если шаблон создается для других пользователей, то можно разрешить заполнение таких бланков, но запретить изменение формы бланка.

11. Связывание данных. Абсолютная и относительная адресации являются характерной чертой всех табличных процессоров, в современных системах они дают возможность работать одновременно с несколькими таблицами, которые могут быть тем или иным образом связаны друг с другом. Например, трехмерные связи, позволяющие работать с несколькими листами, идущими подряд; консолидация рабочих листов, с ее помощью можно обрабатывать суммы, средние значения и вести статистическую обработку, используя данные разных областей одного рабочего листа, нескольких рабочих листов и даже нескольких рабочих книг; связанная консолидация позволяет не только получить результат вычислений по нескольким таблицам, но и динамически его пересчитывать в зависимости от изменения исходных значений.

12. Вычисления. Для удобства вычисления в табличных процессорах имеются встроенные функции, а именно: математические, статистические, финансовые, функции даты и времени, логические и др. Менеджер функций позволяет выбрать нужную функцию и, подставив значения, получить результат.

13. Деловая графика. Трудно представить современный табличный процессор без возможности построения различного типа двумерных,

трехмерных и смешанных диаграмм; насчитывается более 20 различных типов и подтипов диаграмм, которые можно построить в современной системе данного класса.

А возможности оформления диаграмм также многообразны и доступны, например, такие, как: вставка и оформление легенд, меток данных; оформление осей — возможность вставки линий сеток и др. Помимо этого, современные системы работы с электронными таблицами снабжены такими мощными средствами построения и анализа деловой графики, как вставка планок погрешностей, возможность построения трендами выбор функции линии тренда.

14. Выполнение табличными процессорами функций баз данных. Эта возможность обеспечивает заполнение таблиц аналогично заполнению базы данных, т.е. через экранную форму: защиту данных, сортировку по ключу или по нескольким ключам, обработку запросов к базе данных, создание сводных таблиц.

Кроме того, все современные программы работы с электронными таблицами включают средства обработки внешних баз данных, которые позволяют работать с файлами, созданными, например, в формате dBase или PARADOX или других форматах.

15. Моделирование. Подбор параметров и моделирование — одни из самых важных возможностей табличных процессоров. С помощью простых приемов можно находить оптимальные решения для многих задач. Методы оптимизации варьируют от простого подбора (при этом значения ячеекпараметров изменяются так, чтобы число в целевой ячейке стало равным заданному) до метода линейной оптимизации со многими переменными и ограничениями. При моделировании иногда желательно сохранять промежуточные результаты и варианты поиска решения. Это можно делать, создавая сценарии, которые представляют собой описание решаемой задачи.

16. Макропрограммирование. Для автоматизации выполнения часто повторяемых действий можно воспользоваться встроенным языком программирования макрокоманд. Разделяют макрокоманды и макрофункции. Применяя макрокоманды, можно упростить работу с табличным процессором и расширить список его собственных команд. При помощи макрофункций можно определять собственные формулы и функции, расширив таким образом набор функций, предоставляемый системой. В простейшем случае макрос — это записанная последовательность нажатия клавиш, перемещений и щелчков кнопками мыши. Эта последовательность может быть «воспроизведена» как магнитофонная запись. Ее можно обработать и каким-то образом изменить. Например, организовать цикл, переход, подпрограмму. Современные программы обработки электронных таблиц позволяют пользователю создавать и использовать диалоговые окна, которые по своему внешнему виду и удобству работы не отличаются от существующих в системе, что делает диалог с макрокомандой максимально удобным.

3. Технологии управления базами данных

Важнейшие функции систем информационной поддержки управленческой деятельности — хранение и обработка данных — не были реализованы возможностями систем управления файлами, существовавшими в 60-е годы; отсутствовали поддержание логически связанных файлов, средства восстановления данных в системе после сбоев и параллельная работа нескольких пользователей; не был реализован язык манипулирования данными.

3.1. Определение и классификация современных систем управления базами данных

В начале 1970-х годов был разработан новый вид программного обеспечения системы управления базами данных (Data Base Management System — DBMS), позволивший структурировать, систематизировать и организовать данные для их компьютерного хранения и обработки. Системой управления базами данных (СУБД) называют программную систему, предназначенную для создания на ЭВМ общей базы данных для множества приложений; поддержания ее в актуальном состоянии и обеспечения эффективного доступа пользователей к содержащимся в ней данным в рамках предоставленных им полномочий. СУБД предназначена, таким образом, для централизованного управления базой данных как общим ресурсом в интересах всей совокупности пользователей.

В настоящее время практически невозможно представить информационную поддержку современного учреждения без применения профессиональных СУБД. Однако существующий сегодня уровень возможностей программных продуктов данного направления был достигнут не сразу: эволюция СУБД прошла путь от систем, опиравшихся на иерархическую и сетевую модель данных, до систем так называемого третьего поколения, для которых характерны идеи объектно-ориентированного подхода.

Системы управления базами данных можно классифицировать, используя различные признаки:

По используемому языку общения:

замкнутые: имеют собственные самостоятельные языки общения пользователей с БД, они обеспечивают непосредственное общение с системой в режиме диалога, позволяют работать без программистов;

открытые: для общения с БД используется язык программирования, «расширенный» операторами языка манипулирования данными (ЯМД), в этом случае необходимо присутствие квалифицированного программиста.

По числу поддерживаемых СУБД уровней моделей данных:

одноуровневые системы;

двухуровневые системы;

трехуровневые системы.

Теоретически обоснован выбор трехуровневой архитектуры данных; однако на практике СУБД для персональных ЭВМ часто объединяют концептуальный и внутренний уровни представления.

По выполняемым функциям:

операционные: иные виды обработки по получению информации, не хранящейся в явном виде в БД;

информационные: позволяют организовать хранение данных, поиск и выдачу нужных данных из БД и поддерживать их целесообразность и актуальность.

По сфере применения:

универсальные: настраиваются на любую предметную область путем создания соответствующей БД и прикладных программ;

проблемно-ориентированные: ориентация на определенные процедуры обработки данных, присущих конкретной области применения.

По допустимым режимам работы: пакетный; телеобработка.

3.2. Основные функции систем управления базами данных

1. Управление данными во внешней памяти

Функция управления данными во внешней памяти включает в себя обеспечение необходимых структур внешней памяти как для хранения непосредственных данных, так и для служебных целей, например для ускорения доступа к данным в некоторых случаях (обычно используются индексы). В некоторых реализациях СУБД активно используются возможности существующих файловых систем, однако пользователи не должны знать, использует ли СУБД файловую структуру или нет. Существует множество способов организации внешней памяти баз данных. Как и все решения, принимаемые при создании баз данных, конкретные методы организации внешней памяти необходимо выбирать вместе со всеми остальными решениями.

2. Управление буферами оперативной памяти

СУБД обычно работают с базами данных значительных размеров; по крайней мере этот размер превышает доступный объем оперативной памяти. Понятно, что если при обращении к любому элементу данных будет производиться обмен с внешней памятью, то вся система будет работать со скоростью внешней памяти.

