СОДЕРЖАНИЕ

1а **1. Системы программирования**

Машинно-ориентированные языки являются машинно-зависимым языком программирования. Основные конструктивные средства подобных языков дают возможность учитывать особенности архитектуры и принципов работы каждой ЭВМ.

Они позволяют записывать программу в виде, допускающем ее реализацию на ЭВМ с различными типами машинных операций, привязка к которым осуществляется соответствующим транслятором.

Язык СИ обладает некоторыми особенностями:

1) максимально используются возможности определенной вычислительной архитектуры; из-за этого программы на языке СИ компактны и работают эффективно;
2) обладает максимальными возможностями использования огромных выразительных возможностей современных языков высокого уровня.

Процедурно-ориентированные языки чаще всего применяются для описания алгоритмов решения широкого класса задач; среди таких языков — Фортран, Кобол, Бейсик, Паскаль.

Проблемно-ориентированные языки применяются при описании процессов обработки информации в более узкой, специфической области; чаще всего применяются языки: РПГ, Лисп, АПЛ, GPSS.

Объектно-ориентированные языки программирования применяют в случае разработки программных приложений для широкого круга различных задач, которые имеют общность в реализуемых компонентах.

Интерпретация — пооператорная трансляция и последующее выполнение оттранслированного операто-

2а **2. Классификация языков программирования высокого уровня**

Высокоуровневые языки программирования применяются в машинно-независимых системах программирования. Подобные системы программирования в сравнении с машинно-ориентированными системами более просты в применении.

Языки программирования высокого уровня делятся на определенные группы:

1) процедурно-ориентированные языки, которые употребляются для записи процедур или алгоритмов обработки информации на любом круге задач:
   а) язык Фортран (Fortran) (от Formulae Translation — «преобразование формул»). Фортран является одним из старейших языков программирования высокого уровня. Его существование и применение объясняется простотой его структуры;
   б) язык Бейсик (Basic), который можно расшифровать как «Beginner's All-purpose Symbolic Instruction Code» (BASIC) — «многоцелевой символический обучающий код для начинающих», применяется с 1964 г. как язык для обучения программированию;
   в) язык СИ (С), используется с 1970-х гг. как язык системного программирования специально для написания операционной системы UNIX. В 1980-е гг на основании языка С разработали язык С++, который включает в себя язык С и дополнен средствами объектно-ориентированного программирования;

3а **3. Переменные Visual Basic**

В Visual Basic переменные накапливают информацию (значения). При их применении Visual Basic занимают область в памяти компьютера, которая предназначена для сохранения этой информации. Имена переменных, составленные из символов, могут иметь длину в 255 символов. Они начинаются с буквы, затем могут находиться другие буквы, цифры или символы подчеркивания. Регистр символов и наименований переменной не важен.

Все символы в имени переменной значимы, но их регистр не имеет значения. BASE обозначает такую же переменную, что и base. Но Base, Base 1 и Base_ I являются различными переменными. Visual Basic всегда заменяет первую букву переменной заглавной при определении.

Применение осмысленных имен помогает документировать текст программы и позволяет сделать процесс ее отладки намного легче. Выразительное имя переменной служит прекрасным способом объяснения смысла применения многих инструкций в коде программы.

Именем новых переменных не могут быть зарезервированные слова; например, Print не подходит для этого. Но такие слова могут использоваться как часть имени переменной, например: PrintIt. Visual Basic будет показывать сообщение об ошибке, когда программист использует зарезервированное слово как название своей переменной, причем обычно непосредственно после нажатия клавиши ENTER.

Одно из наиболее общих соглашений об именах переменных состоит в использовании заглавных букв

4а **4. Типы переменных**

Integer

Целочисленные переменные способны хранить только не очень большие целые числа, которые располагаются в диапазоне от –32768 до +32767. Арифметические операции над подобными числами производятся очень быстро. Для обозначения подобного типа применяется символ «%».

Long Integer

Подобный тип впервые был применен в языке QuickBASIC. В этих переменных располагаются целые значения от –2 147 483 648 до +2 147 483 647. Обозначается символом «&». Арифметические действия над приведенными числами выполняются тоже очень быстро, и в случае работы с процессором 386DX или 486DX обнаруживается только небольшая разница в скорости вычислений между Long Integer и Integer.

Single Precision

Идентификатором для таких чисел является символ «!». Такой тип переменной дает возможность хранить дробные числа, точность которых до седьмой цифры. То есть если получается результат 12345678.97, то часть 8.97 не точна. Результат может иметь значение, к примеру, 12345670.01. Длина чисел может иметь 38 знаков. Произведения математических операций с данными переменными тоже будут приблизительными. Кроме того, арифметические действия производятся медленнее, чем с целочисленными переменными.

Double Precision

Переменные подобного типа дают возможность хранить числа с точностью до 16 цифр и длиной до 300

2б г) язык Паскаль (Pascal) получил свое название в честь французского ученого Б. Паскаля. Его начал применять с 1968—1971 гг. Н. Вирт. При создании Паскаль использовали для обучения программированию, но впоследствии он стал применяться для разработки программных средств в профессиональном программировании;

2) проблемно-ориентированные языки применяются для разрешения целых классов новых задач, которые появляются при постоянном расширении области применения вычислительной техники:
   а) язык Лисп (Lisp — List Information Symbol Processing) изобрел в 1962 г. Дж. Маккарти. Изначально он использовался как средство работы со строками символов. Лисп применялся в экспертных системах, системах аналитических вычислений и т. п.;
   б) язык Пролог (Prolog — Programming in Logic) предназначается для логического программирования в системах искусственного интеллекта;
   3) объектно-ориентированные языки, которые развиваются и в наше время. Большинство из таких языков — развитые версии процедурных и проблемных языков, но программирование с помощью языков такой группы более наглядно и просто. Среди таких языков можно выделить следующие:
   а) Visual Basic (Basic);
   б) Delphi (Pascal);
   в) Visual Fortran (Fortran);
   г) C++ (C);
   д) Prolog++ (Prolog).

1б ра исходной программы. Существуют следующие основные недостатки метода интерпретации:

1) интерпретирующая программа должна находиться в памяти ЭВМ в течение всего процесса осуществления исходной программы. То есть она должна занимать некоторый определенный объем памяти;
2) процесс трансляции одного и того же оператора повторяется столько раз, сколько должна исполнять эта команда в программе. Это является причиной резкого снижения производительности работы программы.

Но трансляторы-интерпретаторы широко распространены, так как они поддерживают диалоговый режим.

Процессы трансляции и выполнения при компиляции делятся во времени: первоначально исходная программа в полном объеме переводится на машинный язык, потом оттранслированная программа может многократно исполняться. Для трансляции методом компиляции нужен неоднократный «просмотр» транслируемой программы, т. е. трансляторы-компиляторы многопроходны. Трансляция методом компиляции именуется объектными модулями. Это эквивалентная программа в машинных кодах. Нужно, чтобы перед исполнением объектный модуль обработался особой программой операционной системы и преобразовался в загрузочный модуль.

Применяют кроме этого трансляторы интерпретаторы-компиляторы, которые объединяют в себе достоинства обоих принципов трансляции.

4б символов. Идентификатором служит «#». Вычисления с ними тоже приблизительны, а скорость их не очень большая. Чаще всего переменные типа Double Precision применяются для научных расчетов.

Currency

Этого типа не существовало в версиях GW-BASIC и QuickBASIC. Его применяют для того, чтобы не допускать ошибок при преобразовании десятичных чисел в двоичную форму и наоборот. Такой тип может иметь до 4 цифр после запятой и до 14 — перед ней. Внутри этого диапазона вычисления являются точными. Идентификатор такой переменной — символ «@». Так как все арифметические операции, кроме сложения и вычитания, производятся так же медленно, как и в случае переменных с двойной точностью, такой тип более предпочтителен для проведения финансовых расчетов.

Date

С помощью такого типа данных можно хранить значения времени и даты в промежутке от полуночи 1 января 100 года до полуночи 31 декабря 9999 года. Подобные значения в тексте программ обозначенысимволами «#», например: Millenium = #January 1, 2000#.

При введении только значения даты Visual Basic полагает, что время соответствует 00:00.

3б в начале каждого из слов, составляющих данное имя (например, PrintIt, а не Printit). Данное соглашение называется «имена переменных со смешанным регистром». Иногда применяется и символ подчеркивания (например, Print_It), но его применяют не часто, так как это отнимает много места и иногда вызывает проблемы при отладке.

Visual Basic способен работать с 14 стандартными типами переменных. Также можно определить собственный тип данных. Рассмотрим некоторые из них, которые в основном применяются при работе с данными.

String

Строковые переменные предназначены для того, чтобы хранить символы. Обозначить такой тип можно несколькими способами. Например, обозначать данный тип переменной с помощью добавления символа «$» к концу ее имени, например: AStringVariable$. Теоретически данная переменная может иметь до нескольких миллиардов символов. Однако на компьютере данное число будет намного меньше, так как накладываются ограничения на объемы оперативной памяти, ресурсы Windows или число символов, используемых в форме.

Наиболее часто строковые переменные применяются для выбора из полей ввода. К примеру, если есть поле ввода с именем Text1, в этом случае оператор ContentOfText1S = Text1.Text присваивает строку из поля ввода переменной в левой части такого оператора.

5а

5. Целочисленные, переменного типа и переменные данных

Byte

Байтовый тип нов в Visual Basic и используется для хранения целых чисел от 0 до 255. Его применение дает возможность значительно экономить оперативную память и сократить размер массивов по сравнению с предыдущими версиями Visual Basic. К тому же его применяют при работе с двоичными файлами.

Boolean

Булев тип данных способен хранить только два значения: True или False. Его применение вместо целочисленных переменных представляет собой хороший стиль программирования.

Variant

Такой тип был введен в Visual Basic 5 из версии 2.0. Переменная типа variant способна содержать данные любого типа. Если Visual Basic не распознает тип принимаемых данных, следует использовать variant.

Тип информации не имеет значения, так как variant способен содержать любой тип данных (численный, дата и время, строковый). Visual Basic автоматически совершает необходимые преобразования данных, т. е. не стоит беспокоиться об этом. Однако можно применять встроенные функции для проверки типа данных, которые хранятся в переменной типа variant. С их помощью можно легко проверить, правильно ли пользователь вводит информацию.

Применение variant делает работу программы более медленной, так как необходимо время и ресурсы для того, чтобы произошло преобразование типов. К тому

6а

6. Объявление переменных: оператор Dim для различных типов данных

Чаще всего люди стараются не пользоваться идентификаторами при обозначении типа переменной (тем более для таких типов, как дата/время). Вместо этого они применяют оператор Dim. Подобная операция называется объявлением. Объявление типов переменных при осуществлении обработки событий перед их использованием — естественно, с комментариями — представляет собой хороший стиль в программировании. Это также дает возможность улучшить «читабельность» текстов программ.

Если переменную объявили с помощью оператора Dim, в случае применения переменной с тем же именем и другим идентификатором типа будет наблюдаться ошибка «двойное определение» при запуске программы. К примеру, если следующее выражение Dim Count As Integer объявляет переменную Count, то нельзя применять переменные CountS, Count!, Count# и Count@. Следует использовать только имя Count%, но это всего лишь другая форма для имени переменной Count.

Чтобы присвоить переменной тип variant, используют оператор Dim без As:

Dim Foo считает Foo переменной типа variant.

Можно написать и следующим образом: Dim Foo As Variant — это проще для прочтения.

Каждая информация, которая должна быть доступна всем процедурам обработки событий, относящихся к форме, размещается в разделе (General) данной формы.

7а

7. Изменение значений по умолчанию для типов, область видимости

Пусть в новой программе почти все переменные являются целочисленными. Тогда удобно осуществлять их объявление так, чтобы переменная, для которой не указан тип, больше не объявлялась как variant. Для этого применяется оператор DefType.

Например, определить соглашение, что все переменные, которые начинаются с I, будут целочисленными, с помощью DefInt I. Затем оператор Dim I будет объявлять переменные типа integer. Основные типы разных операторов DefType, которые чаще всего используются:

DefInt диапазон букв (для integer);
DefLng диапазон букв (для long integer);
DefSng диапазон букв (для single precision);
DefDbl диапазон букв (для double precision);
DefCur диапазон букв (для currency);
DefStr диапазон букв (для string);
DefVar диапазон букв (для variant);
DefBooi диапазон букв (для boolean);
DefByte диапазон букв (для byte);
DefDate диапазон букв (для date).

Не всегда здесь применяются заглавные буквы: DefStr s-Z и Def-Str S-Z функционируют одинаково. Соглашения о формах можно всег-да изменить, применяя идентификатор или оператор Dim для каждой из переменных. Оператор DefType располагают в том же разделе (General), что и Option Explicit.

При программировании используют термин «область видимости», если хотят сказать о возможности применения переменной из одной части программы в другой

8а

8. Используемые символы языка СИ

Практически все символы, которые применяются в языке СИ, можно разделить на пять групп:

1. Символы, которые применяются для образования ключевых слов и идентификаторов. К ним относят прописные и строчные буквы английского алфавита и символ подчеркивания. Важно отметить, что одинаковые прописные и строчные буквы являются различными символами, так как обладают различными кодами.

