## § 2. ПОНЯТИЕ О СИСТЕМЕ ПРОГРАММИРОВАНИЯ

### 2.1. ОСНОВНЫЕ ФУНКЦИИ И КОМПОНЕНТЫ

Системы программирования - это комплекс инструментальных программных средств, предназначенный для работы с программами на одном из языков программирования. Системы программирования предоставляют сервисные возможности программистам для разработки их собственных компьютерных программ.

В настоящее время разработка любого системного и прикладного программного обеспечения осуществляется с помощью систем программирования, в состав которых входят

• трансляторы с языков высокого уровня;

• средства редактирования, компоновки и загрузки программ;

• макроассемблеры (машинно-ориентированные языки);

• отладчики машинных программ.

Системы программирования, как правило, включают всебя

• текстовый редактор (**Edit**), осуществляющий функции записи и редактирования исходного текста программы;

•загрузчик программ (**Load**), позволяющий выбрать из директория нужный текстовый файл программы;

• запускатель программ (**Run**), осуществляющий процесс выполнения программы;

• компилятор **(Compile),** предназначенный для компиляции или интерпретации исходного текста программы в машинный код с диагностикой синтаксических и семантических (логических) ошибок;

• отладчик **(Debug),** выполняющий сервисные функции по отладке и тестированию программы;

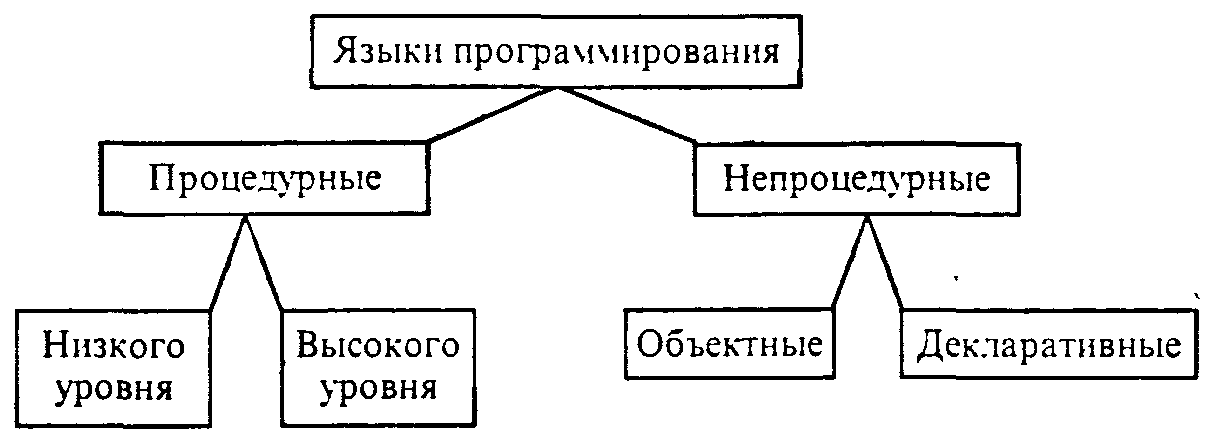
• диспетчер файлов (**File**), предоставляющий возможность выполнять операции с файлами:сохранение, поиск, уничтожение и т.п.

Ядро системы программирования составляет язык. Существующие языки программирования можно разделить на две группы: процедурные и непроцедурные, рис. 2.9.

Процедурные (или алгоритмические) программы представляют из себя систему предписаний для решения конкретной задачи. Роль компьютера сводится к механическому выполнению этих предписаний.

Процедурные языки разделяют на языки низкого и высокого уровня.

Языки низкого уровня (машинно-ориентированные) позволяют создавать программы из машинных кодов, обычно в шестнадцатиричной форме. С ними трудно работать, но созданные с их помощью высококвалифицированным программистом программы занимают меньше места в памяти и работают быстрее. С помощью этих языков удобнее разрабатывать системные программы, драйверы (программы для управления устройствами компьютера), некоторые другие виды программ.



*Рис. 2.9.* Общая классификация языков программирования

Программы на языках высокого уровня близки к естественному (английскому)

языку и представляют набор заданных команд.

Перечислим наиболее известные системы программирования.