Единственным способом реального увеличения этой скорости является буферизация данных в оперативной памяти. И даже если операционная система производит общесистемную буферизацию, этого недостаточно для целей СУБД, которая располагает гораздо большей информацией о полезности буферизации той или иной части базы данных. В развитых СУБД поддерживается свой набор буферов оперативной памяти с собственной дисциплиной замены буферов. При управлении буферами необходимо разрабатывать и применять согласованные алгоритмы буферизации, журнализации и синхронизации. Заметим, что существует собственное

направление СУБД, которое ориентировано на постоянное присутствие всей БД в ОП. Это направление основывается на предположении, что в предвидимом будущем объем оперативной памяти может быть настолько велик, что позволит не беспокоиться о буферизации.

3. Управление транзакциями

Транзакция — это последовательность операций над БД, рассматриваемых СУБД как единое целое. Либо транзакция успешно выполняется и СУБД фиксирует изменения БД, произведенные ею во внешней памяти, либо ни одно из этих изменений никак не отражается в состоянии БД. Понятие транзакции необходимо для поддержания логической целостности БД (например, необходимость объединения элементарных операций над файлами). Поддержание механизма транзакций — необходимое условие даже однопользовательских СУБД. Но понятие транзакции гораздо важнее в многопользовательских СУБД. То свойство, что каждая транзакция начинается при целостном состоянии БД и оставляет это состояние целостным после своего завершения, делает очень удобным использование понятия транзакции как единицы активности пользователя по отношению к БД. При соответствующем механизме управления транзакциями пользователь может почувствовать себя единственным пользователем СУБД.

4. Журнализация и восстановление БД после сбоев

Одно из основных требований к СУБД — надежное хранение данных во внешней памяти. Под надежностью хранения понимается то, что СУБД должна быть в состоянии восстановить последнее согласованное состояние БД после аппаратного или программного сбоя. Поддержание надежного хранения данных в базе данных требует избыточности хранения данных, причем та их часть, которая используется для восстановления, должна храниться особо надежно. Наиболее распространенный метод поддержания такой избыточности — это ведение журнала изменений базы данных. Во всех случаях придерживаются «упреждающей» записи в журнал (так называемый протокол Write Ahead Log). Эта стратегия заключается в том, что запись об изменении любого объекта БД должна попасть во внешнюю память журнала раньше, чем она попадет во внешнюю память основной части БД. Известно, что если в СУБД корректно соблюдается протокол WAL, то с помощью журнала можно решить все проблемы восстановления БД после любого сбоя.

5. Поддержание языков БД

Для работы с БД используются специальные языки, в целом называемые языками баз данных. В ранних СУБД поддерживалось несколько специализированных по своим функциям языков. В современных СУБД обычно поддерживается единый интегрированный язык, содержащий все необходимые средства для работы с БД, обеспечивающий базовый пользовательский интерфейс с базами данных.

3.3. Типовая организация современной СУБД

Организация типичной СУБД и состав ее компонентов соответствуют набору функций. Логически в современной СУБД можно выделить внутреннюю часть — ядро СУБД {Data Base Engine), компилятор языка БД (обычно SQL), подсистему поддержки времени выполнения, набор утилит.

Ядро СУБД отвечает за управление данными во внешней памяти, управление буферами оперативной памяти, управление транзакциями и журнализацию. Соответственно можно выделить и такие компоненты ядра (по крайней мере логически, хотя во многих СУБД они существуют явно), как менеджер данных, менеджер буферов, менеджер транзакций, менеджер журнала.

Все функции взаимосвязаны, поэтому компоненты должны взаимодействовать по продуманным и спланированным протоколам. Ядро СУБД обладает собственным интерфейсом, не доступным пользователю напрямую и используемым в программах, производимых компилятором SQL, и утилитах БД. Ядро СУБД является основной резидентной частью СУБД. При использовании архитектуры «клиент—сервер» ядро является основным составляющим элементом серверной части системы.

Основная функция компилятора языка БД — компиляция операторов языка БД в некоторую выполняемую программу. Основная проблема реляционных СУБД — наличие непроцедурного языка, т.е. в операторе такого языка специфицируется некоторое действие над БД, но эта спецификация не процедура, она лишь описывает в некоторой форме условия совершения желаемого действия. Поэтому компилятор должен сначала решить, каким образом выполнить оператор языка, прежде чем произвести программу. Результатом компиляции является выполнимая программа, представляемая в некоторых системах в машинных кодах, но более часто — в выполняемом внутреннем машинно-независимом коде. В последнем случае реальное выполнение оператора производится с привлечением подсистемы поддержки времени выполнения, представляющей собой по сути интерпретатор этого внутреннего кода.

В отдельные утилиты обычно выделяют такие процедуры, которые слишком накладно выполнять с использованием языка БД, например загрузка БД, сбор статистики, глобальная проверка целостности. Утилиты программируются с использованием ядра СУБД, а иногда с проникновением внутрь ядра.

3.4. Информационная безопасность систем управления базами данных

Системы управления базами данных стали основным инструментом, обеспечивающим хранение больших массивов информации. Современные информационные приложения опираются, как уже говорилось, в первую очередь на многопользовательские СУБД. В этой связи пристальное

внимание в настоящее время уделяется проблемам обеспечения информационной безопасности, которая определяет степень безопасности организации, учреждения в целом.

Под информационной безопасностью понимают защищенность информации от случайных и преднамеренных воздействий естественного или искусственного характера, чреватых нанесением ущерба владельцам или пользователям информации.

В целях защиты информации в базах данных на практике важнейшими являются следующие аспекты информационной безопасности (европейские критерии):

доступность (возможность получить некоторую требуемую информационную услугу);

целостность (актуальность и непротиворечивость информации, ее защищенность от разрушения и несанкционированного изменения);

конфиденциальность (защита от несанкционированного прочтения).

Проблема обеспечения информационной безопасности — комплексная, поэтому ее решение должно рассматриваться на разных уровнях: законодательном, административном, процедурном и программно-техническом. К сожалению, в настоящее время особенно остро стоит проблема с соответствующим законодательством, обеспечивающим использование информационных систем.

3.5. Системы распределенной обработки данных

Потребность в данных коллективного пользования в последнее время все более возрастает. Это послужило причиной все усиливающегося внимания к различным системам распределенной обработки данных.

Существует несколько понятий в этой области, которые необходимо определить более точно. Вначале выделим эти понятия:

распределенная обработка данных;

базы данных с сетевым доступом;

архитектура «клиент—сервер»;

распределенные базы данных.

Распределенная обработка данных

Под распределенной обработкой данных понимают обработку приложений несколькими территориально распределенными машинами. При этом в приложениях, связанных с обработкой базы данных, собственно управление базой данных может выполняться централизованно.

Базы данных с сетевым доступом

Системы баз данных, построенные с помощью сетевых версий, иногда неправомерно называют распределенными базами данных, в то время как фактически имеют дело лишь с распределенным (сетевым) доступом к централизованной базе данных. Такие системы создаются на основе оборудования и программного обеспечения различных локальных вычислительных сетей, большинство СУБД работает в сетях.

Архитектура систем баз данных с сетевым доступом предполагает выделение одной из машин сети в качестве центральной. Эта машина обеспечивает функционирование той части сетевой версии СУБД, которая осуществляет управление данными в терминах базы данных и называется сервером файлов (File Server). Предполагается, что центральная машина обладает жестким диском достаточно большой емкости, на котором хранится совместно используемая централизованная база данных. Все другие машины сети выполняют функции рабочих станций (Workstation), с помощью которых поддерживается доступ пользователей системы к централизованной базе данных. В соответствии с пользовательскими запросами файлы базы данных передаются на рабочие станции, где в основном и производится их обработка. Рабочая станция должна иметь достаточно ресурсов для обеспечения приемлемого уровня реактивности при обработке пользовательских запросов.