2. Прописные и строчные буквы русского алфавита и арабские цифры.

3. Символы, образующие нумерацию, и специальные символы. Данные символы применяются для организации процесса вычислений, а также для передачи компилятору некоторого набора инструкций.

4. Управляющие и разделительные символы. К данной группе относят: пробел, символы табуляции, перевода строки, возврата каретки, новая страница и новая строка. Подобные символы призваны отделять друг от друга объекты, определяемые пользователем, среди которых константы и идентификаторы. Ряд разделительных символов считается компилятором как один символ (последовательность пробелов).

5. Кроме представленных групп символов в языке СИ широко применяются так называемые управляющие последовательности. Это специальные символьные комбинации, которые используются в функциях ввода и вывода информации. Управляющая последовательность начинается с обратной дробной черты (\) (обязательный первый символ) и комбинации латинских букв и цифр.

6б Для размещения Option Explicit в раздел (General), следует выполнить следующие действия.

1. Открыть окно Code.

2. Выбрать объект (General) из списка объектов, которые предлагаются в окне Object.

3. Выбрать (Declaration) из списка Proc.

4. Ввести Option Explicit.

Часто применять объявления на уровне формы в разделе (General) необходимо, когда пользователь экспериментирует с примерами программ из справочной системы.

Для того чтобы копировать пример программы из справочной системы, следует использовать кнопку Copy в окне Code для примера. После этого можно использовать пункт Paste из меню Edit для помещения примера в окно Code. Когда Visual Basic встречает команду Option Explicit, он перестает позволять использовать необъявленные переменные. Если попробовать все же использовать такую переменную, будет показано сообщение об ошибке.

Для применения обязательного объявления типа переменной можно пользоваться страницей Editor диалоговой панели Tools|0ptions. Программист всегда устанавливает такой флажок. После этого оператор Option Explicit автоматически помещается в те места кода, где это необходимо.

5б же многие программисты понимают, что применение автоматических преобразований типов данных является причиной неаккуратного вида программ. Причина использования variant заключается в возможных ошибках при преобразовании типов непосредственно.

В отличие от множеств других версий BASIC, в программе Visual Basic нельзя применять имена переменных, которые отличаются только типом (идентификатором), например A% и A!. В случае попытки применения двойного имени возникает ошибка «двойное определение» (duplicate difmition), когда происходит запуск программы.

При первом применении переменной Visual Basic временно присваивает ей пустое значение и тип variant. Такое значение пропадает в тот момент, когда переменной присваивают реальное имя. Любой тип данных имеет свое «пустое» значение. В случае строковых переменных это строка нулевой длины («»). В случае численных переменных — ноль. Полагаться следует только на значения по умолчанию, если они являются документированными (например, в комментариях). В противном случае появится множество трудно уловимых ошибок. Исходя из этого, следует инициализировать величины переменных в первых строках процедур обработки событий.

Одной из самых распространенных задач является обмен значениями между двумя переменными. Важно отметить, что разработчики Visual Basic убрали из языка оператор Swap, применяемый в QuickBASIC. Поэтому код следует писать самим.

8б Последовательности вида \ddd и \xddd (d является цифрой) дают возможность представить символ из кода ПЭВМ как ряд восьмеричных или шестнадцатеричных цифр соответственно. К примеру, символ возврата каретки можно записать различными способами:

1) \r — общая управляющая последовательность;
2) \015 — восьмеричная управляющая последовательность;
3) \x00D — шестнадцатеричная управляющая последовательность.

Важно отметить, что в строковых константах следует задавать все три цифры в управляющей последовательности. Так, отдельную управляющую последовательность \n (переход на новую строку) можно переписать так: \010 или \xA. Однако в строковых константах нужно задавать все три цифры, иначе символ или символы, идущие за управляющей последовательностью, будут считаться ее недостающей частью. К примеру: «ABCDE\x009FGH». Эта строковая команда будет напечатана с применением определенных функций языка СИ как два слова ABCDE FGH, между которыми располагаются 8 пробелов. Если указать неполную управляющую строку «ABCDE\x09FGH», то на печати появится ABCDE=|=GH, вследствие того, что компилятор воспримет последовательность \x09F как символ «=+=». Важно отметить, что если обратная дробная черта предшествует символу, не являющемуся управляющей последовательностью (т. е. не включенному в табл. 4) и не являющемуся цифрой, то данная черта не учитывается, а символ представляется как литеральный.

7б ее части. В старых языках все переменные были применимы во всех частях программы, поэтому сохранение целостности наименований было огромной проблемой. Например, если в приложении применялись две переменные Total, то их значения уничтожали друг друга.

Решение подобной проблемы в современных языках высокого уровня, среди которых и Visual Basic, заключается в изолировании переменных внутри процедур. Пока это не определено специальным образом, значение переменной Total в одной процедуре никак не влияет на переменную с таким же именем в другой процедуре. На языке программирования такой подход означает, что данные переменные локальны по отношению к процедурам, если не указано иначе. Например, процедура обработки события чаще всего не имеет никакой связи с другой процедурой того же типа. Обычно в работе не используют переменные по умолчанию. Если нужно быть уверенным, что эта переменная является локальной, нужно объявить ее внутри процедуры события, применяя при этом оператор Dim.

9a

9. Константы языка СИ

Константы — это перечисление величин в программе. В языке СИ можно выделить четыре вида констант: целые константы, константы с плавающей запятой, символьные константы и строковые литералы.

Целая константа — это десятичное, восьмеричное или шестнадцатеричное число, представляющее целую величину в одной из известных форм: десятичной, восьмеричной или шестнадцатеричной. Десятичная константа включает в себя одну или несколько десятичных цифр, при этом первая цифра не должна быть нулем (иначе число будет воспринято как восьмеричное). Восьмеричная константа включает в себя обязательный нуль и одну или несколько восьмеричных цифр (среди цифр не должно быть восьмерки и девятки, так как данные цифры не входят в восьмеричную систему счисления). Шестнадцатеричная константа начинается с непременной последовательности 0x или 0X и включает в себя одну или не-сколько шестнадцатеричных цифр, которые являются набором цифр шестнадцатеричной системы счисления: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Если необходимо образовать отрицательную целую константу, то применяется знак «–» перед записью константы (который называется унарным минусом). К примеру: –0x3A, –098, –36. Любой целой константе присваивается тип, который определяет преобразования, которые должны быть выполнены, если константа применяется в выражениях. Тип константы определяется так:

1) десятичные константы — это величины со знаком, и

10a

10. Примеры использования констант языка СИ

119.75, 1.7E2, –0.825, 0.035, –0.89E2.

Символьная константа является символом, который заключен в апострофы. Управляющая последовательность является одиночным символом, допустимо ее применять в символьных константах. Значением символьной константы считается числовой код символа.

Примеры:

‘ ‘ — пробел;
‘\\’ — обратная дробная черта;
‘W’ — буква W;
‘\n’ — символ новой строки;
‘\v’ — вертикальная табуляция.

Символьные константы принадлежат типу int, и при преобразовании типов к ним приписывается соответствующий знак.

Строковая константа (литерал) представляет собой последовательность символов, среди которых строковые и прописные буквы русского и латинского алфавита или цифры, которые заключены в кавычки («»). Например: «Школа N 35», «город Саратов», «YZPT КОД». Важно заметить, что все управляющие символы, кавычки («»), обратная дробная черта (\) и символ новой строки в строковом литерале и в символьной константе являются соответствующими управляющими последовательностями. Любая управляющая последовательность записывается как один символ. К примеру, при печати фразы «Школа \n N 35» часть «Школа» будет располагаться на одной строке, а часть «N 35» — на следующей. Знаки строкового выражения

11a

11. Идентификатор. Ключевые слова

Идентификатором называется последовательность цифр, букв и специальных символов. При этом первой стоит буква или специальный символ. Для получения идентификаторов можно использовать строчные или прописные буквы латинского алфавита. Специальным символом может служить символ подчеркивания (_). Два идентификатора, для получения которых применяются совпадающие строчные и прописные буквы, считают различными. К примеру: abc, ABC, A328B, a328b. Компилятор допускает всякое количество символов в идентификаторе, но значим только первый 31 символ. Идентификатор образуется на этапе объявления переменной, функции, структуры и т. п. После этого его можно применять в последующих операторах разрабатываемой программы. Важно отметить некоторые особенности при выборе идентификатора. Во-первых, идентификатор и ключевое слово не должны совпадать. Также не должно быть совпадения с зарезервированными словами и названиями функций библиотеки компилятора языка СИ.

Во-вторых, важно обратить особое внимание на применение символа подчеркивания (_) первым символом идентификатора, так как идентификаторы выстраиваются так, что, с одной стороны, могут совпадать с именами системных функций и (или) переменных, но при этом при применении таких идентификаторов программы могут стать непереносимыми, т. е. их нельзя применять на компьютерах других типов.

В-третьих, на идентификаторы, применяемые для определения внешних переменных, должны быть на-

12a

12. Комментарии. Исходные файлы

Комментарием является набор символов, игнорируемых компилятором. Но на данный набор символов накладываются определенные ограничения. Внутри набора символов, представляющих комментарий, не может быть специальных символов, которые определяют начало и конец комментариев, соответственно (/* и */). Важно показать, что комментарии способны заменить одну или несколько строк.

Приведем конкретные примеры:

```
/* комментарии к программе */
/* начало алгоритма */
```

или /*

Комментарии могут быть записаны в любом виде, но следует быть осторожным и не допустить внутри последовательности, которая игнорируется компилятором, появления оператора программы, который также будет игнорироваться */. При этом производится неправильное определение комментариев.

```
/* комментарии к алгоритму /* решение краевой задачи */ */
```

или

```
/* комментарии к алгоритму решения */ краевой задачи */
```

Обычная СИ-программа является определением функции main, которая для выполнения определенных действий вызывает другие функции. Связь между функциями производилась по данным через передачу параметров и возврата значений функций. Однако компилятор языка СИ дает возможность также разбить программу на несколько отдельных частей, которые являются исходными файлами, оттранслировать любую

10б сохраняются в области оперативной памяти. В конец любого строкового литерала компилятор прибавляет нулевой символ, который представляется управляющей последовательностью \0. Строковое выражение имеет тип char[]. То есть строка представляется как массив символов. Количество элементов массива соответствует числу символов в строке плюс 1, вследствие того, что нулевой символ (символ конца строки) также служит эле-ментом массива. Все строковые выражения рассматриваются компилятором как различные объекты. Строковые литералы способны находиться на нескольких строках. Такие фразы формируются на основе применения обратной дробной черты и клавиши «ввод». Обратная черта и символ новой строки рассматривается компилятором, в результате чего следующая строка служит продолжением предыдущей. К примеру, «строка неопределенной \n длины» полностью аналогична фразе «строка неопределенной длины». С целью сцепления строковых фраз можно применять символ (или символы) пробела. Когда в программе присутствуют два или более строковых выражения, которые разделены только пробелами, они будут рассматриваться как одна символьная строка. Данный принцип можно применять для формирования строковых литералов, занимающих более одной строки.

9б им присваивается тип int (целая) или long (длинная целая) по значению константы. Если константа меньше 32 768, то ее тип — int, иначе — long;
2) восьмеричным и шестнадцатеричным константам присваивается тип int, unsigned int (беззнаковая целая), long или unsigned long по величине константы.

Для того чтобы каждую целую константу определить типом long, в конце константы ставится буква «l» или «L». Пример: 2l, 9l, 138L, 0905L, OX2911L.

Константа с плавающей точкой — десятичное число, которое представлено в виде действительной величины с десятичной точкой или экспонентой. Формат записывается так: [цифры].[цифры] [E|e [+|–] цифры].

Число с плавающей точкой включает себя целую и дробную части и (или) экспоненты. Константы с плавающей точкой — это положительные величины удвоенной точности, тип которых double. Для того чтобы определить отрицательную величину, следует сформировать константное выражение, которое состоит из знака минуса и положительной константы.

12б часть отдельно и после этого объединить все части в один выполняемый файл при помощи редактора связей. При данной структуре исходной программы функции, располагающиеся в разных исходных файлах, могут применять глобальные внешние переменные. Все функции в языке СИ по определению внешние и постоянно доступны из каждого файла. Для выполнения определяемой функцией каких-либо действий она должна применять переменные. В языке СИ все переменные объявляются до их применения. Объявления определяют соответствие имени и атрибутов переменной, функции или типа. Определение переменной приводит к выделению памяти для хранения ее значения. Класс отводимой памяти определяется спецификатором вида памяти и задает время жизни и область види-мости переменной, которые связаны с понятием блока программы. В СИ блоком является ряд объявлений, определений и операторов, располагающихся в фигурных скобках.

Можно выделить два вида блоков — составной оператор и определение функции, которые состоят из составного оператора (тела функции) и заголовка функции, который находится перед телом функции (в него входят имя функции, типы возвращаемого значения и формальных параметров). Блоки могут состоять из операторов, но не определения функций. Внутренний блок носит название вложенного, а внешний — объемлющего.

Временем при жизни называется интервал времени выполнения программы, за который программный объект (переменная или функция) существует. Время жизни переменной бывает локальным или глобальным. Переменная с глобальным временем жизни обладает распределенной для нее памятью и определенным значением на протяжении всего времени выполнения программы.