1. Фортран (FORmula TRANslating system - система трансляции формул); старейший и по сей день активно используемый в решении задач математической ориентацииязык.

2. Бейсик (Beginner's All-purpose Symbolic Instruction Code - универсальный символический код инструкций для начинающих); несмотря на многие недостатки и изобилиеплохо совместимых версий - самый популярный по числу пользователей.

3. Алгол (ALGOrithmic Language - алгоритмический язык); сыграл большую роль в теории, но для практического программирования сейчас почти не используется.

4. ПЛ/1 (PL/I Programming Language - язык программирования первый). Многоцелевой язык; сейчас почти не используется.

5. Си (С - «си»); широко используется при создании системного программного обеспечения.

6. Паскаль (Pascal - назван в честь ученого Блеза Паскаля); чрезвычайно популярен как при изучении программирования, так и среди профессионалов. На его базе созданы несколько более мощных языков (Модула, Ада, Дельфи).

7. Кобол (COmmon Business Oriented Language - язык, ориентированный на общийбизнес); в значительной мере вышел из употребления.

8. Дельфи (Delphi) - язык объектно-ориентированного «визуального» программирования; в данный момент чрезвычайно популярен.

9. Джава (Java) - платформенно-независимый язык объектно-ориентированного программирования, чрезвычайно эффективен для создания интерактивных веб-страниц.

Среди непроцедурных языков наиболее известны

1. Лисп (Lisp);

2. Пролог (PROgramming in LOGic);

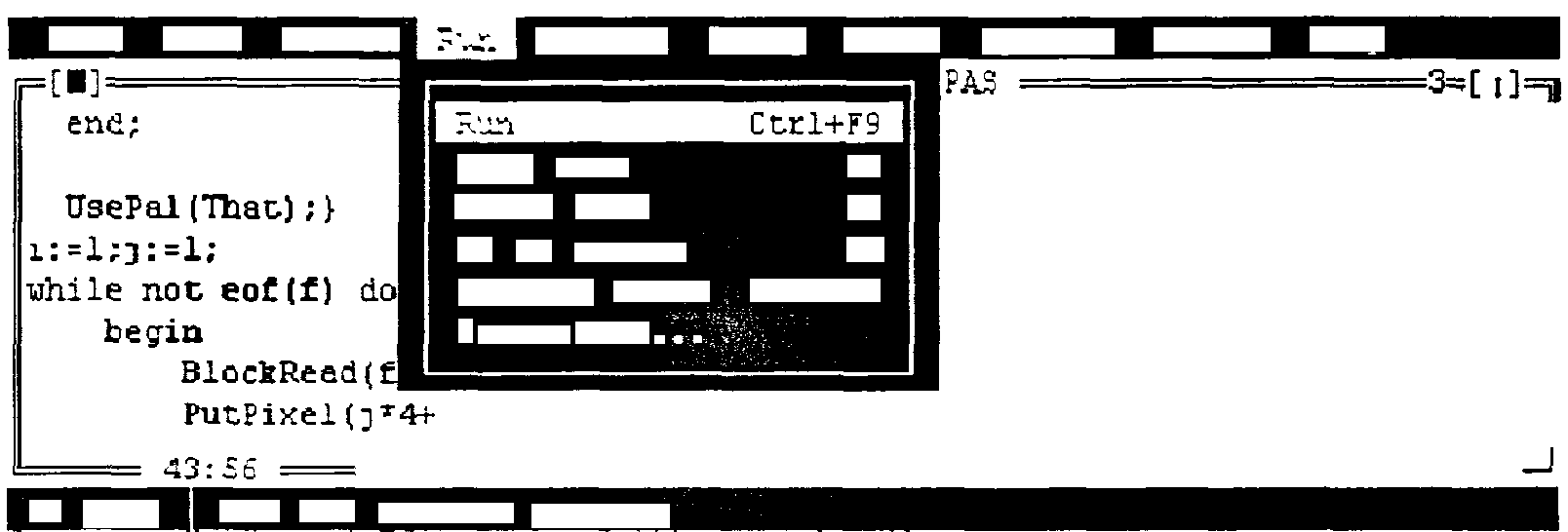
3. Оккам (назван в честь философа У. Оккама).