В последнее время происходит существенная трансформация подходов к использованию баз данных в обстановке локальных сетей, направленная на повышение роли центральной машины сети, ранее используемой лишь для реализации сервера файлов.

Это означает, что сервер файлов центральной машины обеспечивает обработку параллельных запросов на файлы, поступающих с рабочих станций, используя необходимую дисциплину блокирования ресурсов; рабочие станции должны при этом копировать с центральной машины нужные им файлы базы данных и выполнять их обработку.

Архитектура «клиент—сервер»«

Каждый из составляющих эту архитектуру элементов играет свою роль: сервер владеет и распоряжается информационными ресурсами системы, клиент имеет возможность воспользоваться ими.

Сервер базы данных представляет собой мультипользовательскую версию СУБД, параллельно обрабатывающую запросы, поступившие со всех рабочих станций. В его задачу входит реализациялогики обработки транзакций с применением необходимой техники синхронизации с поддержкой протоколов блокирования ресурсов, обеспечением предотвращения и/или устранения тупиковых ситуаций.

В ответ на пользовательский запрос рабочая станция получит не «сырье» для последующей обработки, а готовые результаты.

Программное обеспечение рабочей станции при такой архитектуре играет роль только внешнего интерфейса (Front-end) централизованной системы управления данными. Это позволяет существенно уменьшить сетевой трафик, сократить время на ожидание блокированных ресурсов данных в мультипользовательском режиме, разгрузить рабочие станции и при достаточно мощной центральной машине использовать для них более дешевое оборудование.

Для современных СУБД архитектура «клиент—сервер» стала фактически стандартом.

Распределенные базы данных

СУБД и централизация обработки информации позволили устранить такие недостатки традиционных файловых систем, как несвязанность, несогласованность и избыточность данных. По мере роста баз данных и особенно при их использовании в территориально разделенных организациях появляются другие проблемы. Так, для централизованной СУБД, находящейся в узле телекоммуникационной сети, с помощью которой различные подразделения организации получают доступ к данным, с ростом объема информации и количества транзакций возникают следующие трудности:

большой поток обменов данными;

низкая надежность;

низкая общая производительность;

большие затраты на разработку.

Хотя в централизованной базе данных легче обеспечить безопасность, целостность и непротиворечивость информации при обновлениях, перечисленные проблемы создают определенные трудности. В качестве возможного решения этих проблем предполагается децентрализация данных. При децентрализации достигается:

более высокая степень одновременности обработки вследствие распределения нагрузки;

улучшенное использование данных на местах при выполнении удаленных (дистанционных) запросов;

меньшие затраты;

простота управления.

Затраты на создание сети, в узлах которой находятся малые ЭВМ, гораздо ниже, чем затраты на создание аналогичной системы с использованием большой ЭВМ.

Дадим следующее определение: распределенная база данных — это набор файлов (отношений), хранящихся в разных узлах информационной сети и логически связанных таким образом, чтобы составлять единую совокупность данных (связь может быть функциональной или через копии одного и того же файла).

Распределенная база данных предполагает хранение и выполнение функций управления данными в нескольких узлах и передачу данных между этими узлами в процессе выполнения запросов.

Разбиение данных в распределенной базе данных может достигаться путем хранения различных таблиц на разных компьютерах или даже хранения разных частей и фрагментов одной таблицы на разных компьютерах. Для пользователя (или прикладной программы) не должно иметь значение, каким образом распределены данные между компьютерами. Работать с распределенной базой данных, если она действительно распределенная, следует так же, как и с централизованной, т.е. размещение базы данных должно быть прозрачно.

3.6. Перспективы развития смаем управления базами данных

Будучи основным фундаментальным средством построения информационных систем, используемых в производстве, бизнесе и научной деятельности, базы данных и системы управления ими составляют обширную область исследований. Ниже дадим обзор наиболее важных направлений исследований.

Несмотря на то что реляционные СУБД давно и прочно заняли основные позиции на рынке программного обеспечения по обработке данных, в этой области остается много нерешенных проблем. Во-первых, это касается нового стандарта языка SQL-3, возможности которого должны быть расширены за счет включения в него возможности определения триггеров, работы с объектами, расширения типов данных. Во-вторых, движение в сторону концепции открытых систем предполагает пересмотр организации серверов баз данных, допустив в них внутреннюю параллельность.

В-третьих, решение проблемы использования старых баз данных в рамках новых программных продуктов.

Значительное число разработок замечено в области постреляционных баз данных. Отметим следующие пути решения в этой области: во-первых, базы данных сложных объектов (реляционнаямодель с отказом от первой нормальной формы), нашедшие применение в нетрадиционных приложениях, требующих операций со сложноструктурированными объектами; во-вторых, разработка активных баз данных, для которых СУБД выполняет не только указанные пользователем действия, но и дополнительные действия в соответствии с правилами, заложенными в саму базу данных; в-третьих, темпоральные базы данных, как надстройка над реляционной базой данных, позволяющие поддерживать исторические данные системы; в-четвертых, интегрированные системы, обеспечивающие решение задачи интеграции неоднородных баз данных в единую глобальную систему.

Отдельный раздел в СУБД следующего поколения занимают объектно-ориентированные базы данных. Возникновение данного направления определяется прежде всего потребностями практики:

необходимостью разработки сложных информационных систем, для которых технология предшествующих баз данных не была удовлетворительной. В таких СУБД должны быть решены проблемы поддержки иерархии и наследования типов, возможность управления сложными объектами. Однако для решения этих задач существуют значительные ограничения, а именно: отсутствие общепринятой объектно-ориентированной модели данных, декларативного языка запросов и т.п. Разработчики в области баз данных определяют объектно-реляционным и объектно-ориентированным базам данных значительное место на рынке в ближайшее десятилетие.

Распределенные базы данных представляют еще одно измерение в пространстве разработок систем управления базами данных: применение

протоколов синхронизации транзакций, сокращение расходов на пересылку данных между узлами вычислительной сети в ходе выполнения распределенного запроса посредством репликации данных далеко не все возможные проблемы в данной области.

2.2. Технологии личных информационных систем

Личные информационные системы предназначены для информационного обслуживания рабочего места управленческого работника и по существу выполняют функции секретаря. Они, в частности, позволяют осуществлять:

планирование личного времени на различных временных уровнях с возможностью своевременного напоминания о наступлении запланированных мероприятий;

ведение персональных или иных картотек с возможностью автоматической выборки необходимой информации;

соединение по телефонным линиям с ведением журнала телефонных переговоров и выполнением функций, характерных для многофункциональных телефонных аппаратов;

ведение персональных информационных блокнотов для хранения разнообразной личной информации.

Из имеющихся сегодня на рынке программных продуктов личных информационных систем следует отметить прежде всего Lotus Organizer, для которого характерен великолепный интерфейс, обеспечивающий те же принципы работы, что и привычные многим бумажные органайзеры.

Кроме того, достаточно полезна интегрированная в рамках офисного пакета MS Office система Microsoft Outlook.