11б ложены ограничения, которые формируются используемым редактором связей. Кроме того, использование различных версий редактора связей или различных редакторов определяет различные требования на имена внешних переменных.

Ключевыми словами называются зарезервированные идентификаторы, наделенные определенным смыслом. Их можно применять только в соответствии со значением, известным компилятору языка СИ. Приведем список ключевых слов:

auto double int struct break else long switch register tupedef char extern return void case float unsigned default for signed union do if sizeof volatile continue enum short while.

При этом в определенных версиях реализации языка СИ зарезервированными словами являются следующие: _asm, fortran, near, far, cdecl, huge, pascal, interrupt.

Ключевые слова far, huge, near дают возможность определить размеры указателей на области памяти. Ключевые слова _asm, cdecl, fortran, pascal используются для организации связи с функциями, которые написаны на других языках, а также для применения команд языка ассемблера непосредственно в теле будущей программы на языке СИ. Ключевые слова не могут применяться в качестве идентификаторов.

13а

13. Область видимости

Областью при видимости называется часть текста программы, в которой может быть использован определенный объект. Объект является видимым в блоке или в исходном файле, когда в данном блоке или файле определены имя и тип объекта. Объект может быть видимым внутри блока, исходного файла или в каждом исходном файле, образующем программу. Это определяется тем, на каком уровне объявлен объект: на внутреннем (внутри определенного блока) или на внешнем (вне всех блоков). Когда объект объявлен внутри блока, он является видимым в данном блоке и в каждом внутреннем блоке. Когда объект объявлен на внешнем уровне, он является видимым от точки его объявления до завершения этого исходного файла. Объект можно сделать глобально видимым с помощью определенных объявлений во всех исходных файлах, образующих программу. Спецификатором класса памяти в объявлении переменной может быть auto, register, static или extern. Если класс памяти определен, он определяется по умолчанию из контекста объявления. Объекты, принадлежащие классам auto и register, обладают локальным временем жизни. Спецификаторы static и extern задают объекты, обладающие глобальным временем жизни. В случае объявления переменной на внутреннем уровне можно применить любой из четырех спецификаторов класса памяти, а если его не указали, то подразумевается auto. Переменная с классом памяти auto обладает локальным временем жизни и видна только в блоке, в котором объявлена. Память для данной переменной выделяется при входе в блок и высвобождается при выходе из блока. В случае повторного

14а

14. Объявление переменной на внутреннем уровне с классом памяти static

В качестве примера рассмотрим объявление переменной i на внутреннем уровне с классом памяти static.

```
исходный файл file1.c
main()
{
}
fun1()
{static int i = 0;
исходный файл file2.c
fun2()
{static int i = 0;
}
fun3()
{static int i = 0;
}
```

В этом примере объявлены три различные переменные с классом памяти static, которые имеют одинаковые имена i. Все эти переменные обладают глобальным временем жизни, но видимы только в том блоке (функции), в котором они объявлены. Данные переменные можно применять для подсчета числа обращений к каждой из трех функций.

Переменные класса памяти static способны быть инициализированными константным выражением. Когда явной инициализации нет, то данной переменной присваивается нулевое значение. В случае инициализации константным адресным выражением

15а

15. Объявление переменной, которая служит именем внешнего массива

Рассмотрим пример: объявление переменной i, которая служит именем внешнего массива длинных целых чисел, на локальном уровне.

```
исходный файл file1.c
main()
{…
}
fun1()
{extern long i[]; ...
}
/* исходный файл file2.c */
long i[MAX] = {0};
fun2()
{…
}
fun3()
{…
}
```

Объявление переменной i[] как extern в рассмотренном примере делает ее видимой в функции fun1. Определение данной переменной находится в файле file2.c на глобальном уровне и должно быть единственным. При этом объявлений с классом памяти extern может быть много.

Объявление переменной со спецификатором extern дает знать компилятору о том, что память для переменной не нужна, так как это выполнено где-то в другом месте программы.

16а

16. Методы доступа к элементам массивов

Доступ к элементам массива может производиться двумя различными способами.

Первый способ связан с применением обычных индексных выражений в квадратных скобках, например: array[18] = 3 или array[i + 3] = 9. При данном способе доступа записываются два выражения. Второе выражение должно быть заключено в квадратные скобки. Одно из данных выражений должно являться указателем, а второе — выражением целого типа. Последовательность записи данных выражений может быть произвольной, однако в квадратных скобках следует записывать выражение, следующее вторым. Поэтому записи array[16] и 16[array] будут являться одинаковыми и обозначающими элемент массива с номером шестнадцать. Указатель, который используется в индексном выражении, не всегда является константой, которая указывает на какой-либо массив, это может быть и переменная. Например, после выполнения присваивания ptr = array доступ к шестнадцатому элементу массива можно получить, применяя указатель ptr в форме ptr[16] или 16[ptr].

Второй способ доступа к элементам массива связан с применением адресных выражений и операции разадресации в виде *(array+16) = 3 или *(array+i+2) = 7. При данном способе доступа адресное выражение соответствует адресу шестнадцатого элемента массива, тоже может быть записано различными способами: *(array+16) или *(16+array).

При работе на компьютере первый способ приводится ко второму, т. е. индексное выражение стано-

14б можно применять адреса любых внешних объектов, кроме адресов объектов, для которых класс памяти auto, так как их адрес не является константой и меняется при любом входе в блок. Инициализация осуществляется один раз при первом входе в блок.

Переменная, которая объявлена локально с классом памяти extern, служит ссылкой на переменную с таким же именем, определенную глобально в каком-либо из исходных файлов программы. Цель подобного объявления заключается в том, чтобы сделать определение переменной глобального уровня видимым внутри блока.

13б входа в блок этой переменной может быть выделен другой участок памяти. Переменная класса auto автоматически не инициализируется, так как она должна быть проинициализирована явно в случае объявления через присвоение ей начального значения. Значение неинициализированной переменной, класс памяти которой auto, считается неопределенным.

Спецификатор класса памяти register заставляет компилятор распределить память для переменной в регистре, если это возможно. Употребление регистровой памяти чаще всего приводит к сокращению времени доступа к переменной. Переменная, которая объявлена с классом памяти register, обладает той же областью видимости, что и переменная auto. Количество регистров, которые можно применить для значений переменных, не безгранично, так как не безграничны и возможности компьютера. В случае когда компилятор не обладает свободными регистрами, переменной выделяется память как для класса auto. Класс памяти register может указываться для переменных с типом int или указателей с размером, равным размеру int. Переменные, которые объявлены на внутреннем уровне со спецификатором класса памяти static, дают возможность сохранить вид переменной при выходе из блока и применять ее при повторном входе в блок. Данная переменная обладает глобальным временем жизни и областью видимости внутри блока, в котором она объявлена. Для переменных с классом static память выделяется в сегменте данных. В отличие от них переменные класса auto имеют память, которая выделяется в стеке. Исходя из этого, значение переменных с классом static сохраняется при выходе из блока.

16б вится адресным. Для ранее рассмотренных примеров array[16] и 16[array] преобразуются в *(array+16).

Для доступа к начальному элементу массива, т. е. к элементу с нулевым индексом, можно применять просто значение указателя array или ptr. Любое из присваиваний

```
*array = 2;
array[0] = 2;
*(array+0) = 2;
*ptr = 2;
ptr[0] = 2;
*(ptr+0) = 2;
```

присваивает начальному элементу массива значение 2, но быстрее всего выполнятся присваивания *array = 2 и *ptr = 2, так как в них не требуется выполнять операции сложения.

15б В случае объявления переменных на глобальном уровне можно применить спецификатор класса памяти static или extern. Кроме того, можно объявлять переменные без указания класса памяти. Классы памяти auto и register для глобального объявления применять нельзя.

Объявление переменных на глобальном уровне представляет собой или определение переменных, или ссылки на определения, которые сделаны в другой части программы. Объявление глобальной переменной, инициализирующее данную переменную (явно или неявно), служит определением переменной. Определение на глобальном уровне может быть задано в нескольких формах.

1. С помощью класса памяти static. Данная переменная может быть инициализирована явно константным выражением либо по умолчанию нулевым значением. То есть объявления static int i = 0 и static int i одинаковы, и в том и в другом случае переменной i будет присвоено значение 0.

2. Переменная может быть объявлена без указания класса памяти, но с явной инициализацией. Подобной переменной по умолчанию присваивается класс памяти static. То есть объявления int i = 1 и static int i = 1 будут одинаковы.

17a

17. Директивы препроцессора

Директивы препроцессора — это особые инструкции, которые записаны в тексте программы на СИ и выполнены до трансляции программы. Директивы препроцессора дают возможность изменить текст программы. Среди таких действий — замена некоторых лексем в тексте, вставка текста из другого файла, запрет на трансляцию части текста и т. п. Все директивы препроцессора должны начинаться со знака #. После директив препроцессора точки с запятой быть не должно.

Директива #include включает в программу содержимое определенного файла. Эта директива может быть представлена в двух формах:

```
#include «имя файла»
#include <имя файла>
```

Имя файла должно соответствовать соглашениям операционной системы. Оно может включать в себя либо только имя файла, либо имя файла с предшествующим ему маршрутом. Когда имя файла указано в кавычках, то поиск файла производится по заданному маршруту, а при его отсутствии — в текущем каталоге. Когда имя файла задано в угловых скобках, поиск файла осуществляется в обычных директориях операционной системы, которые задаются командой PATH.

Директива #include может являться вложенной, т. е. во включаемом файле тоже может содержаться директива #include, способная замещаться после включения файла, содержащего эту директиву.

Директива #include часто применяется для включения в программу так называемых заголовочных фай-

18a

18. Применение директив

Рассмотрим пример:

```
#define WIDTH 80
#define LENGTH (WIDTH+10)
```

Данные директивы заменят в тексте программы каждое слово WIDTH на число 80 и любое слово LENGTH на выражение (80+10) вместе с окружающими его скобками.

Скобки, которые содержатся в макроопределении, дают возможность избежать недоразумений, связанных с порядком вычисления операций. К примеру, если в скобках выражение t = LENGTH*7 будет преобразовано в выражение t = 80 + 10*7, а не в выражение t = (80 + +10)*7, как это получается, если есть скобки, в результате будем иметь 780, а не 630.

Во второй синтаксической форме в директиве #define присутствует список формальных параметров, который может включать в себя один или несколько идентификаторов, которые разделены запятыми. Формальные параметры в тексте макроопределения отмечают позиции, на которые должны быть подставлены фактические аргументы макровызова. Любой формальный параметр способен появиться в тексте макроопределения несколько раз.

При макровызове за идентификатором следует список фактических аргументов, количество которых следует сделать совпадающим с количеством формальных параметров.

Пример:

```
#define MAX(x,y) ((x) > (y))?(x) : (y)
```

19a

19. Рекурсия

Функция является рекурсивной, когда во время обработки появляется ее повторный вызов непосредственно или косвенно, через цепочку вызовов других функций.

Прямая (непосредственная) рекурсия — это вызов функции внутри тела этой функции.

```
int a()
{.....a().....}
```

Косвенная рекурсия — это рекурсия, которая осуществляет рекурсивный вызов функции через цепочку вызова других функций. Все функции, которые входят в цепочку, тоже являются рекурсивными. Рассмотрим пример:

```
a(){.....b().....}
b(){.....c().....}
c(){.....a().....}.
```

Все представленные функции a, b, c считаются рекурсивными, так как в случае вызова одной из них производится вызов других и самой себя.

Последовательность вызовов процедуры tn, если m = 3, можно проиллюстрировать древовидной структурой (рис. 2). Всякий раз при вызове процедуры tn под параметры n, i, j, w определяется память и запоминается место возврата. В случае возврата из процедуры tn память, которая выделяется под параметры n, i, j, w, освобождается и становится доступной па-

20a

20. Знакомство с языком СИ++

Рассмотрим ряд программ и частей программ на C++.

Прежде всего, рассмотрим программу, которая выводит строку выдачи:

```
#include

main()
{
cout << «Hello, world\n»; }
```

Строка #include дает знать компилятору, что он включил обычные возможности потока ввода и вывода, которые находятся в файле stream.h. Без таких описаний выражение cout << «Hello, world\n» не имело бы смысла. Операция << («поместить в») следует написать первый аргумент во второй (в нашем случае строку «Hello, world\n» в стандартный поток вывода cout). Строка представляет собой последовательность символов, которые заключены в двойные кавычки. В строке символ обратной косой \, после которого идет другой символ, обозначает один специальный символ; в рассмотренном случае \n служит символом новой строки. Получаем, что выводимые символы состоят из Hello, world и перевода строки. Остальная часть программы

```
main() { ... }
```

задает функцию, названную main. Любая программа

18б Приведенная директива заменит фрагмент t = = MAX(i,s[i]) на выражение t = ((i) > (s[i])?(i) : (s[i]).

Как и в ранее приведенном примере, круглые скобки, в которые заключены формальные параметры макроопределения, дают возможность избежать ошибок, связанных с неправильным порядком осуществления, если фактические аргументы являются выражениями.