Широкое распространение среди разработчиков программ, а также при обучении программированию, получили системы программирования «Турбо» (Turbo) фирмы Borland, ядром которых являются трансляторы с языков программирования Бейсик, Паскаль, Си, Пролог и др. Интерфейс Турбо-оболочки для любых систем программирования внешне совершенно одинаков и предоставляет пользователю стандартный набор функций и команд, описанных выше и отображаемых в главном меню системы.

Рассмотрим технологию разработки программ с использованием популярной системы программирования Турбо-Паскаль 7 (оставив знакомство с самим языком до следующей главы).

В подобных интегрированных системах программирования сделана попытка предоставить разработчику программ максимум сервисных возможностей. Помимо основных функций система Турбо-Паскаль 7 позволяет настроить компилятор на работу в трех режимах: обычном режиме MS DOS (Real), защищенном режиме (Protected) и в режиме операционной среды Windows (Windows).

После загрузки системы (файл TURBO. EXE), на экране монитора появляется интерфейсное окно, рис. 2.10.



*Рис. 2.10.* Вид экрана интегрированной среды Турбо-Паскаля версии 7 (монтаж)

Главное меню системы (верхняя строка экрана) содержит команды, которые позволяют осуществлять следующие виды работ:

File - работа с файлами (сохранение, загрузка, связь с операционной   
 системой);

Edit - работа с текстовым редактором (после загрузки системы по умолчанию   
 текстовый редактор находится в активном состоянии);

Search - поиск и замена фрагментов текста;

Run -запуск программы на выполнение;

Compile — компиляция программы и установка параметров компиляции;

Debug - установка параметров отладки программы;

Tools - инструментальные программные средства (ненавязчивый сервис);

Options -установка опций интегрированной среды;

Window - работа с окнами;

Help -система помощи и подсказок.

Для начала работы с системой программирования необходимо иметь проект текста программы, который можно набирать на рабочем поле окна системы. Встроенный текстовый редактор прост и максимально приспособлен для набора текстов программ на языке Паскаль. В нем предусмотрена специальная подсветка управляющих структур, команд. Удобна система контекстной помощи (Shift+Fl), которая вызовет подсказку по набираемому текущему тексту программы в любой момент и в любом месте. Впрочем, текст программы можно приготовить в любом текстовом редакторе, хранящем тексты в ASCII-кодах (например, в Лексиконе); необходимо лишь снабдить имя файла расширением .pas.

Если текст (тексты) программы был ранее сохранен на жестком диске или дискете, то он может быть загружен в поле редактирования с помощью пункта меню **File**.

После окончания формирования текста необходимо откомпилировать программу (пункт меню Compile). Если в программе есть ошибки, то компилятор их укажет. После исправления ошибок можно снова повторить компиляцию.

После удачной компиляции запуск программы осуществляется командой меню **Run.**

Но на этом этапе чаще всего работа не заканчивается. Сложные алгоритмы требуют тестирования и отладки. Многие программы составляются из отдельных модулей, требуют связи с другими программами и системами и т.д. Для решения всех этих проблем предназначены другие команды системы (**Debug**, **Options** и пр.).

Разумеется, программисту, работающему на Паскале, нет нужды самому программировать такие непростые, но часто встречающиеся операции, как вычисление значений математических функций, построение изображений простых геометрических объектов (отрезков прямых, окружностей и т.д.), очистка экрана и множество других. Высокоэффективные, тщательно отлаженные программы таких действий сведены в стандартные модули и надо лишь уметь к ним обратиться. В состав пакета библиотек стандартных модулей входят: **Crt -** работы с экраном, **Graph** -работы с графикой и другие, такие как **Overlay**, **String**, **System**, **Turbo3**, **WinAPI, WinCrt, WinDos, WinPrn, WinTypes, WinProcs.**