2.3. Технологии управления документами

Постоянное увеличение количества информации, необходимой для принятия правильного управленческого решения, приводит к тому, что традиционные методы работы с документами становятся неэффективными. Так, по сведениям компании Delphi, 15% бумажных документов безвозвратно теряются и для их поиска сотрудники тратят до 30% своего рабочего времени. При переходе к электронным документам и автоматизации документооборота рост производительности сотрудников увеличивается на 25—50%, сокращается время обработки одного документа более чем на 75%, на 80% уменьшаются расходы на оплату площади для хранения документов (оценка Nortan Nolan Institute).

Тенденцию перехода от традиционных технологий организации документооборота к компьютерным отражают следующие цифры: соотношение количества бумажных и электронных документов в течение ближайших пяти лет составит 50 на 50%, а в последующие десять лет — 30

на 70%; количество электронных документов удваивается за год, а бумажных документов растет только на 7% (данные компании XPLOR).

В конце 80-х — начале 90-х годов появились и начали интенсивно развиваться ряд новых технологий, успешно используемых в современных системах автоматизации документооборота:

технологии обработки изображений документов (Imaging System);

системы оптического распознавания символов (Optical Character Recognition System, OCR);

системы управления документами, СУД (Document Management System, DMS);

полнотекстовые базы данных (Full-Text System);

системы автоматизации деловых процедур, АДП (Work-Flow System);

программное обеспечение для рабочих групп (Groupware).

5.1. Системы обработки изображений документов

Системы обработки изображений документов предназначены для ввода, обработки, хранения и поиска графических образов бумажных документов. Подобные системы целесообразно применять в организациях с большим объемом документооборота. Техническое обеспечение систем включает высокоскоростные сканеры, документные контроллеры (выполняют быструю и высокоэффективную компрессию/декомпрессию документов и обеспечивают скоростную работу со сканерами и принтерами), библиотеки-автоматы на базе оптических накопителей с автоматической подачей дисков. Компьютерные образы документов находятся на сервере изображений и просматриваются на рабочих станциях-клиентах.

Системы обработки изображений осуществляют сканирование документов для записи на сервер, их классификацию по различным критериям, передачу изображений на рабочую станцию для просмотра, модификацию или печать. Подобные системы предусматривают также определение маршрута передачи изображений по сети, их рассылку по факсу или электронной почте, поиск изображений по отдельным элементам.

Так как файлы изображений достигают больших размеров, существуют различные варианты организации их хранения. В целях экономии памяти на запоминающем устройстве большинство систем сжимают изображения и создают специальный индекс изображений, где содержатся соответствующие значения атрибутов документов, например наименование, автор, тема.

В высокопроизводительных системах реализованы технологии, позволяющие увеличить скорость работы. Например: предварительная выборка и перенос изображений с медленных оптических носителей на более быстрые магнитные; адаптируемое кэширование, позволяющее хранить часто используемые изображения в памяти сервера; вывод на лазерный диск; групповое сканирование, обеспечивающее считывание нескольких страниц за одну операцию.

5.2. Системы оптического распознавания символов

Многие системы обработки изображений включают программное обеспечение оптического распознавания символов (OCR).

Применение OCR позволяет решить проблему перевода бумажных документов в электронную форму в виде текстового файла. Системы OCR позволяют получать электронную копию документа с печатного листа либо копию документа, пришедшего по факсу. Существуют экспериментальные системы, позволяющие подобным образом обрабатывать также и рукописные материалы (Intelligent Character Recognition).

Кратко функционирование системы OCR можно представить следующим образом. С помощью сканирующего устройства считывается изображение документа. В результате распознавания текста изображение документа отображается в файл, отформатированный как текстовый. Таким образом, бумажный документ, минуя низкопроизводительный и трудоемкий ручной ввод, автоматически преобразуется в электронную форму.

Выделяют два класса систем OCR — обучаемые и интеллектуальные. Принцип действия систем первого класса основан на поточечном сравнении оцифрованного символа с образцом из справочника. При совпадении образца и символа последний считается распознанным и добавляется в результирующий файл. При таком способе распознавания размеры образца и шрифта документа должны совпадать, т.е. в системе необходимо иметь «маски» для каждого размера каждого типа шрифта, поэтому подобная система более эффективна в случае однотипного и качественного текста.

Во втором случае «маска» символа заменяется на его «образ», который может быть использован для любых размеров шрифтов.

Для повышения точности распознавания интеллектуальные системы могут выполнять ряд проверок результирующего текста.

Например: осуществлять частотный анализ текста и сравнивать частоту появления данного символа в тексте с его частотой в языке оригинала или обнаруживать неправильное сочетание символов, исходя из правил орфографии.

В реальных системах OCR сочетаются различные распознавательные механизмы, что дает возможность обрабатывать любые шрифты и любые тексты.

На сегодняшний день известно несколько достаточно качественных программных продуктов по распознаванию текста, в том числе две системы отечественных фирм, ориентированных прежде всего на распознавание русскоязычных текстов (FineReader). Средняя скорость работы системы OCR на оборудовании средней мощности составляет примерно одну машинописную страницу в секунду. Качество распознавания в среднем 1—2 ошибки на 1000 знаков в тексте среднего качества.

Можно рекомендовать следующие критерии выбора системы OCR:

совместимость с существующим или приобретаемым программным и аппаратным обеспечением;

скорость сканирования и распознавания преобладающего в данной организации типа текста, например факс — русский язык, ксерокопия различного качества, машинопись различного качества и др.;

качество распознавания текстов различных типов, например количество ошибок на 1000 знаков;

способность распознавать редкие шрифты;

способность обучения новым символам;

наличие элементов семантического анализа текста;

наличие модуля проверки орфографии;

удобство пользовательского интерфейса.

5.3. Системы управления документами

Системы управления документами (СУД) предназначены для автоматизации хранения, поиска и управления электронными документами разнообразных форматов, в том числе и изображениями документов. Можно сказать, что СУД фактически выполняют роль СУБД для неструктурированной информации.

Развитые системы управления документами осуществляют следующие функции:

индексирование документов;

полнотекстовый поиск по ключевым словам;

управление конфигурацией документа с установлением взаимосвязи между отдельными структурными компонентами;

ассемблирование документов, позволяющее объединить все части составного документа для отображения на экране;

организация доступа к документу независимо от места его хранения;

поиск и управление документами с помощью ключевых компонентов, таких, как оглавление или название раздела;

многоуровневая защита данных, которая разрешает доступ к документам только отдельным пользователям или устанавливает виды доступа, например «только для чтения»;

администрирование учета и архивирования;

организация выдачи/возврата документа;

контроль версий документа;

рассылка документов.

Развитие сети Internet определяет появление ряда новых функций современных СУД. Например, возможность помещения документов на Web-узле; поддержка обмена документами по интрасети между сотрудниками фирмы, а по Internet — с клиентами и партнерами фирмы.

5.4. Программное обеспечение для рабочих групп (groupware)

Программное обеспечение для рабочих групп {groupware) предназначено для организаций, сотрудникам которых по характеру их деятельности требуется постоянный обмен документами. Осуществляет задачи хранения, просмотра и совместного использования документов. Системы класса groupware позволяют автоматизировать такую деятельность, которая не вписывается в стандартные схемы реляционных баз данных. Например, взаимодействие большого числа людей, исполняющих различные работы в физически удаленных друг от друга местах. Такие приложения могут обрабатывать как структурированную, так и неструктурированную информацию.