Например, если есть скобки, фрагмент

```
t = MAX(i&j, s[i]||j)
```

будет заменен выражением

```
t = ((i&j) > (s[i]||j)?(i&j) : (s[i]||j);
```

а если скобок нет — фрагментом

```
t = (i&j>s[i]||j)?i&j : s[i]||j;
```

где условное выражение вычисляется в другом порядке.

Директива #undef применяется для отмены действия директивы #define. Синтаксис данной директивы следующий: #undef идентификатор.

Директива отменяет операцию текущего определения #define для определенного идентификатора.

17б лов, которые содержат прототипы библиотечных функций, и поэтому чаще всего программы на СИ начинаются с этой директивы.

Директива #define применяется для замены часто использующихся констант, ключевых слов, операторов или выражений определенными идентификаторами. Идентификаторы, которые заменяют текстовые или числовые константы, называются именованными константами. Идентификаторы, которые заменяют фрагменты программ, называют макроопределениями, при этом макроопределения могут иметь аргументы.

Директива #define может быть записана в двух синтаксических формах:

```
#define идентификатор текст
#define идентификатор (список параметров) текст
```

Данная директива заменяет все дальнейшие вхождения идентификатора на текст. Подобный процесс называется макроподстановкой. Текст может быть любым фрагментом программы на СИ, а также может и отсутствовать.

20б должна включать в себя функцию с именем main, и действие программы начинается с выполнения этой функции.

Откуда появились выходной поток cout и код, который реализует операцию вывода, были показаны в stream.h, т. е. были определены их типы, но не было дано каких-либо подробностей относительно их реализации. В стандартную библиотеку входит спецификация пространства и инициализирующий код для cout. Команда компиляции в C++ чаще всего называется CC. Она применяется так же, как команда cc для программ на C. Пусть программа с «Hello, world» находится в файле с именем hello.c, тогда можно ее скомпилировать и запустить приблизительно так ($ — системное приглашение):

```
$ CC hello.c
$ a.out
Hello,world
$
```

a.out является принимаемым по умолчанию именем исполняемого результата компиляции. Если необходимо назвать программу, можно осуществить это с помощью опции -o:

```
$ CC hello.c -o hello
$ hello
Hello,world
$
```

19б мять, которая выделена под параметры n, i, j, w предыдущим вызовом, а управление передается в место возврата.

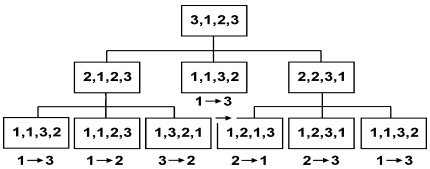

Рис. Последовательность вызовов процедуры tn

Очень часто рекурсивные функции можно заменить нерекурсивными функциями или фрагментами. Это производится путем использования стеков для хранения точек вызова и вспомогательных переменных.

21a

21. Комментарии в СИ++

Часто бывает необходимо вставлять в программу текст, который используется в качестве комментария только для читающего программу человека и не учитывается компилятором в программе. В С++ это возможно осуществить одним из двух способов. Символы /* начинают комментарий, который заканчивается символами */. Вся данная последовательность символов эквивалентна символу пропуска. Это очень полезно для многострочных комментариев и изъятия частей программы в процессе редактирования, но стоит учитывать, что комментарии /* */ не могут быть вложенными.

Символы // начинают комментарий, заканчивающийся в конце строки, на которой они появились. Как и в предыдущем случае, вся последовательность символов соответствует пропуску. Такой способ наиболее удобен для коротких комментариев. Символы // можно применять для того, чтобы закомментировать символы /* или */, а символами /* можно закомментировать //.

Любое имя или выражение имеет тип, который определяет операции, которые могут над ними осуществляться. К примеру, описание

```
int inch;
```

дает понять, что inch имеет тип int, т. е. inch — целая переменная.

Описание представляет собой оператор, который вводит имя в программе. Описание определяет тип этого имени. Тип назначает правильное использова-

22a

22. Соотношение между основными типами данных в СИ++

Соотношение между основными типами можно представить так:

```
1 = sizeof(char) <= sizeof(short) <=«sizeof(int)» <=«sizeof(long)» sizeof(float) <=«sizeof(double)»
```

В итоге предполагать что-то еще относительно основных типов неразумно. Например, то, что целое достаточно для хранения указателя, справедливо не для всех машин. В основном типе можно использовать прилагательное const. Это дает тип, который имеет те же свойства, что и первоначальный тип, за исключением того, что значение переменных типа const не может меняться после инициализации.

```
const float pi = 3.14;
const char plus = '+';
```

Символ, который заключен в одинарные кавычки, является символьной константой. Часто константа, которая определяется таким образом, не занимает память. Там, где необходимо, ее значение может использоваться непосредственно. Константе следует инициализироваться при описании. Для переменных инициализация не всегда обязательна, но очень рекомендуется. Оснований для применения локальной переменной без ее инициализации очень немного.

К каждой комбинации данных типов могут применяться арифметические операции: + (плюс, унарный

23a

23. Операции языка СИ++

С++ обладает богатым набором операций, которые позволяют в выражениях образовывать новые значения и изменять значения переменных. Поток управления в программе определяется с помощью операторов, а описания применяются для введения в программе имен переменных, констант и т. д. Описания служат операторами, поэтому они свободно могут сочетаться с другими операторами.

Необходимо учесть, что операции из таблицы 1 применяются к целым и что не существует отдельного типа данных для логических действий.

Таблица 1

Символ	Обозначение
~	дополнение
&	И
^	исключающее ИЛИ
|	включающее ИЛИ
<<	логический сдвиг влево
>>	логический сдвиг вправо

Операции, применяемые к целым операндам

Смысл операции определяется числом операндов; унарное & — операция взятия адреса, а бинарное & — операция логического И.

Смысл операции определяется также типом ее операндов: + в выражении a + b означает сложение с плавающей точкой, если тип операндов float, но целое сложение, если их тип int.

24a

24. Оператор выражение. Пустой оператор

Самый простой вид оператора — оператор выражение. Он включает в себя выражение, за которым следует точка с запятой.

К примеру:

```
a = b*3 + c;
cout << «go go go»; lseek(fd,0,2);
```

Простейший оператор — пустой оператор:

```
;
```

Он не делает ничего. Но он используется тогда, когда синтаксис требует присутствия оператора, а данный оператор не нужен.

Блок представляет собой возможно пустой список операторов, находящихся в фигурных скобках:

```
{a = b + 2; b++;}
```

Блок дает возможность рассматривать несколько операторов как один. Область видимости имени, которая описана в блоке, простирается до конца блока. Имя может быть невидимым с помощью описаний такого же имени во внутренних блоках.

Программа в следующем примере производит преобразование дюймов в сантиметры и сантиметров в дюймы; предполагается, что указаны единицы измерения вводимых данных, добавляется i для дюймов и с для сантиметров:

22б и бинарный), – (минус, унарный и бинарный), * (умножение), / (деление).

Кроме этого применяются операции сравнения: = (равно), != (не равно), < (меньше), > (больше), <= (меньше или равно), >= (больше или равно).

При присваивании и арифметических операциях C++ осуществляет все осмысленные изменения между основными типами, чтобы их можно было сочетать без ограничений:

```
double d = 1;
int i = 1;
d = d + i;
i = d + i;
```

Рассмотрим операции, которые создают из основных типов новые типы: * (указатель на), *const (константный указатель на), & (ссылка на), [] (вектор), () (функция, возвращающая).

У всех векторов нижней границей индекса является ноль, поэтому в v десять элементов: v[0] ... v[9].

В предметном указателе могут быть адреса объекта соответствующего типа:

```
char c;
// ...
p = &c // p указывает на c
```

Унарное & — операция взятия адреса.

21б ние имени или выражения. Для целых определены такие операции: +, –, * и /. После включения файла stream.h объект типа int способен также быть вторым операндом <<, когда первый операнд ostream.

Тип объекта задает не только применяемые к нему операции, но и смысл этих операций. К примеру, оператор:

```
cout << inch << « in=» << inch*2.54 << « cm\n»;
```

правильно обрабатывает четыре вводных значения разными способами. Строки печатаются буквально, а целое inch и значение с плавающей точкой inch*2.54 изменяются из их внутреннего представления в подходящее для человеческого восприятия символьное представление. В C++ существует несколько основных типов и несколько способов создавать новые.

Основные типы, которые наиболее непосредственно отвечают средствам аппаратного обеспечения, имеют вид: char short int long float double.

Первые четыре типа применяются для представления целых, последние два — для представления чисел с плавающей точкой. Переменная типа char обладает размером, естественным для хранения символа на этой машине (обычно байт), а переменная типа int обладает размерами, соответствующими целой арифметике на этой машине (обычно слово).

Диапазон целых чисел, которые можно представить типом, определяется его размерами. В C++ размеры определяются единицами размера данных типа char, поэтому char по определению обладает единичным размером.

24б

```
#include

main()
{
const float fac = 2.54;
float x, in, cm;
char ch = 0;

cout << «введите длину: «; cin>> x >> ch;

if (ch == ‘i’) { // inch — дюймы
in = x;
cm = x*fac;
}
else if (ch == ‘c’) // cm — сантиметры
in = x/fac;
cm = x;
}
else
in = cm = 0;

cout << in << « in=» << cm << « cm\n»;}
```

23б В C++ существует операция присваивания =, а не оператор при-сваивания, как в некоторых языках. То есть присваивание может употребляться в неожиданном контексте; например, x = sqrt(a = 3*x).

Выражение a = b = c значит присвоение с объекту b, а потом объекту a. Другое свойство операции присваивания — она способна совмещаться с большинством бинарных операций. К примеру, x[i + 3]* = 4 имеет значение x[i + 3] = x[i + 3]*4, за исключением того, что выражение x[i + 3] определяется только один раз. Это дает большую степень эффективности без необходимости употребления оптимизирующего компилятора. К тому же это менее длинно.

Во многих программах на C++ широко используются указатели.

Унарная операция * разыменовывает указатель, т. е. *p представляет собой объект, на который указывает p. Данная операция также именуется косвенной адресацией. Например, если имеется char* p, то *p — это символ, на который указывает p. Часто при работе с указателями применяются операция увеличения ++ и операция уменьшения —.

25a

25. Оператор switch, break. Копирование строки

Оператор switch дает возможность сопоставления значения с множеством констант. Проверки в предыдущем примере можно представить в следующем виде:

```
switch (ch) {
case 'i':
in = x;
cm = x*fac;
break;
case 'c':
in = x/fac;
cm = x;
break;
default:
in = cm = 0;
break;
}
```

Операторы break используются для выхода из оператора switch. Константы в вариантах case могут иметь различные значения, и если проверяемое значение не совпадает ни с одной из констант, принимается вариант default. Программист не обязательно должен предусматривать default.

Покажем копирование строки, когда заданы указатель p на ее первый символ и указатель q на целевую строку. По соглашению строка заканчивается символом с целым значением 0.

26a

26. Описание функций

При использовании операции ++ к целой переменной к ней просто добавляется единица. Первая часть оператора for не обязательно должна быть описанием, она может быть простым оператором. К примеру:

```
for (i=0; i<10; i++) q[i]=»p[i];»
```

тоже по смыслу соответствует предыдущей записи при условии, что i соответствующим образом описано раньше.

Описание представляет собой оператор, который вводит имя в программе. Оно способно и инициализировать объект с этим именем. Выполнение описания значит, что при достижении потоком управления описания вычисляется инициализирующее выражение (инициализатор) и наблюдалась инициализация.

Например:

```
for (int i = 1; i
```

Функция является именованной частью программы, к которой можно обращаться из других частей программы любое количество раз. Покажем программу, печатающую степени числа 2:

```
extern float pow(float, int); //pow()

main()
{
for (int i=0; i<10; i++) cout << pow(2,i) << «\n»; }
```

27a

27. Исходные файлы C++

Программа на C++ чаще всего включает в себя большое число исходных файлов, в каждом из которых находятся описания типов, функций, переменных и констант. Чтобы имя можно было применять в разных исходных файлах для ссылки на определенный объект, оно должно быть описано как внешнее. К примеру:

```
extern double sqrt(double);
extern instream cin;
```

Самый простой способ обеспечить согласованность исходных файлов — помещение таких же описаний в отдельные файлы, которые называются заголовочными (или хэдер) файлами, после чего включить, т. е. скопировать, эти заголовочные файлы во все файлы, где необходимы эти описания. К примеру, если описание sqrt располагается в заголовочном файле для стандартных математических функций math.h и необходимо извлечь квадратный корень из 4, следует применить программу:

```
#include
//...
x = sqrt(4);
```

Так как обычные заголовочные файлы состоят из большого числа исходных файлов, в них нет описаний, которые не должны повторяться.

В команде включения include имя файла, которое располагается в угловых скобках, например, относится к файлу с таким именем в стандартном каталоге (обычно

28a

28. Описание

Имя вводится в программе с помощью описания, которое задает его тип и, возможно, начальную величину. Даны понятия описания, определения, области видимости имен, времени существования объектов и типов.