### 

### 2.2. ТРАНСЛЯЦИЯ ПРОГРАММ И СОПУТСТВУЮЩИЕ ПРОЦЕССЫ

С появления первых компьютеров программисты серьезно задумывались над проблемой кодирования компьютерных программ. Уже с конца 40-х годов стали появляться первые примитивные языки программирования высокого уровня. В них программист записывал решаемую задачу в виде математических формул, а затем, используя специальную таблицу, переводил символ за символом, преобразовывал эти формулы в двухлитерные коды. В дальнейшем специальная программа (впоследствии названная интерпретатором) превращала эти коды в двоичный машинный код. Первый компилятор был разработан Г. Хоппер в начале 50-х годов; он осуществлял функцию объединения команд и в ходе трансляции производил организацию подпрограмм, выделение памяти компьютера, преобразование команд высокого уровня (в то время псевдокодов) в машинные команды. В дальнейшем компиляторы и интерпретаторы для языков Ассемблера стали развиваться и прочно вошли в практику компьютерного дела.

Идеи трансляции (перекодирования) одних символов в другие легли в основу создания различных языков программирования с соответствующими трансляторами - компиляторами и/или интерпретаторами. Отличие компиляторов от интерпретаторов заключается в процедуре трансляции текста в машинный код. Компилятор преобразует весь текст программы в последовательный набор машинных команд, который в дальнейшем отправляется на выполнение (пример компилятора с языка Паскаль). Интерпретатор же осуществляет трансляцию по принципу синхронного перевода. Каждая отдельная строка программного текста транслируется, а затем, после ее интерпретации, команды этой строки выполняются (пример языка Бейсик). Современные трансляторы с языков программирования высокого уровня, систем управления базами данных интегрируют в себе возможности и достоинства компиляторов и интерпретаторов, а в системы программирования добавляют различные сервисные утилиты по трансляции и отладке создаваемых программ.

Важнейшим элементом в развитии систем программирования выступили подпрограммы. Появление аппарата подпрограмм существенно облегчило процесс разработки системных и прикладных программ. Подпрограммы позволили формировать библиотеки из наиболее часто употребляемых в программах алгоритмов -процедур и функций. В системах программирования обязательно присутствуют стандартные (встроенные в систему) библиотеки подпрограмм. Например, в их число входят подпрограммы вычисления математических функций sin(х), cos(x), abs(х) и др.

В настоящее время распространены пользовательские и прикладные библиотеки подпрограмм. Их число увеличивается. Меняется структура библиотечных подпрограмм. В современных языках получили распространение модули (**Unit**), представляющие специализированные пакеты взаимосвязанных подпрограмм определенного предназначения, например по работе с клавиатурой, с графикой и пр. Развитие объектно-ориентированного программирования позволило создавать библиотеки объектов и подпрограмм с объектными типами данных (**Object**). Примером могут служить оболочки типа **TurboVision**.

Современная программа представляет набор команд, операторов и выражений, в которых имеются ссылки (прямые или косвенные) на различные подпрограммы из существующих в системе программирования библиотек, модулей, объектов. В этой связи исходный текст программы, как правило, занимает по объему места в памяти в несколько раз меньше, чем его оттранслированный вариант в машинных кодах. Как это происходит?

Рассмотрим один из вариантов трансляции программы с языка программирования Паскаль. Исходный текст программы решения квадратного уравнения представленниже**:**

program KvadUravn;

var А, В, С, D, XI, Х2: REAL;

begin

writeln;

writeln( 'введи А,В,С') ; read(A,B,C);

D:=B\*B-4\*A\*C;

if D<0 then write('корней нет')

else begin

X1:=(-B+sqrt(D))/(2\*A);

X2:=(-B-sqrt(D))/(2\*A);

write('X1=', X1,' X2=', Х2);

end

end.

Предположив, что этот текст (по отношению к процессу трансляции выступающий как **исходный** модуль) сформирован одним из текстовых редакторов, попытаемся отправить его на выполнение. Прежде всего его необходимо перевести в машинный двоичный код (называемый **абсолютным** или **загрузочным модулем**). Для этого на первых этапах осуществляется трансляция (в данном случае, как это реализовано в системах программирования Паскаля, компиляция) исходного текста в машинный код (**объектный модуль**). Однако, объектный модуль не может быть использован для выполнения программы, поскольку в нем нет программ по выполнению процедур ввода (read) и вывода (write, writeln), а также вычисления функции извлечения квадратногокорня(sqrt). В исходном тексте программы ссылкинауказанные библиотечные подпрограммы отмечены знаком {\*}.