Основными функциями ПО для рабочих групп являются:

электронная почта;

поддержка видеоконференций/совещаний;

управление изображениями документов;

совместное использование документов;

маршрутизация документов;

календарное планирование.

5.5. Системы автоматизации деловых процедур

Системы автоматизации деловых процедур (АДП) предназначены для создания сложных прикладных систем коллективной обработки документов в процессе осуществления конкретных бизнеспроцессов. Документальные потоки на предприятии привязываются к существующим бизнес-процессам и регламенту их взаимодействия. При жесткой маршрутизации документа заранее прописывается движение документа по всем рабочим местам. Определяются права пользователей на документ в каждой точке маршрута. При свободной маршрутизации исполнитель может определить дальнейший путь движения документа, обычно на одном уровне.

Примером систем АДП может служить продукт фирмы Staffware. Документы в системе обрабатываются по принятому в организации алгоритму и перемещаются в рамках корпоративной системы между отдельными подразделениями и исполнителями по заранее определенным маршрутам. Система основана на технологии «клиент—сервер», интегрируется с программными продуктами, работающими на платформах Windows, UNIX. В состав системы может входить графический построитель процедур {Graphical Workflow Defmer), описывающий документопоток в виде диаграмм с указанием логических шагов, маршрутизации, предельных сроков и форм отчетов. Является инструментом для разработки модели бизнес-процессов.

В последнее время наблюдается тенденция сближения и пересечения функциональных возможностей вышеописанных систем.

Так, в системах, классифицируемых как group-ware, могут использоваться технологии полнотекстовых баз данных, систем управления документами и автоматизации деловых процедур. Поэтому в современных системах автоматизации документооборота вышеуказанные технологии могут применяться в качестве как отдельных, так и интегрированных компонент.

3. Корпоративная информационная система (КИС)

Не всякая информационная система может быть названа корпоративной. Ниже приведен перечень требований которым должна удовлетворять информационная система, относящаяся к классу КИС (по материалам работы Е. В. Дворниковой, режим доступа http://www.osp.ru/os/1998/02/179433/#part_1)

1. Функциональная полнота системы:

выполнение международных стандартов управленческого учета - MRP II, ERP, CSRP;

автоматизация в рамках системы решения задач:

- планирования, бюджетирования, прогнозирования;
- оперативного (управленческого) учета;
- бухгалтерского учета;
- статистического учета;
- финансово-экономического анализа.

формирование отчетов и ведение учета одновременно по российским и международным стандартам (IAS и GAAP);

общими характеристиками функциональной полноты корпоративной информационной системы является количество однократно учитываемых параметров деятельности предприятия.

Для КИС количество этих параметров должно быть примерно следующим:

Количество учитываемых параметров 2000 — 10000

Количество таблиц баз данных 800 — 3000

2. Локализация информационной системы:

функциональная (учет особенностей российского законодательства и системы расчетов);

лингвистическая (интерфейс, система помощи и документация на русском языке).

3. Система должна обеспечивать надежную защиту информации.

Для этого необходимы:

парольная система разграничения доступа к данным и функциям;

многоуровневая система защиты данных, включающая средства авторизации вводимой и корректируемой информации, регистрация времени ввода и модификации данных, протокол удалений;

программно-аппаратные средства шифровки данных, сертифицированные ФАПСИ.

4. Реализация удаленного доступа и работы в распределенных сетях.

5. Наличие инструментальных средств адаптации и сопровождения системы:

изменение структуры и функций бизнес-процессов;

изменение информационного пространства (изменение структуры, добавление или удаление БД, модификация полей таблиц, связей, индексов и т.п);

изменение интерфейсов ввода, просмотра и корректировки информации;

изменение организационного и функционального наполнения рабочего места пользователя;

генератор произвольных отчетов;

генератор сложных хозяйственных операций;

генератор форм (в том числе стандартизованных).

6. Обеспечение обмена данными между ранее разработанными ИС и другими программными продуктами, функционирующими на предприятии.

7. Возможность консолидации информации:

на уровне предприятий — для объединения информации филиалов, дочерних компаний, предприятий, входящих в холдинг, и т.п.

на уровне отдельных задач;

на уровне временных периодов — для выполнения анализа изменения тех или иных показателей за период, превышающий отчетный.

8. Наличие специальных средств анализа состояния системы в процессе эксплуатации:

анализ архитектуры баз данных;

анализ алгоритмов;

анализ статистики количества обработанной информации (количество записей, документов, проводок; объем дисковой памяти);

журнал выполненных операций;

список работающих станций, внутрисистемная почта.

Типы корпоративных информационных систем

Выделяют несколько типов КИС (см.: Никитин А.В. и др. Управление предприятием (фирмой) с использованием информационных систем:.- М.: ИНФРА-М, 2007. - , 188 с. ISBN 5-16-002036-5, стр.25):

- по степени автоматизации - автоматизированные и неавтоматизированные;

- по сфере функционирования объекта управления - промышленность, транспорт, торговля и т.п.;

- по уровню в системе управления - отраслевые, территориальные, корпоративные и т.п.;

- по объекту - управление технологическими процессами, управление проектом, управление складом и т.п.;

- по охвату объектов управления - интегрированные и локальные;

- по уровню адаптации к предприятию - «коробочный» продукт, «конструктор», разработанная на заказ;

- по принципу построения ИС - алгоритмические и интеллектуальные системы;

- по своим функциям - операционно-учетные, управленческие, системы поддержки принятия решений.

Выводы

1. Программное обеспечение играет ключевую роль в реализации технологий управления информационными ресурсами, во многом определяя конкретные особенности выполнения функций информационной поддержки управленческой деятельности.

2. С точки зрения общей информационной поддержки важное значение имеет выбор системного программного обеспечения, и прежде всего операционной системы, определяющей эффективность реализации информационных технологий и качество взаимодействия пользователей с компьютерными комплексами.

3. Выбор и реализация конкретных технологий управления информационными ресурсами определяется конкретными потребностями организации и имеющимися финансовыми возможностями.

4. В любом случае имеется определенный фиксированный набор базовых технологий, реализация которых необходима практически для любых форм и масштабов управленческой деятельности: подготовка текстовых документов; обработка финансово-экономической информации и подготовка табличных документов; управление базами данных; личная информационная поддержка.

5. В зависимости от текущего состояния организации и перспектив ее развития возможно рассмотрение и практическая реализация таких достаточно крупномасштабных информационных проектов, как технологии управления проектами, CASEтехнологии, технологии систем комплексного управления документами.

Выбор и использование технологий информационной поддержки управленческой деятельности с соответствующими программными средствами — функция специалистов по информационным технологиям, к помощи которых необходимо обращаться в той или иной форме: на постоянной основе, имея их в штате организации или приглашая их для временной работы в рамках разработки и реализации конкретных проектов или консультаций.