Перед использованием имени (идентификатора) в C++ программе оно должно быть описано. То есть следует задать его тип, чтобы сообщить компилятору, к какого вида объектам относится имя. Рассмотрим несколько примеров, показывающих разнообразие описаний:

```
char ch;
int count = 1;
char* name = «Bjarne»;
struct complex {float re, im;};
complex cvar;
extern complex sqrt(complex);
extern int error_number;
typedef complex point;
float real(complex* p) {return p->re;};
const double pi = 3.1415926535897932385;
struct user;
```

Большинство описаний служит также определениями; т. е. они к тому же определяют для имени сущность, к которой оно относится. Для ch, count и cvar этой сущностью служит соответствующий объем памяти, который должен применяться как переменная, — эта память будет выделена. Для real это заданная функция, для constant pi это значение 3.1415926535897932385, для com-

26б Первая строка функции — описание, которое указывает, что pow — функция, которая получает параметры типа float и int и возвращает float. Описание функции применяется для выполнения определенных обращений к функции в других местах. При вызове тип любого параметра функции сопоставляется с ожидаемым типом аналогично тому, как если бы инициализировалась переменная приведенного типа. Это дает гарантии надлежащей проверки и преобразования типов. К примеру, обращение pow (12.3, «abcd») вызовет недовольство компилятора, так как «abcd» является строкой, а не int. В случае вызова (2,i) компилятор преобразует 2 к типу float, т. е. требуется функция. Функция pow может быть определена, к примеру, следующим образом:

```
float pow(float x, int n)
{
if (n < 0) error(«извините, отрицательный показатель для pow()»); switch(n) {case 0: return 1; case 1: return x; default: return x*pow(x,n–1);}}
```

25б
```
while (p != 0) {
    *q = *p; // скопировать символ
q = q+1;
p = p+1;
}
*q = 0; // завершающий символ 0 скопирован не был.
```

После while любое условие должно находиться в круглых скобках. Условие вычисляется, и, если оно не нулевое, выполняется непосредственно следующий за ним оператор. Это происходит до тех пор, пока вычисление условия не даст ноль.

Можно применять операцию ++ для непосредственного указания увеличения, и проверка будет значительно проще:

```
while (*p) *q++ = *p++;
*q = 0;
```

где конструкция *p++ значит: «взять символ, на который задает p, затем увеличить p».

28б plex этой сущностью служит новый тип. Для point это тип complex, поэтому point принимает смысл complex. Только описания

```
extern complex sqrt(complex);
extern int error_number;
struct user;
```

не служат одновременно определениями. То есть объект, к которому они относятся, должен быть определен где-то еще. Код (тело) функции sqrt должен определяться определенным описанием, память для переменной error_number типа int должна выделяться другим описанием, и какое-то другое описание типа user должно определять, что он из себя представляет. В C++ программе всегда должно присутствовать только одно определение каждого имени, но описаний может быть большое количество и все описания должны согласовываться с типом объекта, которого они касаются, поэтому в этом фрагменте есть две ошибки:

```
int count;
int count; // ошибка: переопределение
exnern int error_number;
exnern int error_number; //
ошибка: несоответствие типов
```

27б это /usr/include/CC); на файлы, хранящиеся в других местах, ссылаются, применяя имена, расположенные в двойных кавычках. Например:

```
#include «math1.h»
#include «/usr/bs/math2.h»
```

включит math1.h из текущего пользовательского каталога, а math2.h из каталога /usr/bs.

Покажем, как мы могли бы определить тип потока вывода ostream. Для упрощения задачи предположим, что для буферизации определен тип streambuf. Тип streambuf определен в том месте, где также располагается и настоящее определение ostream. Значение типа, определяемого пользователем, специфицирует данные, которые нужны для представления объекта этого типа, и большое число операций для работы с этими объектами. Определение состоит из двух частей: закрытой (private) части, в которой находится информация, используемая только его разработчиком, и открытой (public) части, являющейся интерфейсом типа с пользователем.

29a

29. Описание и скрытие имен

Описание определяет имя в области видимости. Таким образом, имя может применяться только в некоторой части программы. Для имени, которое описано в функции (такое имя часто называют локальным), эта область видимости располагается от точки описания до конца блока, в котором появилось описание. Для имени не в функции и не в классе (называемого часто глобальным именем) область видимости находится от точки описания до конца файла, в котором появилось описание. Описание имени в блоке может прятать описание во внутреннем блоке или глобальное имя, т. е. можно переопределять имя внутри блока с целью ссылки на другой объект. После выхода из блока имя опять получает свое прежнее значение.

Например:

```
int x; // глобальное x

f() {
int x; // локальное x прячет глобальное x
x = 1; // присвоить локальному x
{
int x; // прячет первое локальное x
x = 2; // присвоить второму локальному x
}
x = 3; // присвоить первому локальному x
}

int* p = &x // взять адрес глобального x
```

30a

30. Имена переменных

Имя (идентификатор) включает в себя последовательность букв и цифр. Первый символ должен являться буквой. Знак подчерка _ считается буквой. C++ не ограничивает число символов в имени, но определенные части реализации находятся вне ведения автора компилятора (в частности, загрузчик), и они подобные ограничения налагают.

Приведем примеры последовательностей символов, которые не могут применяться как идентификаторы:

```
012 a fool $sys class 3var
pay.due foo~bar .name if
```

Буквы в верхнем и нижнем регистрах являются различными, т. е. Count и count — разные имена, но вводить имена, почти не отличающиеся друг от друга, нежелательно. Имена, которые начинаются с подчерка, по традиции применяются для специальных средств среды выполнения, поэтому применять такие имена в прикладных программах не стоит.

Каждое имя (идентификатор) в C++ программе обладает ассоциированным с ним типом. Данный тип определяет, какие операции возможно применить к имени, а также как эти операции интерпретируются.

Целый тип char удобнее всего применять для хранения и обработки символов на некотором компьютере; обычно это 8-битовый байт. Размеры объектов C++ выражаются в единицах размера char, т. е. можно записать sizeof(char)==1.

31a

31. Разыменование

Основной операцией над указателем является разыменование, т. е. ссылка на объект, на который указывает указатель. Эту операцию также именуют косвенным обращением. Например:

```
char c1 = 'a';
char* p = &c1 // в p хранится адрес c1
char c2 = *p; // c2 = 'a'
```

Над указателями можно осуществлять определенные арифметические действия. К примеру, функция, подсчитывающая число символов в строке (не считая завершающего 0):

```
int strlen(char* p)
{
int i = 0;
while (*p++) i++;
return i;
}
```

Два структурных типа различны, даже когда они имеют одинаковые члены. К примеру:

```
struct s1 {int a;};
struct s2 {int a;};
```

являются двумя разными типами, поэтому

```
s1 x;
s2 y = x; // ошибка: несоответствие типов.
```

32a

32. Ссылка

Ссылка — это другое имя объекта. Главное применение ссылок заключается в спецификации операций для типов, определяе-мых пользователем. Их можно также применять как параметры функции. Запись x& представляет собой ссылку на x.

К примеру:

```
int i = 1;
int& r = i; // r и i теперь ссылаются на один int
int x = r // x = 1
r = 2; // i = 2;
```

Ссылке следует быть инициализированной.

В большинстве машин можно обращаться к объектам намного быстрее, когда они помещены в регистр. В идеальном случае компилятор сам определяет оптимальную стратегию применения всех возможностей, доступных на машине, для которой компилируется программа. Но это не простая задача, поэтому иногда необходимо дать подсказку компилятору. Это осуществляется с помощью описания объекта как register.

К примеру:

```
register int i;
register point cursor;
register char* p;
```

Описание register применяют только тогда, когда эффективность действительно важна. C++ позволяет записать значения основных типов: символьных кон-

30б Тип unsigned char является беззнаковым, и при его использовании имеем более переносимые программы, но при применении его вместо char могут появиться значительные потери в эффективности.

Тип void (пустой) синтаксически проявляет себя как основной тип. Но применять его следует только как часть производного типа, объектов типа void нет. Он применяется для указания, что функция не возвращает значение, или в качестве базового типа для указателей на объекты неизвестного типа.

```
void f() // f не возвращает значение
void* pv; // указатель на объект неизвестного типа
```

Для большинства типов T T* служит типом указатель на T. То есть в переменной типа T* может располагаться адрес объекта типа T. Для указателей на вектора и указателей на функции необходимо пользоваться более сложной записью:

```
int* pi;
char** cpp; // указатель на указатель на char
int (*vp)[10]; // указатель на вектор из 10 int'ов
int (*fp)(char, char*); // указатель на функцию
// получающую параметры (char, char*)
// и возвращающую int
```

29б Скрытие имен обязательно при написании больших программ. Но читающий человек легко может не заметить, что имя скрыто, и некоторые ошибки, которые возникают вследствие этого, очень тяжело обнаружить, в основном потому, что они редкие. Применение для глобальных переменных имен i или x напрашивается на неприятности. С помощью операции разрешения области видимости :: можно применять скрытое глобальное имя. Например:

```
int x;

f()
{
int x = 1; // скрывает глобальное x
::x = 2; // присваивает глобальному x
}
```

Однако возможности применять скрытое локальное имя нет. Область видимости имени начинается точкой описания. То есть имя можно применять даже для задания его собственного значения. К примеру:

```
int x;

f()
{
int x = x; // извращение
}
```

32б стант, целых констант и констант с плавающей точкой. Также ноль (0) может применяться как константа любого указательного типа, и символьные строки служат константами типа char[]. Можно также определить символические константы. Символическая константа представляет собой имя, значение которого нельзя изменить в его области видимости. В C++ существует три вида символических констант:

1) любому значению любого типа можно присвоить имя и использовать его как константу, добавив к его описанию ключевое слово const;
2) множество целых констант может быть задано как перечисление;
3) любое имя вектора или функции является константой.

При программировании нетривиальных разработок приходит момент, когда необходимо иметь больше пространства памяти, чем имеется или отпущено. Существует два способа получить побольше пространства из того, что доступно:

1) помещение в байт более одного небольшого объекта;
2) применение одного и того же пространства для хранения разных объектов в разное время.

Первое можно осуществить с помощью применения полей, второе — объединений.

31б Структурные типы отличаются и от основных типов, поэтому

```
s1 x;
int i = x; // ошибка: несоответствие типов
```

Но существует механизм описания нового имени для типа, который не требует введения нового типа. Описание с префиксом typedef вводит не новую переменную данного типа, а новое имя этого типа. К примеру:

```
typedef char* Pchar;
Pchar p1, p2;
char* p3 = p1;
```

33а

33. Выражения и операторы

C++ обладает небольшим, но гибким набором различных видов операторов для контроля потока управления в программе и богатым выбором операций для управления данными.

То есть программа представляет собой последовательность строк. Каждая строка включает в себя одно или более выражений, разделенных запятой. Основными элементами выражения служат числа, имена и операции *, /, +, – (унарный и бинарный) и =. Имена не всегда описываются до использования.

Применяемый метод синтаксического анализа обычно именуется рекурсивным спуском; это популярный и простой нисходящий метод. В таком языке, как C++, в котором вызовы функций вполне дешевы, кроме того, данный метод эффективен. Для любого правила вывода грамматики существует функция, вызывающая другие функции. Терминальные символы (например, END, NUMBER, + и –) определяются лексическим анализатором get_token(), а нетерминальные символы определяются функциями синтаксического анализа expr(), term() и prim().

Программа разбора для обнаружения ввода применяет функцию get_token(). Значение вызова get_token() определяется в переменной curr_tok; curr_tok принимает одно из значений перечисления token_value.

В любой функции разбора предполагается, что было обращение к get_token() и в curr_tok располагается очередной символ, подлежащий анализу. Это дает возможность программе разбора заглядывать на один лексический символ вперед и вынуждает функцию разбора

34а

34. Функции и файлы

Все неэлементарные программы включают в себя несколько раздельно компилируемых единиц (их называют просто файлами). Покажем, как раздельно откомпилированные функции могут обращаться друг к другу.

Присутствие всей программы в одном файле обычно невозможно, так как коды стандартных библиотек и операционной системы располагаются где-то в другом месте. При этом хранить весь текст программы в одном файле обычно непрактично и неудобно. Так как единицей компиляции служит файл, то во всех случаях, когда в файле производятся изменения, весь файл необходимо компилировать заново. Даже для небольшой программы время, затрачиваемое на перекомпиляцию, можно заметно сократить с помощью разбиения программы на файлы подходящих размеров.

Покажем пример с калькулятором. Он был представлен одним исходным файлом. Если он набит, то наверняка были трудности с размещением описаний в правильном порядке и необходимо было бы применить по меньшей мере одно «фальшивое» описание, чтобы компилятор обрабатывал взаимно рекурсивные функции expr(), term() и prim(). Программа заключает в себе четыре части (лексический анализатор, программа синтаксического разбора, таблица имен и драйвер), но это никак не было отражено внутри программы. В общем, калькулятор был написан по-другому. Так это не делается; даже если в этой программе «на выброс» пренебречь всеми соображениями методологии программирования, эксплуатации и эффективности компиляции,

35а

35. Классы

Предназначение понятия класса заключается в том, чтобы предоставить инструмент для образования новых типов, таких же удобных в обращении, как и встроенные типы. В идеальном случае новый тип способом применения не должен отличаться от встроенных типов, только способом создания.

Тип является конкретным представлением некоторой концепции. К примеру, включающийся в C++ тип float с его операциями +, –, * и т. д. обеспечивает ограниченную, но конкретную версию математического понятия действительного числа. Новый тип образуется для того, чтобы дать специальное и конкретное описание понятия, которому ничто прямо и очевидно среди встроенных типов не отвечает.