Следующий шаг трансляции - компоновка - заключается в подключении к исходному объектному модулю объектных модулей соответствующих подпрограмм в места ссылок на них (исходные тексты этих подпрограмм в системе вовсе отсутствуют). Другими словами, на место процедуры Write помещается подпрограмма, осуществляющая процедуру вывода данных на экран дисплея. Таким образом после компоновки (или, иначе, редактирования связей link editor) возникает абсолютный модуль, намного превышающий по объему размер исходного текста программы, Он и является исполняемым компьютером после его запуска. Расширениями его файлового имени, как правило, являются .com или .ехе.

В силу того, что объектные модули не предназначены для непосредственного исполнения, в них обычно нет привязки составляющих их машинных команд к конкретному месту в ОЗУ. Адреса машинных слов бывают условными, что помогает компоновщику размещать объектные модули в свободныхместах ОЗУ (заменяя условные адреса команд на конкретные).

Многие системы программирования дополнительно содержат промежуточные этапы трансляции. В этих системах на первом шаге предусмотрена трансляция исходного текста в макроассемблерный код, а затем в объектный модуль. Это связано с историей развития языков программирования, а также с тем, что многие подпрограммы удобнее писать на языке Ассемблера, и подключать их легче на этапе линко-вания ассемблерного модуля с ассемблерными библиотеками подпрограмм.

В современных системах программирования, например, Турбо-Паскаль, Турбо-Си весь этот сложный процесс трансляции с компоновкой подпрограмм скрыт от пользователя и осуществляется специальными компиляторами.

Коротко об отладчиках.Эти программы входят в современные системы программирования и предоставляют средства для просмотра и изменения значений переменных в ходе отладки программы, поиска ошибок и т.д. Использование отладчиков значительно облегчает процесс доводки больших программ.

Заметим, что описанный процесс трансляции характерен для компиляции. Последовательно реализованный интерпретатор объектного модуля фактически не создает. В этом его и недостаток, и достоинство (экономия машинной памяти). Впрочем, у современных ЭВМ, в том числе и персональных, проблема малого ОЗУ отходит на второй план, и интерпретация встречается все реже, так как эффективность этого процесса в целом значительно ниже.

Остается непонятным, как детально происходит трансляция. Пользователь может не уметь сам вручную оттранслировать программу (даже столь короткую, как вышеприведенная), но элементарное понимание этого сложного процесса необходимо.

На первом этапе транслятор производит синтаксический анализ исходной программы - проверяет, не нарушены ли формальные правила, содержащиеся в данном языке программирования. Например, в Паскале текст может встретиться либо внутри текстовой константы (т.е. в апострофах), либо внутри комментария. Если такой текст встретился в другом месте, то это явная ошибка. В системе программирования встроены описания всех синтаксически разрешенных конструкций, и транслятор их применяет к исходной программе. Для задания синтаксиса применяются формы Бэкуса-Наура и синтаксические диаграммы, о которых будет рассказано в следующей главе.

Первой фазой синтаксического анализа является лексический анализ. Он заключается в просмотре литер исходной программы и построении из них лексически допустимых единиц - идентификаторов, ключевых слов языка, чисел и т.д. Во второй фазе эти единицы уже рассматриваются как неделимые и проверяется допустимость их сочетания.

Даже если в синтаксическом смысле исходная программа верна, это не означает, что она имеет смысл в рамках данного языка программирования. На следующем этапе семантического анализа транслятор ищет ошибки такого рода: числа употребления слов BEGIN и END не совпадают; переменные не описаны (в языке, требующем обязательного явного описания переменных), т.е. текст программы *непонятен* (семантика - смысловая сторона языка).

Лишь после того, как в программе все синтаксически правильно и семантически понятно, транслятор переводит операторы программы в машинный код. Это отнюдь не означает, что в программе все благополучно - не исключены ошибки этапа исполнения (деление на ноль, выход за границу массива, переполнение разрядов и т.д.).

Различные фазы компиляции могут быть как последовательными, так и частично перекрывающимися во времени. В зависимости от способа реализации компилятор читает и обрабатывает исходный текст один илинесколько раз, называясь соответственно однопроходным, двухпроходным и т.д.