Глоссарий

PASCAL	универсальный язык, являющийся развитием системы ALGOL, ориентированный на широкий класс задач обработки данных и организации вычислений (научные, инженерные, экономические, управленческие и пр.).
WINDOWS	семейство оболочек MS-DOS и операционных систем для IBM PC-совместимых ПЭВМ, реализующих дружественный пользовательский интерфейс.
Адресация	способ размещения и выборки данных в памяти.
Алгоритм	понятное и точное предписание (указание) исполнителю совершить определенную последовательность действий, направленных на достижение указанной цели или решение поставленной задачи (приводящую от исходных данных к искомому результату).
Арифметические операции	четыре действия арифметики (+, -, *, /) и операция получения остатка от деления (%) образуют группу арифметических операций. Их выполнение не имеет каких-либо особенностей, кроме как преобразование типов переменных при их несовпадении. Если в одном выражении встречаются переменные разных типов, как правило, осуществляется преобразование (приведение) типов.
Байт	машинное слово минимальной размерности, адресуемое в процессе обработки данных. Размерность байта - 8 бит - принята не только для представления данных в большинстве компьютеров, но и в качестве стандарта для хранения данных на внешних носителях, для передачи данных по каналам связи, для представления текстовой информации.
Библиотека (library)	набор функций, в том числе из стандартных библиотек, предопределенных

	переменных и констант, которые могут быть использованы в программе и хранятся в откомпилированном виде.
Блок	составной оператор, включающий описания переменных в начале. Это собственные переменные блока, действие которых не распространяется за его пределы, а время существования совпадает с временем его выполнения.
Виртуальная память	отличается от обычной ОП тем, что какие-то ее редко используемые фрагменты могут находиться на диске и подгружаться в реальную ОП по мере необходимости. Такая организация памяти позволяет снять ограничение, накладываемое объемом физической памяти, установленной на ЭВМ. Для реализации ВП используют, например, динамическую переадресацию.
Внешняя Ссылка	обращение к переменной или вызов функции во внутреннем представлении модуля, которые определены в другом модуле и отсутствуют в текущем.
Время выполнения (run time)	период, во время которого происходит выполнение программы.
Время компиляции (compiler time)	период, во время которого происходит компиляция программы. Во время компиляции обнаруживаются синтаксические ошибки.
Вызов функции (call interface)	выполнение ее тела с заданными значениями формальных параметров.
Выражение	множество взаимосвязанных операций над переменными и константами и скобок, в котором результат одной операции является операндом другой.
Генерация кода	преобразование элементарных действий, полученных в результате лексического, синтаксического и семантического анализа программы, в некоторое внутреннее представление. Это могут быть коды команд, адреса и содержимое памяти данных, либо текст программы на языке Ассемблера, либо стандартизованный промежуточный код (например, Р-код). В процессе генерации кода производится и его оптимизация.
Главная (внутренняя,	представляет собой упорядоченную

оперативная) память компьютера	последовательность байтов или машинных слов (ячеек памяти), проще говоря - массив. Номер байта или слова памяти, через который оно доступно как из команд компьютера, так и во всех других случаях, называется адресом. Если в команде непосредственно содержится адрес памяти, то такой доступ этому слову памяти называется прямой адресацией.
Глобальные переменные	вне тела функции можно определить глобальные переменные. Этими переменными могут пользоваться все функции, следующие по тексту. Глобальные переменные представляют собой общие данные программы.
Графика (Graphical data)	тип данных, предназначенный для ввода и отображения графической (обычно растровой) информации.
Документ	агрегат данных в документальных системах (АИПС), имеющий иерархическую структуру и, кроме форматных полей (элементы или агрегаты данных фиксированной длины), обычно содержащий текстовые поля, или символьные последовательности неопределенной длины, логически подразделяющиеся на параграфы (par, segm), предложения (sent), слова (word).
Дополнительный код	беззнаковая форма представления чисел со знаком. В двоичной системе счисления дополнение каждой цифры выглядит как инвертирование двоичного разряда, т. е. замена 0 на 1 и наоборот. Если же знак числа представляется старшим разрядом машинного слова, то получается простой способ представления отрицательного числа: взять абсолютное значение числа в двоичной системе; инвертировать все разряды, включая знаковый; добавить 1.
Драйвер (Driver)	резидентный программный модуль, осуществляющий управление внешним устройством и связь с операционной системой и прикладными программами.
Задача	одна или несколько программ, связанных общим назначением, ресурсами.
Запись (Record)	Агрегат данных, составляющий элемент

	базы данных (файла), содержащий разнотипную информацию, описывающую некоторый объект (сущность, экземпляр, аспект).
Запись логическая	идентифицируемая (именованная) совокупность элементов или агрегатов данных, воспринимаемая прикладной программой как единое целое при обмене информацией с внешней памятью. Запись - это упорядоченная в соответствии с характером взаимосвязей совокупность полей (элементов) данных, размещаемых в памяти в соответствии с их типом.
Запись физическая	совокупность данных, которая может быть считана или записана как единое целое одной командой ввода-вывода.
Идентификатор	имя переменной (идентификатор), состоит из больших и маленьких латинских букв, цифр и знака «__» («подчеркивание») и начинается с буквы.
Имя файла	последовательность от одного до максимума (FilenameMax) символов, использующаяся для именования обычных файлов, каталогов или специальных файлов. В имени файла допустимы любые символы кода ASCII, за исключением управляющего кода О (ПУС) и символа «/». Не рекомендуется использовать в именах файлов символы, имеющие специальное значение для языков управления заданиями (типа «*», «?»).
Интерфейс (Interface)	совокупность технических и программных средств визуализации информации и ввода данных, обеспечивающая интерактивное взаимодействия пользователя с системой.
Информационная система (Information System)	совокупность информационных технологий и организационных мероприятий, обеспечивающая сбор, обработку, хранение, поиск и представление информации.
Информационная технология (Information Technology)	целенаправленное применение информации, технических, программных средств и интеллектуальных усилий пользователя для решения задачи.
Информация,	это один двоичный разряд. 8 бит

единицы измерения. Элементарной единицей информации является бит	образует 1 байт. Информация в памяти ЭВМ измеряется в следующих единицах: килобайт (К), 1 К =1024 байта (2exp10); мегабайт (М), 1 М = 1024 К; гигабайт (Г) 1 Г= 1024 М.
Исполняемый модуль	модуль, содержащий готовую к выполнению программу, может быть двух видов: точный образ памяти программы с привязкой к абсолютным адресам (в MS-DOS - формат файла *.com), перемещаемый исполняемый формат.
Исходный код программы	код, написанный на языке программирования. Может включать модули на ЯВУ и модули с подпрограммами на языке ассемблера.
Каталог	специальный тип файла (иногда именуется директорией или папкой), содержащий информацию о файлах, которые могут адресоваться из данного каталога без указания полного имени (т. е. по имени файла).
Ключ	значение (элемент данных), используемый для идентификации или определения адреса записи.
Код ASCII (ASCII code) -- American Standart Code for Information Interchange	7- или 8-битовый код обмена данными.
Кодирование (Coding)	установление согласованного (узаконенного) соответствия между набором символов и сигналами или битовыми комбинациями, представляющими каждый символ для целей передачи, хранения или обработки данных.
Компоновщик, редактор связей, линкер (link, Linkage editor)	программа, осуществляющая процесс сборки программы из объектных модулей, в котором производится их объединение в исполняемую программу и связывание вызовов внешних функций и их внутреннего представления (кодов), расположенных в различных объектных модулях. Эта программа собирает откомпилированный текст программы и функции из стандартных библиотек языка в одну выполняемую программу.