К примеру, в программе, работающей с телефоном, можно было бы создать тип trunk_module (элемент линии), а в программе обработки текстов — тип list_of_paragraphs (список параграфов). Обычно программу, в которой образуются типы, хорошо отвечающие понятиям приложения, понять легче, чем ту, в которой это не происходит. Хорошо выбранные типы, которые определяются авторами программы, делают программу более четкой и короткой. Это также дает возможность компилятору обнаруживать недопустимые применения объектов, которые в противном случае останутся ненайденными до тестирования программы.

В определении нового типа основной идеей является отделить несущественные подробности реализации (формат данных, которые применяются для хранения объекта типа) от качеств, существенных для его пра-

36а

36. Перегрузка операций

Часто программы имеют дело с объектами, которые являются представлениями абстрактных понятий. К примеру, тип данных int в C++ вместе с операциями +, –, *, / и т. д. является реализацией математического понятия целых чисел. Подобные понятия чаще всего включают в себя множество операций, которые кратко, удобно и привычно описывают основные действия над объектами. Язык программирования может непосредственно поддерживать только очень малое количество таких понятий. Например, понятия, комплексная арифметика, матричная алгебра, логические сигналы и строки не имеют прямой поддержки в C++. Классы дают метод спецификации в C++ представления неэлементарных объектов с множеством действий, которые выполняются над данными объектами. Часто определение того, как работают операции на объекты классов, дает возможность обеспечить более удобную запись для манипуляции объектами классов, чем та, которую можно получить, применяя только основную функциональную запись.

К примеру:

```
class complex {
double re, im;
public:
complex(double r, double i) { re=r; im=i; }
friend complex operator+(complex, complex);
friend complex operator*(complex, complex);
};
```

дает возможность просто определить понятие ком-

34б следует разбить эту программу в 200 строк на несколько файлов, чтобы программировать было приятнее.

Программа, которая состоит из нескольких раздельно компилируемых файлов, должна быть согласованной в смысле применения имен и типов, так же, как и программа, которая состоит из одного исходного файла. Вообще это может обеспечить и компоновщик. Компоновщик представляет собой программу, которая стыкует отдельно скомпилированные части вместе. Компоновщик часто именуют загрузчиком. В UNIX'е компоновщик именуется ld. Но компоновщики, которые имеются в большинстве систем, обеспечивают очень слабую поддержку проверки согласованности.

Программист способен скомпенсировать недостаток поддержки со стороны компоновщика, предоставив дополнительную информацию о типах (описания). После этого согласованность программы осуществляется проверкой согласованности описаний, которые располагаются в отдельно компилируемых частях. Средства, которые это осуществляют, обеспечивают, в C++ разработаны так, чтобы способствовать такой явной компоновке.

Если оговорено иное, то имя, которое не является локальным для функции или класса, в любой части программы, компилируемой отдельно, должно относиться к определенному типу, значению, функции или объекту. То есть в программе может существовать только один нелокальный тип, значение, функция или объект с данным именем.

33б читать на одну лексему больше, чем применяется правилом, для обработки которого она была вызвана. Каждая функция разбора определяет «свое» выражение и возвращает значение. Функ-ция expr() обрабатывает сложение и вычитание; она включает в себя простой цикл, который обнаруживает термы для сложения или вычитания.

Сама функция делает мало. В манере, которая типична для функций более высокого уровня в громоздких программах, она вызывает для осуществления работы другие функции.

Обработка ошибок в программах C++ не составляет большого труда. Функция обработки ошибок просто определяет ошибки, пишет сообщение об ошибке и возвращает управление обратно:

Возвращение производится потому, что ошибки чаще всего встречаются в середине вычисления выражения, и поэтому следует или полностью прекращать вычисление, или возвращать значение, которое не должно привести к последующим ошибкам. Для обычного калькулятора больше подходит последнее. Если бы get_token() обнаруживала номера строк, то error() сообщала бы, где приблизительно обнаружена ошибка. Это было бы полезно, если бы калькулятор применялся неинтерактивно.

Когда все части программы разделены, необходим только драйвер для инициализации и того, что связано с запуском. Например:

Принято обычно, что main() возвращает ноль при обычном завершении программы и не ноль в противном случае, поэтому это прекрасно осуществляет возвращение числа ошибок.

36б плексного числа, в котором число является парой чисел с плавающей точкой двойной точности, работа с которыми производится операциями + и *. Человек определяет смысл операций + и * с помощью определения функций с именами operator+ и operator*. Если, к примеру, имеем b и c типа complex, то b+c значит (по определению) operator+(b,c). Теперь существует возможность приблизить простую интерпретацию комплексных выражений. К примеру:

```
void f()
{
complex a = complex(1, 3.1);
complex b = complex(1.2, 2);
complex c = b;

a = b+c;
b = b+c*a;
c = a*b+complex(1,2);
}
```

35б вильного использования. Подобное разделение можно описать так, что работа со структурой данных и внутренними административными подпрограммами производится через специальный интерфейс.

Класс представляет собой определяемый пользователем тип. Данный раздел знакомит с основными средствами определения класса, создания объекта класса, работы с такими объектами и, наконец, уничтожения таких объектов после использования.

Явной связи между функциями и типом данных не существует. Такую связь можно определить, описав функции как члены:

```
struct date {
int month, day, year;

void set(int, int, int);
void get(int*, int*, int*);
void next();
void print();
};
```

Функции, описанные данным способом, называются функциями-членами и могут вызываться только для специальной переменной некоторого типа с применением стандартного синтаксиса для доступа к членам структуры.

37a

37. Производные классы

Производные классы предоставляют простой и эффективный аппарат задания для класса альтернативного интерфейса и установления класса путем добавления возможностей к уже существующему классу без перепрограммирования или перекомпиляции. С помощью производных классов возможно и обеспечить общий интерфейс для определенных классов так, чтобы другие части программы работали с объектами этих классов одинаковым образом. При этом чаще всего в каждый объект закладывается информация о типе, чтобы эти объекты могли обрабатываться в ситуациях, когда их тип невозможно определить во время компиляции. Для простой и надежной обработки таких динамических зависимостей типов вводится понятие виртуальной функции. По сути, производные классы применяются для того, чтобы облегчить формулировку общности.

Рассмотрим процесс написания средства общего назначения, которое будет использоваться в различных обстоятельствах. Ясно, что таких средств множество и выгоды от их стандартизации огромны. В большой программе вполне может быть много копий похожих частей кода для работы с такими фундаментальными понятиями.

Причина подобного хаоса частично заключается в том, что представить такие общие понятия в языке программирования не просто с концептуальной точки зрения. Кроме того, средства, которые обладают достаточной общностью, налагают дополнительные расходы по памяти и/или по времени, в результате чего делает их неудобными для самых простых и наиболее

38a

38. Потоки

Язык C++ не обладает средствами для ввода/вывода. Ему это и не нужно; подобные средства легко и элегантно можно создать, применяя сам язык. Стандартная библиотека потокового ввода/вывода дает возможность осуществлять гибкий и эффективный с гарантией типа метод обработки символьного ввода целых чисел, чисел с плавающей точкой и символьных строк, кроме того, простую модель ее расширения с целью обработки типов, определяемых пользователем.

Разработка и осуществление стандартных средств ввода/вывода для языка программирования является заведомо трудной работой. Традиционно средства ввода/вывода изобретались только для небольшого числа встроенных типов данных. Но в C++ программах обычно применяется много типов, определенных пользователем, и необходимо обрабатывать ввод и вывод также и значений этих типов. Средство ввода/вывода должно являться простым, удобным, надежным в употреблении, эффективным и гибким и при этом полным. Еще не удавалось угодить всем, поэтому у пользователя должна иметься возможность задавать альтернативные средства ввода/вывода и увеличивать стандартные средства ввода/вывода применительно к требованиям приложения.

C++ устроен так, чтобы у пользователя имелась возможность определять новые типы, такие же эффективные и удобные, сколь и встроенные типы.

Средства ввода/вывода связаны только с обработкой преобразования типизированных объектов в последовательности символов и обратно. Существуют и

39a

39. Инспектор объектов для языка Дельфи

В окне инспектора объектов присутствует выпадающее меню и две вкладки. Это меню показывает, конфигурация какого объекта в данный момент представлена, а вкладки, собственно, дают нам возможность данную конфигурацию созерцать и изменять. Во вкладке Свойства (Properties) показано свойство формы проекта.

Рассмотрим некоторые основные свойства подробнее:

Action — определяет для объекта действие;

ActiveControl — показывает на элементы формы, имеющие фокус;

Align — определяет положение объекта:

BiDiMode — данное свойство применяется для локализации приложения. Оно дает возможность установить направление чтения текста в программе;

BorderStyle — свойство, которое отвечает за тип формы. BorderIcons — группа свойств, которые отвечают за видимость системных элементов — кнопок: «свернуть», «закрыть», «развернуть», «помощь» и «системное меню»;

Caption — включает в себя текст названия формы проекта;

CurrentHelpFile — включает в себя имя файла контекстной помощи, применяемого приложением;

HelpFile — определяет имя файла помощи для приложения;

ShowHint — включает или отключает показ всплывающих ярлыков подсказок;

Hint — имеет строку ярлыка подсказки;

Icon — указывает на знак приложения;

40a

40. Окно редактора кода Дельфи

Окно редактора кода. Окно данного редактора имеет три части:

1) панель модулей описания переменных и используемых модулей;
2) панель кода программы;
3) панель найденных ошибок программы (на этапе создания программы данное окно является невидимым, активизируется, если это необходимо, при компиляции программы).

Рассмотрим конкретные проблемы программирования. Покажем раздел Delphi, который называется диалоги. С одной стороны, это достаточно простой раздел. Затруднительным является вопрос о том, как можно совершить операции над файлом, который должен выбрать пользователь. В различных литературных источниках встречаются разные варианты решения и создания дополнительной формы объекта: и попытка заставить пользователя от руки написать адрес файла, и предлагали воспользоваться компонентами Treeview, OutLine и им подобными. Однако существует и более короткий и простой путь для решения проблемы. Достаточно использовать компоненты OpenDialog. Этот компонент легко устанавливается на форму проекта в форме квадрата. В случае запуска программы он невидим. Квадрат только указывает на то, что компонент установлен и дает возможность обращаться к его свойствам с помощью инспектора объектов. Рассмотрим их поподробнее.

После установки компонента на конструкцию проекта диалоговое окно возможно активизировать, приме-

38б другие схемы ввода/вывода, но эта служит основополагающей в системе UNIX, и большая часть видов бинарного ввода/вывода обрабатывается через изучение символа просто как набор бит, при этом его общепринятая связь с алфавитом не воспринимается. Тогда ключевая проблема состоит в задании соответствия между типизированным объектом и принципиально не типизированной строкой.

Обработка как встроенных, так и определенных пользователем типов однородным образом и с гарантией типа осуществляется при помощи одного перегруженного имени функции для набора функций вывода.

К примеру:

```
put(cerr,»x = «); // cerr — поток вывода ошибок
put(cerr,x);
put(cerr,»\n»);
```

Тип параметра устанавливает то, какая из функций put будет вызываться для каждого параметра. Это решение использовалось в нескольких языках. Но ему не хватает лаконичности. Перегрузка операции << значением «поместить в» дает лучшую запись и дает возможность программисту выводить несколько объектов одним оператором. К примеру:

```
cerr << «x=» << x << « \n»;
```

где cerr представляет собой стандартный поток вывода ошибок.

37б напряженно используемых средств, где они были бы полезны. Понятие производного класса в C++ не обеспечивает общего решения всех рассмотренных проблем, но оно определяет способ справляться с довольно небольшим числом важных случаев.

Написание общецелевых средств является сложной задачей, и часто основной акцент в их разработке другой, чем при разработке программ специального назначения. Нет четкой границы между средствами общего и специального назначения, и к методам и языковым средствам можно относиться так, будто они являются более полезными связи с ростом объема и сложности создаваемых программ.

Покажем построение программы, которая имеет дело с людьми, работающими в некоторой компании. Структура данных в такой программе может быть, например:

```
struct employee { // служащий
char* name; // имя
short age; // возраст
short department; // подразделение
int salary; //
employee* next;
// ...
};
```

40б няя функции Execute:Boolean. В этом случае, если пользователь выбрал один или несколько файлов и нажал кнопку OK, функция возвращает True. Свойство элемента FileName включает имя последнего файла из всех выбранных в диалоге. Свойство Files является списком всех файлов, выбранных в диалоге. Свойство InitialDir определяет каталог, к которому диалог обращается в случае открытия. Свойство Title включает в себя заголовок диалогового окна. Необходимо кроме этого отметить свойство Filter. С помощью данного свойства в случае выбора и сохранения файлов удобно производить отбор файлов только с заданными расширениями. Например, при создании фильтра для исполняемых файлов в свойство должна входить следующая строка:

```
'Исполняемые файлы | *.EXE'
```

В одном фильтре можно отбирать файлы и с разными расширениями:

```
'Графические изображения | *.JPG; *.GIF; *.PNG'
```

Фильтр также можно создать в обычном редакторе, связанном со свойством Filter в инспекторе объектов.

39б Cursor — определяет тип курсора, который будет определен при наведении мыши на область объекта;

Visible — определяет такое свойство объекта, как видимость;

Enabled — определяет состояние объекта. В случае значения False объект недоступен;

Font — группа свойств, которые характеризуют отображение текста, применяемого объектом. С помощью данных свойств можно менять цвет, размер, стиль, кодировку и шрифт текста;

Height — определяет высоту объекта в пикселях;

Width — определяет ширину объекта в пикселях;

Left — определяет отступ левого края объекта от левого края формы проекта в пикселях;

Top — определяет отступ верхнего края объекта от верхнего края проекта в пикселях;

Name — внутреннее название объекта программы;

TabOrder — показывает направление движения фокуса в случае нажатия на клавишу Tab;

TabStop — при некотором значении False фокус в случае нажатия на клавишу Tab на данный объект не переводится.

Вышеприведенные свойства присущи многим компонентам. Но важно отметить, что, кроме приведенных параметров, компоненты обладают и специфическими свойствами.

41a

41. Сообщения Дельфи

Осуществить сообщения можно с помощью нескольких процедур: с помощью процедуры ShowMessage, функции MessageDlg, создания дополнительного окна.

Процедура showmessage дает возможность вывести пользователю простое сообщение. При этом образуется дополнительное окно с названием проекта и кнопкой OK. Выглядит все это так:

Синтаксис: ShowMessage(Msg: string);

Пример:
ShowMessage('Все задачи выполнены успешно');

Функция messageDlg дает возможность создавать сложные диалоговые запросы с применением обратной связи. Имеет форму дополнительного окна, дополненного изображением. Кроме того, можно применять диалоговые запросы. Название окна определяется типом запроса.

Синтаксис:

MessageDlg(Msg: string; AType: TMsgDlgType; AButtons: TMsgDlgButtons; HelpCtx: Longint): Word;

Msg — строковой параметр. Определяет выводимое сообщение;

Atype — внутренний параметр функции. Определяет тип сообщения:

1) mtWarning — сообщение о предупреждении;
2) mtError — сообщение об ошибке;
3) mtInformation — информационное сообщение;
4) mtConfirmation — сообщение о подтверждении;

42a

42. Оптимизация по быстродействию в Ассемблер

Приведем некоторые из самых общих процедур этой категории.

1. Замена универсальных инструкций учитывающими конкретную ситуацию, например замена команды умножения на степень двойки на команды сдвига.

Уменьшение числа передач в программе: вследствие преобразования подпрограмм в макрокоманды для прямого включения в исполнимый код; за счет преобразования условных переходов, так, чтобы условие перехода было истинным относительно реже, чем причины для его отсутствия; перемещение условий общего характера к началу разветвленной последовательности переходов; изменение вызовов, сразу после чего происходит возврат в программу, в переходы и т. д.

2. Оптимизация циклов, в том числе сдвиг вычислений неизменяющихся величин за границы циклов, разворачивание циклов и «соединение» отдельных циклов, выполняемых одно и то же количество раз, в единый цикл («сжатие цикла»).

3. Наибольшее применение всех доступных регистров, в результате хранения в них рабочих значений каждый раз, когда это возможно, чтобы уменьшить число обращений к памяти, упаковка большого числа значений или флагов в регистры и устранение лишних продвижений стека (особенно на входах и выходах подпрограмм).

4. Применение специфических для этого процессора инструкций, например, инструкции засылки в стек прямого значения или умножения числа на непосред-

43a

23. Оптимизация по размеру в Ассемблер

Если работоспособность некоторой программы ограничена ее размером, а не скоростью реализации, то следует применить стратегию оптимизации. При этом работать следует над ухищрениями, в точности противоположными тем, что применялись для увеличения быстродействия. Необходимо тщательно изучить свою программу и определить, что является причиной основной проблемы — размер кода или объем данных.

Если производится работа с большими блоками данных, то необходимый эффект может дать их организация в нетривиальные структуры. Однако замена быстрообрабатываемых, но неплотных массивов и таблиц менее громоздкими структурами типа связных списков или упаковка данных с применением битовых полей, вероятно, даст не очень большие преимущества. Обычное уплотнение таблиц и других структур данных и их дальнейшее разуплотнение не всегда полезно из-за того, что часто необходимо разуплотнять все данные только для того, чтобы добраться до некоторого пункта, а программы уплотнения/разуплотнения сами по себе чаще всего занимают большой объем памяти.

Оптимизация программы для уменьшения размера не похожа на оптимизацию для повышения быстродействия.

Во-первых, следует просмотреть весь текст программы и убрать все предложения и процедуры, которые никогда не осуществляются или недоступны ни из какой точки программы (мертвые коды). Если речь идет о большой прикладной программе, много строк можно безболезненно удалить.

44a

44. Достоинства и недостатки оптимизации

Оптимизация кодов для любого языка всегда заставляет идти на компромиссы. Такими компромиссами являются:

1) сокращение используемого объема памяти в результате снижения быстродействия;
2) увеличение быстродействия в результате ухудшения возможностей сопровождения и доступности текста программы для чтения;
3) уменьшение времени деятельности программы в результате увеличения времени ее разработки.

Среди операций, которые приведены ниже, каждая следующая требует больше времени, чем предшествующая. Рассмотрим эти операции: регистр/регистр, операции с памятью, операции с диском и операции взаимодействия с пользователем. Так что не стоит тратить силы на сокращение нескольких машинных циклов в программе, когда скорость ее исполнения ограничена операциями обращения к дисковым файлам. Взамен можно применить сокращение числа таких операций. А после выполнения того, что, в принципе, могло бы быть оптимизацией, следует осуществить тщательную проверку полученных результатов.

Прежде чем рассматривать настройку некоторой программы, следует убедиться, что она правильная и полная, что применяется правильный для решения поставленной задачи подход и что составлен наиболее ясный, наиболее простой, наиболее структурированный код, который только был возможен.

Если программа удовлетворяет всем приведенным критериям, то на самом деле ее объем и скорость вы-

42б ственный операнд, имеющийся в процессорах 80186, 80188, 80286, 80386 и 80486. Также примером могут быть двухсловные строковые инструкции, команды перемножения 32-разрядных чисел и деления 64-разрядного на 32-разрядное число, которые проводятся в процессорах 80386 и 80486. Программа должна, конечно, первоначально определять, с каким типом процессора она работает.

В процессорах 80 x 86, но не 80 x 88, возможно, удастся повысить скорость действия программы на несколько процентов в результате выравнивания расположения данных и меток, на которые осуществляется передача управления, относительно определенных границ.

Процессоры 8088 и 80188 имеют 8-разрядную шину, и для них не имеет значения, на какую границу выровнены данные, поэтому выравнивание можно не применять или установить на границу байта (1 байт, 8 бит); процессоры 8086, 80186 и 80286 обладают 16-разрядной шиной, и им проще действовать с данными, выровненными на границу слова (2 байта, 16 бит); процессор 80386, для которого свойственна 32-разрядная шина, использует выравнивание на границу двойного слова (4 байта, 32 бита); из-за особенностей своей внутренней кэш-памяти процессору 80486, тоже с 32-разрядной шиной, проще работать, если осуществляется выравнивание на границу параграфа (16 байт, 96 бит).

41б 5) mtCustom — сообщение не содержит в углу изображения. Название окна соответствует названию исполняемого файла (аналогично showmessage);

Abuttons — определяет имена кнопок, отображающиеся в диалоговом запросе. Возможны значения: mbYes, mbNo, mbOK, mbCancel, mbHelp, mbAbort, mbRetry, mbIgnore, mbAll;

HelpCtx — включает номер раздела, используемый пользователем, если он вызовет справку, когда диалог активизирован.

Например:

MessageDlg('Продолжить выполнение программы', mtConfirmation, [mbYes, mbNo], 0);

Для определения того, какой вариант ответа выбрал пользователь, можно применить простую проверку. В этом случае исходный код будет иметь следующий вид:

If MessageDlg('Закончить выполнение задачи?', mtConfirmation, [mbYes, mbNo], 0) = mrYes

Then

Begin

MessageDlg('Выполнение задачи закончено.', mtInformation, [mbOk], 0);

Close;

end;

Создание сообщения с применением дополнительного окна достаточно трудно. Данный способ применяется для создания сложных запросов, когда одного только «Да» и «Нет» недостаточно. Например, когда от пользователя необходимо получить код продолжения, в зависимости от которого приложение выполнит те или иные операции.

44б полнения чаще всего будут вполне приемлемыми без каких-либо дальнейших усовершенствований. Но только применение ассемблера само по себе приводит к повышению скорости выполнения программы в 2—3 раза и примерно к такому же уменьшению размера по сравнению с такой же программой на языке высокого уровня. Кроме того, если что-то делает проще чтение программы и ее сопровождение, то обычно при этом увеличивается скорость исполнения. Можно отказаться от «макаронных» кодов со многими ненужными переходами и вызовами подпрограмм, а также предпочтение простых прямолинейных машинных команд похожим сложным.

Кроме того, самой главной заботой должны быть ощущения потенциального пользователя при работе с данной программой: насколько производительной покажется программа ему? Если о полученной программе складывается мнение, как о неуклюжей, то есть вероятность, что она не будет должным образом оценена. Примером является судьба пакета ToolBook.

43б Во-вторых, проанализируйте программу. Необходимо опять собрать все идентичные или функционально сходные последовательности кода в подпрограммы, выбираемые из любой точки программы. Чем более универсальными будут написанные подпрограммы, тем более вероятно, что их код можно применять повторно. Если последовательно придерживаться данного подхода где только возможно, то получится очень компактная программа модульного типа, которая состоит главным образом из вызовов подпрограмм.

45a

45. Отказ от универсальности

Для операции умножения и деления необходимы значительные усилия от почти любого центрального процессора, так как они должны быть осуществлены (аппаратно или программно) через сдвиги и сложения или сдвиги и вычитания соответственно. Традиционные 4-разрядные и 8-разрядные процессоры не имели машинных команд для умножения или деления, так что данные операции приходилось осуществлять при помощи длинных подпрограмм, где явно осуществляются сдвиги и сложения или вычитания. Первые 16-разрядные процессоры, среди которых 8086, 8088 и 68000, действительно дают возможность осуществить операции умножения и деления аппаратными средствами, но соответствующие процедуры были очень медленными: в процессорах 8086 и 8088, например, для деления 32-разрядного числа на 16-разрядное было необходимо около 150 тактов.

Поэтому небольшие хитрости для увеличения скорости или устранения операций умножения и деления были и остаются одними из первых приемов, которые рассматривает каждый программист, который стремится к совершенству. Большинство из данных хитростей относится к категории, которую именуют «отказ от универсальности». Это замена рассчитанных на общий случай команд умножения и деления (или вызов соответствующих подпрограмм) рядом сдвигов и сложений или вычитаний для конкретных операндов.

Рассмотрим одну из простых процедур оптимизации умножения. Для умножения числа на степень двойки его следует просто сдвинуть влево на необхо-

46a

46. Оптимизация переходов и вызовов подпрограмм

Программы, которые изобилуют ветвлениями и переходами во всех направлениях, нежелательны во всех смыслах, а в случае работы с процессорами серий 80 x 86 и 80 x 88 — особенно. Это является напутствием, цель которого — побудить программистов на ассемблере и тех, кто оптимизирует компиляторы, должным образом структурировать программы. В этом случае существуют свои проблемы, но сначала рассмотрим некоторые особенности процессоров фирмы Intel.

Быстродействие данных процессоров в значительной мере зависит от их архитектуры, основанной на простой конвейерной схеме, которая содержит три компоненты: шинный интерфейс (BIU — Bus Interface Unit), очередь упреждающей выборки и исполнительный модуль (EU — Execution Unit). Если шина памяти в нерабочем состоянии, например в случае выполнения команды из многих циклов, с операндами, находящимися в регистрах, шинный интерфейс получает байты команд из памяти и располагает их в очередь упреждающей выборки, последовательно продвигаясь дальше от текущего расположения командного счетчика центрального процессора. Когда исполнительный модуль заканчивает выполнение очередной команды, он ищет следующую команду в ряде упреждающей выборки: если она есть, к ее расшифровке можно приступать непосредственно, не обращаясь лишний раз к памяти.

Каждый раз, когда исполнительный модуль уточняет команду перехода или вызова, он аннулирует теку-

47a

47. Оптимизация циклов

Существует большое число методов оптимизации циклов с самыми экзотическими названиями: «разгрузка циклов», «вывод инвариантов за циклы», «устранение индуктивных переменных», «сращивание циклов», «разматывание циклов» и т. д. В действительности все эти методы можно объединить в два эмпирических правила.

1. Никогда не следует делать в цикле ничего такого, что можно сделать вне его.

2. Где это можно, следует избавиться от передач управления внутри циклов.

Первое правило следует из истины, по которой 90% времени исполнения программы приходится на 10% ее кода. Эти 10% чаще всего оказываются циклами того или иного рода. Таким образом, первое, что необходимо сделать для ускорения выполнения программы, — это определить в ней «горячие точки» и проверить все циклы в них в качестве потенциальных объектов оптимизации. Цикл далеко не всегда представляет собой изящную конструкцию, которая завершается командами LOOP, LOOPZ или LOOPNZ; часто это просто серия команд, выполнение которых повторяется в зависимости от величины некоторой управляющей переменной или флажка.