Конечный пользователь (End User)	пользователь, на обслуживание которого ориентирована система (информационно-поисковая, операционная и пр.).
Косвенная адресация	случай, когда машинное слово содержит адрес другого машинного слова.
Лексика языка программирования	правила «правописания слов» программы, таких, как идентификаторы, константы, служебные слова, комментарии. Лексический анализ разбивает текст программы на указанные элементы. Особенность любой лексики - ее элементы представляют собой регулярные линейные последовательности символов. Например, идентификатор - это произвольная последовательность букв, цифр и символа «_», начинающаяся с буквы или «_». Лексика ЯП анализируется и интерпретируется на фазе лексического анализа при трансляции.
Лингвистическое обеспечение (Software languages)	информационные языки, форматы данных, словари, классификаторы и тезаурусы, обеспечивающие функционирование информационных систем.
Логические операции	над значениями условных выражений можно выполнить логические операции и (&, and), или (|, OR) и не (!, NOT), которые объединяют по правилам логики несколько условий в одно.
Логическое данное (Logical)	тип данных, предназначенный для составления логических выражений и управления вычислительным процессом, типу соответствуют определенные операции и функции (логические).
Массив	упорядоченная последовательность переменных одного и того же типа, имеющая общее имя. Номер элемента в последовательности называется индексом. Количество элементов в массиве не может быть изменено в процессе выполнения программы. Элементы массива размещаются в памяти последовательно и нумеруются от 1 до л, где п - их количество в массиве.
Машинное слово	упорядоченное множество двоичных разрядов, используемое для хранения команд программы и обрабатываемых данных.

	Каждый разряд, называемый битом - это двоичное число, принимающее значения только 0 или 1. Разряды в слове обычно нумеруются, справа налево, начиная с 0. Количество разрядов в слове называется размерностью машинного слова или его разрядностью.
Модульное программирование	разбиение программы на подпрограммы по специфике обрабатываемых данных. Для этой цели в ЯВУ используются функции и процедуры.
Наследование	Новый, или производный класс может быть определен на основе уже имеющегося или базового. При этом новый класс сохраняет все свойства старого: данные объекта базового класса включаются в данные объекта производного, а методы базового класса могут быть вызваны для объекта производного класса, причем они будут выполняться над данными включенного в него объекта базового класса. Иначе говоря, новый класс наследует как данные старого класса, так и методы их обработки.
Объект (object)	Формально это совокупность данных и методов работы с ними, некоторые из которых могут использоваться другим приложением. Объектно-ориентированные технологии позволяют создателю объекта определить интерфейсы к возможностям объекта, скрыв при этом особенности его реализации. Это делает возможным использование объекта многими непосредственно не относящимися к нему приложениями.
Объектно-ориентированное программирование	Технология ООП прежде всего накладывает ограничения на способы представления данных в программе. Любая программа отражает в них состояние физических предметов либо абстрактных понятий (объекты программирования), для работы с которыми она предназначена. В традиционной технологии варианты представления данных могут быть разными. В худшем случае программист может «равномерно размазать» данные о некотором объекте программирования по всей

	программе. В противоположность этому все данные об объекте программирования и его связях с другими объектами можно объединить в одну структурированную переменную. В первом приближении ее можно назвать объектом. Кроме того, с объектом связывается набор действий, иначе называемых методами. С точки зрения языка программирования это функции, получающие в качестве обязательного параметра указатель на объект. Технология ООП запрещает работать с объектом иначе, чем через методы, т. е. внутренняя структура объекта скрыта от внешнего пользователя. Описание множества однотипных объектов называется классом. Традиционная технология программирования «от функции к функции» определяет первичность алгоритма (процедур, функций) по отношению к структурам данных. Технология ООП определяет первичность данных (объектов) по отношению к алгоритмам их обработки (методам).
Объектный модуль	код программы после трансляции (компиляции), преобразованный в машинные коды. Помимо них содержит внешние ссылки и информацию для редактора связей и может также содержать отладочную информацию (debug info).
Операнд	переменная, константа, выражение, участвующие в операции. Унарная операция - операция с одним операндом. Бинарная - операция с двумя операндами.
Оператор	синтаксическая единица программы, которая отражает логику ее работы (последовательная, ветвящаяся, повторяющаяся). Для операторов характерен принцип вложенности: составными частями оператора могут быть любые другие операторы, и сам он, в свою очередь, может входить составной частью в оператор более высокого уровня.
Оператор цикла	обеспечивает повторяющееся выполнение следующего за ним оператора (или блока) - тела цикла. Шаг цикла (итерация цикла) - конкретный факт выполнения тела

	цикла.
Операции сравнения	(«=» - равно, «!=» - не равно, а также «<>, <, >=, <=») дают в качестве результата значения «истина» или «ложь». Выражения с такими значениями называются условными, поскольку обозначают выполнение или, наоборот, невыполнение некоторого условия в программе. Они используются в условном операторе (if) и в операторах цикла (do - while, for).
Операционная система	обеспечивает следующие функции: управление процессором путем передачи управления программам, обработка прерываний, синхронизация доступа к ресурсам, управление памятью, управление устройствами ввода-вывода, управление инициализацией программ, межпрограммные связи, управление данными на долговременных носителях путем поддержки файловой системы.
Определение переменной	«создает» переменную, выделяя под нее память, задавая имя, тип и, возможно, начальное значение.
Открытость	свойство информационных технологий и систем, предполагающее способность объединять разные информационные системы, аппаратуру и программные продукты различных производителей, что делает возможным обмен между ними данными, распределенный доступ к информационным ресурсам.
Переменная	именованная область памяти программы, в которой размещены данные с определенной формой представления (типом). Для того чтобы воспользоваться переменной, необходимо произвести некоторые предварительные действия - выполнить определение переменной. Только при наличии такого определения транслятор будет знать ее имя, тип данных и, возможно, начальные значения.
Поле (Field)	однородный элемент данных определенного типа, описывающий аспект объекта или соответствующий фрагменту документа.

Полиморфизм	основывается на возможности включения в данные объекта также и информации о методах их обработки (в виде указателей на функции). Принципиально важно, что такой объект становится «самодостаточным». Будучи доступным в некоторой точке программы, даже при отсутствии полной информации о его типе, он всегда может корректно вызвать свойственные ему методы. Полиморфной называется функция, независимо определенная в каждом из группы производных классов и имеющая в них общее имя.
Полное имя файла	последовательность имен каталогов, разделенных символами «/», предшествующая имени файла. Полное имя файла содержит информацию о положении каталога с файлом в дереве файлов.
Пользователь (User)	физическое или юридическое лицо, непосредственно применяющее информационный ресурс, систему, технологию для решения задач.
Препроцессор	предварительная фаза трансляции на уровне преобразования исходного текста программы с использованием директив препроцессора.
Приведение типов	в выражениях в качестве операндов могут присутствовать переменные и константы разных типов. Результат каждой операции также имеет свой определенный тип, который зависит от типов операндов. Если в бинарных операциях типы данных обоих операндов совпадают, то результат будет иметь тот же самый тип. Если нет, то транслятор должен включить в код программы неявные операции, которые преобразуют тип операндов, т. е. выполнить приведение типов.
Прикладная программа	подготовленная на языке программирования (Pascal, BASIC) обычно называется исходной программой (source program). Двоичную версию программы, выходящую из компилятора и размещаемую в оперативной памяти для последующего выполнения называют исполнительный модуль (object module) в машинном

	представлении. Она представляет собой последовательность машинных инструкций (команд), которые конкретно указывают компьютеру, что надо делать.
Присваивание	операция, которая запоминает значение выражения, стоящее справа от символа «=» в переменной или элементе массива, стоящем слева. При присваивании происходит преобразование форм представления (типов), если они не совпадают.
Программа	одна или несколько последовательностей, связанных команд (инструкций), которые, будучи выполнены компьютером, реализуют определенную функцию или операцию.
Пустой оператор	не производящий никаких действий, обозначается символом «;», который встречается в программе «сам по себе». Пустой оператор используется там, где по синтаксису требуется наличие оператора, но никаких действий производить не нужно.
Разметка (Markup)	размещение в тексте и использование при обработке и отображении документов (данных) управляющих меток (тагов, тэгов) - бинарных кодов или слов.
Редактор текстовый (Text editor)	программа, предназначенная для приема, обработки и отображения текстовых файлов.
Резидентная программа	программа, которая после загрузки в ОЗУ и передаче ей управления инициализируется таким образом, что постоянно находится в ОЗУ и выполняется параллельно другим программам.
Результат Функции	выходная переменная, которую формирует функция и значение которой используется затем в том месте программы, где она вызывается. Результат, как любая другая переменная, должен быть определенного типа.
Семантика языка программирования	это смысл, который закладывается в каждую конструкцию языка. Семантический анализ - это проверка смысловой правильности конструкции.
Символьное данное	тип данных, предназначенный для ввода

(Character)	и отображения алфавитной, цифровой и спецсимвольной информации; типу соответствуют определенные операции над переменными и функции (строчные).
Синтаксис языка программирования	правила составления предложений языка из отдельных слов. Такими предложениями являются операции, операторы, определения функций и переменных. Особенностью синтаксиса является принцип вложенности (рекур-сивность) правил построения предложений.
Система программирования (Programming System)	программная среда разработки приложений с использованием языка программирования, библиотеки функций, компилятора или интерпретатора ЯП.
Система счисления	совокупность правил наименования и изображения чисел с помощью набора символов, называемых цифрами. Система счисления делится на позиционные и непозиционные. Пример непозиционной системы счисления - римская, к позиционным системам счисления относится двоичная, десятичная, восьмеричная, шестнадцатеричная.
Система управления базами данных (СУБД)	совокупность языковых и программных средств, предназначенных для создания, ведения и совместного использования БД многими пользователями. Систему управления базой данных можно также определить как специальный пакет программ, посредством которого реализуется централизованное управление базой данных и обеспечивается доступ к данным.
Системное программное обеспечение	обеспечивает интерфейс между программистом или пользователем и аппаратной частью ЭВМ (операционная система, программы-оболочки) и может выполнять вспомогательные функции (программы-утилиты).
Составной оператор	последовательность операторов, заключенная в операторные скобки ({}, begin end), образует составной оператор (или блок) и входит в охватывающую его конструкцию как одно целое, т. е. с точки зрения транслятора становится одним оператором.

Список	элемент, содержащий все возможные в каждом конкретном случае значения, которые пользователь может установить. Добавить или изменить эти значения непосредственно в списке нельзя.
Стандартное машинное слово	машинное слово, размерность которого совпадает с разрядностью процессора. Большинство команд процессора использует для обработки данных стандартное машинное слово.
Статическое выделение памяти	выделение памяти под данные вн>три сегмента данных программы. Такие данные существуют на протяжении всей жизни программы до ее завершения.
Стек	среда для размещения данных для возврата из подпрограмм, а также их аргументы и автоматические данные. Все это может потребовать достаточно большого размера стека. Как правило, программист может определять размер стека в программе.
Техническое обеспечение (Hardware)	комплекс технических средств, обеспечивающих информационные технологии, связанные с приемом, передачей, хранением и отображением информации.
Тип данных	характер данных, определяющий способ представления значения в памяти и, соответственно, множество стандартных операций (примитивов) преобразования этого значения.
Трансляция	получение объектного кода из исходного.
Управляющие структуры (структурные операторы)	Для того чтобы установить порядок выполнения отдельных операторов, в программу вводят структурные операторы. К структурным операторам относятся составной оператор, условный оператор и цикл.
Условный оператор if	используется в программе, когда нужно выполнить одну или другую последовательность действий в зависимости от выполнения некоторого условия.
Файл	именуемая единица информации, поддерживаемая операционной системой. Доступ к данным реализуется либо в рамках ОС, либо пользовательскими программами,

	либо в рамках СУБД, либо комбинированно.
Файл ASCII (ASCII-Hie)	файл, содержащий символьную информацию, представленную только ASCII-кодами «левой части» (первые 128 символов кодовой таблицы, или код Latin-1) и символьную разметку.
Файл бинарный (Binary File)	файл, содержащий произвольную двоичную информацию (текст с бинарной разметкой, программа, графика, архивный файл).
Файловая система (File management system)	динамически поддерживаемая информационная структура на устройствах прямого доступа (диски), обеспечивающая функцию управления данными ОС путем связи «имя-адрес».
Фиксированная точка (fixed)	простейший тип числовых данных, когда число размещено в машинном слове, и диапазон значений зависит только от разрядности слова.
Формальные параметры	После имени функции в круглых скобках идет список формальных параметров, разделенных запятыми. Формальные параметры - это тоже переменные особого рода - их синтаксис в перечислении напоминает определение обычных переменных.
Функция	выполняет некоторое законченное действие и имеет «стандартный интерфейс» с набором параметров, через который она вызывается из других функций.
Цикл ПОКА	отличается от цикла ДО тем, что проверка условия проводится до выполнения тела цикла, и если при первой проверке условие выхода из цикла выполняется, то тело цикла не выполняется ни разу.
Цикл ДО	применяется при необходимости выполнить какие-либо вычисления в несколько раз до выполнения некоторого условия. Особенностью этого цикла является то, что он выполняется хотя бы 1 раз.
Числа с плавающей тонкой (float)	числовое данное, размещенное в машинном слове в форме мантиссы и порядка, что позволяет представлять широкий диапазон значений; предполагает наличие встроенной

	или эмулируемой арифметики (операции с плавающей точкой).
Числовое данное (Numerical)	тип данных, предназначенный для вычислений и составления арифметических выражений, которому соответствуют определенные операции и функции (арифметические).
Элемент данных (элементарное данное)	неделимое именованное данное, характеризующееся типом (напр., символьный, числовой, логический, и пр.), длиной (в байтах) и обычно рассчитанное на размещение в одном машинном слове соответствующей разрядности. Это минимальная адресуемая (идентифицируемая) часть памяти - единица данных, на которую можно ссылаться при обращении к данным. Ранние языки программирования (Алгол, Фортран) были рассчитаны на обработку элементарных данных или их простейших агрегатов - массивов (матрицы, векторы). Позднее появляется возможность представления и обработки агрегатов разнотипных данных (записей). В реляционных БД элементарное данное есть элемент таблицы. Иногда используется термин поле записи в качестве синонима.
Язык	См.: ***Язык командный (Command Language)***; ***Язык программирования (Programming Language)***
Язык командный (Command Language)	система формализованных запросов оператора и ответов операционной системы, обеспечивающая функцию связи с оператором (для управления задачами и данными).
Язык программирования (Programming Language)	совокупность средств, предназначенная для описания алгоритмов, реализуемых в программах ЭВМ.