Циклы можно разделить на два вида: к первому относятся циклы со временем исполнения, которое определяется некоторыми внешними механизмами синхронизации, ко второму — те, в которых участвует только процессор. Примером первого вида цикла служит такой, в котором набор символов передается на параллельный порт. Скорость выполнения программы никог-

48a

48. Управляющие таблицы

Очень часто целесообразно перенести вычисления из цикла за его пределы и отсрочить вычисления, пока их результаты реально не потребуются. Еще более эффективный вариант оптимизации заключается в том, чтобы приурочить вычисления не ко времени выполнения программы, а к моменту ее компиляции или ассемблирования либо выполнять вычисления, применяя специализированные программы, сохранять результаты в промежуточном файле и вытаскивать их оттуда при необходимости.

Особенно удобно применять оптимизацию просмотром управляющих таблиц.

Покажем прикладную систему, в которой удобнее всего применять управляющие таблицы. Это программа, в которой необходимо поворачивать и перемещать отрезки линий, чтобы создавать у пользователя иллюзию объемного изображения. Подобная программа должна определять синусы и косинусы углов. Для вычисления данных функций обычно используют числа с плавающей точкой и разложение в ряды, расчет которых влечет за собой множественные умножения и деления, а эти операции по времени счета «дорогостоящие». При этом получаемые величины обладают значительно большей точностью, чем это реально необходимо для обычных графических адаптеров персональных компьютеров: даже цифры с плавающей точкой одинарной точности (32 разряда) вычисляются до 8 десятичных знаков, из которых необходимы только 4 или 5. В подобной задаче и можно пользоваться преимуществами таблицы, в которую

щее содержимое очереди упреждающей выборки и определяет новый счетчик команд. Затем шинный интерфейс снова выбирает байты команд, начиная при этом с нового адреса, и заносит их в очередь. Исполнительный модуль в это время должен «простаивать», пока не будет определена полная команда. При этом все обращения к памяти, необходимые для исполнения команды перехода по новому адресу, тоже влияют на выборку следующих команд из памяти. Может пройти много времени, прежде чем шина опять заполнит очередь упреждающей выборки, так, чтобы применяемый модуль мог работать с наибольшей скоростью. Кроме того, размер очереди командных байтов не одинаков для разных моделей центральных процессоров. Он составляет только 4 байта в ранних моделях и 32 байта в современных компьютерах. Таким образом, крайне сложно предсказать время исполнения для данных последовательностей команд исходя из количества тактов и длин в байтах. Также состояние очереди команд для разных типов центральных процессоров определяется «выравниванием» команд. Шинный интерфейс обязан выбирать команды по разрядности адресной и информационной частей шины.

Исходя из всего вышесказанного, можно сформулировать первое правило оптимизации переходов и вызовов: необходимо проверить, что их точки назначения попадают в подходящие границы адресов для того типа процессора, на котором данная программа будет работать чаще всего. При этом следует добавить подходящий атрибут выравнивания (WORD или DWORD) в объявления сегментов, а также вставить директиву ALIGN перед каждой меткой.

димое число двоичных (битовых) позиций. Вот такой, например, имеет вид некоторая общая, но медленная последовательность команд для умножения значения переменной MyVar на 8:

```
mov ax,MyVar
mov bx,8
mul bx
mov MyVar,ax
```

Применение отказа от универсальности при выполнении деления несколько более ограничено. Деление на степень двойки, безусловно, очень просто. Для этого следует сдвинуть число вправо, следя только за выбором соответствующей команды сдвига для желаемого типа деления (со знаком или без знака). Определение остатка от деления на степень двойки для чисел без знака еще проще. Для этого следует осуществить просто одну команду операции логического И над операндом и непосредственным значением, которое должно быть записано в виде уменьшенного на единицу значения делителя. Деление чисел со знаком не так просто, так как знак остатка от деления должен соответствовать знаку делителя и не зависит от знака делимого. Реализация данных операций потребует непременного присутствия условных переходов, а это уже плохо.

можно занести синусы углов с шагом в 1 градус и с точностью до 4 десятичных знаков.

Иногда управляющие таблицы вполне эффективно применяются в самых неожиданных ситуациях. Например, необходимо составить подпрограмму, которая будет рассчитывать число ненулевых разрядов в байте. Можно составить цикл со сдвигами и действительно сосчитать ненулевые разряды. Однако намного быстрее будет применить таблицу, позиции в которой будут соответствовать значениям байта — от 0 до 255, а значения в данных позициях — числу ненулевых разрядов для каждого из подобных значений байта.

При этом для повышения быстродействия можно оформить данную подпрограмму как макроопределение и встраивать в программу везде, где необходимо. Для байтовых таблиц можно также повысить производительность с помощью замещения команды MOV на специальные команды XLAT. При этом можно будет обрабатывать не только байтовые таблицы.

да не будет выше темпа приема байтов параллельным портом, а быстродействие данного порта на два порядка ниже, чем время выполнения любой приемлемой кодовой последовательности управления портом. Оптимизация подобных циклов с внешней синхронизацией не часто применяется. Циклы второй категории — свободные от взаимодействия с «внешним миром».

Для полной оптимизации циклов нужен методический подход к проблеме. Сначала следует тщательно проверить все циклы для отыскания операций, которые абсолютно не связаны с переменной цикла, и разгрузить цикл от этих вычислений. Следует проанализировать оставшиеся коды и по возможности упростить их, используя просмотр управляющих таблиц, которые ориентированы на определенную модель процессора команды, отказ от универсальности и любые другие известные приемы, позволяющие уменьшить кодовые последовательности и убрать «дорогостоящие» команды.

Если в некоторых вычислениях применяется текущее значение переменной цикла, следует вывернуть ситуацию наизнанку, определяя нужные величины из начального и конечного значений переменной цикла, т. е. без перебора.

После оптимизации содержимого цикла, насколько это возможно, необходимо посмотреть, можно ли где-то убрать управляющие циклы операций перехода или вызова подпрограмм.

49a

49. Оптимизация для конкретных моделей процессоров

Если некоторая программа будет работать на компьютерах со строго определенными моделями процессоров, можно попытаться применить ориентированные на определенные модели процессоров команды.

Многие из новых команд дают возможность повысить производительность программы.

1. Линейные и циклические сдвиги с аргументом, не равным единице.

2. Команда PUSH с непосредственным операндом.

3. Команды ввода и вывода символьных строк.

4. Команды обмена со стеком тем, что содержится во всех регистрах PUSHA и POPA.

5. Команды ENTER и LEAVE для выделения и освобождения кадра стека.

6. Команды контроля соблюдения границ массива BOUND.

7. Команды умножения числа на непосредственный операнд.

Можно увеличить производительность на несколько процентов за счет малого объема памяти.

При составлении программ для процессоров 80386 и 80486 и их разновидностей можно повысить производительность 16-разрядной программы, используя все вышеупомянутые команды для процессоров 80188, 80188 и 80286 и при этом выравнивая данные и адреса передачи управления по границам 32-разрядных слов, применить следующие дополнительные особенности.

1. 32-разрядные регистры (но применять их следует с осторожностью, так как их содержимое не сохра-

50a

50. Органы управления (controls) Active X

Орган управления является небольшой программой, которой броузер выделяет на странице некоторый участок прямоугольной формы. Внутри своего участка орган управления несет ответственность за перерисовку экрана и взаимодействие с пользователем. К примеру, орган управления способен реализовать что-нибудь вроде движка прокрутки или выпадающего меню, которых сам HTML построить не может; другие модули способны принимать от пользователя, обрабатывать и выводить данные, создавая динамически меняющиеся диаграммы или даже ведя некоторую электронную таблицу прямо в окне броузера; наконец, органы управления еще одного типа выполняют чисто оформительские функции — например, покрывают выделенный им участок узором, плавным переходом цветов, перемещающимся текстом или изображением.

В отличие от модулей Netscape Navigator органы управления ActiveX обладают более узкой и дробной специализацией, меньшими размерами передаваемых по сети файлов, а также полностью автоматизированной установкой. Наткнувшись в HTML на ссылку на определенный орган управления, броузер проверяет, нет ли его на компьютере пользователя (т. е. не применялся ли он раньше). Если орган управления был найден, броузер его запускает, передает необходимые для работы данные и тем самым избавляется от лишнего выхода в сеть. При отсутствии в компьютере этого компонента броузер обращается к серверу, адрес которого указан в HTML-документе и, взяв с него файл, устанавливает и регистрирует новый орган управления в Windows. В результате органом управления может пользоваться не только броузер, но и каждое приложение, которое может работать с так называемыми «нестандартными органами управления OLE» («OLE Custom Controls», OCX).

Самые важные достоинства и недостатки органов управления ActiveX ярче всего демонстрирует сравне-

51a

51. Синтаксис Active X

Для органов управления ActiveX попускать кавычки нельзя.

CODEBASE = URL

В данном атрибуте пишется URL-адрес файла, который содержит вызываемый орган управления и доступный для получения с одного из серверов Интернета.

CODETYPE = MIME-тип, TYPE = MIME-тип

Данные два необязательных атрибута дают возможность указать типы (в терминах стандарта MIME) файлов, к которым обращаются атрибуты CLASSID (атрибут CODETYPE) и DATA (атрибут TYPE).

DATA = URL

С помощью такого атрибута определяется местонахождение файла данных, необходимых для работы данному органу управления.

DECLARE

Добавление такого пустого атрибута вынудит тег <OBJECT> произвести только объявление, а не образование объекта. При этом в память броузера заносится весь набор атрибутов и параметров объекта, но перекачки файлов или запуска программ не наблюдается.

ID = идентификатор

Атрибут ID дает возможность приписать создаваемому объекту имя в форме какого-то идентификатора. Обращение к данному объекту от других объектов или сценариев возможно только через указание данного имени.

NAME = идентификатор

Этот необязательный атрибут. Создаваемый объект может внести свой вклад в данные, которые броузер

52a

52. Практикум Active X

Рассмотрим, как применяются теоретические сведения на практике. Попробуем написать небольшой HTML-файл, вызывающий один из органов управления ActiveX, которые разработала фирма Microsoft, — модуль для образования плавного перехода цветов (градиента). Рассмотрев документацию на данный компонент, можно узнать соответствующий ему идентификатор CLSID и URL-адрес одной его копии на сервере Microsoft, на которую возможно будет сослаться. При этом в том же месте можно отыскать список параметров и их значений, способный принимать этот орган управления, в частности:

StartColor и **EndColor**

Два цвета, плавный переход между которыми можно увидеть на экране, задаются в простом для HTML виде «#rrggbb», где rr, gg и bb — шестнадцатеричная величина красной, зеленой и синей составляющих цвета.

Direction

Направление градиента: 0 — горизонтальное, 1 — вертикальное, 2 — радиальное от центра к краям и т. д.

Теперь необходимо заполнить атрибуты тега <OBJECT> и учесть нужное количество тегов <PARAM>. Такой вид имеет текст рассматриваемого HTML-файла:

```
<HTML>
<TITLE>Пример вызова органа управления ActiveX</TITLE>
<BODY>
```

Данный градиент на вид не отличен от простого графического файла:

50б ние их с Java-апплетами. Первым явным недостатком всей системы ActiveX является ее жесткая привязка к определенной операционной системе (Windows 95/NT). Так как значительную часть поддержки ActiveX и взаимодействия компонентов OLE осуществляет сама операционная система, то перенести все это, например, на UNIX практически невозможно.

Еще один серьезный недостаток — безопасность. Файл, расширение которого .ocx с компонентом системы ActiveX, получает управление почти так же, как и любой другой применяемый файл в Windows, и имеет те же права — к примеру, право бесконтрольной записи на диск (под Windows NT такие права могут быть ограничены уровнем привилегий пользователя, который запускает этот компонент). Здесь открываются большие перспективы для деятельности авторов вирусов и других вредных программ, для которых ActiveX может стать вполне комфортной питательной средой.

Система ActiveX и использованный в ней механизм безопасности кое в чем намного удобнее использования языка Java.

Во-первых, отсутствие защитной оболочки виртуальной машины дает возможность расширить функциональность компонентов ActiveX — они имеют прямой и эффективный контроль над компьютером. Однако основной плюс технологии ActiveX, из-за которого она так резво стартовала и так быстро завоевала свой круг потребителей, — это то, что программистам, которые желают заняться разработкой компонентов ActiveX, почти не приходится переучиваться. OCX-модули появились одновременно с версией 4.0 языка Visual Basic и очень многое взяли от еще более старого стандарта VBX («Visual Basic Controls»). Органы управления VBX в свое время и стали причиной развития целой индустрии программных модулей, которые каждый программист может купить и применять в своих разработках. Переделать же OCX-модуль в компонент ActiveX даже легче, чем старый VBX в OCX.

52б
```
<OBJECT
ID = «grad1»
CLASSID = «clsid:017C99A0-8637-11CF-A3A9-00A0C9034920»
CODEBASE = «http://activex.microsoft.com/controls/iexplorer/iegrad.ocx#Version = 4,70,0,1161»
WIDTH = 200
HEIGHT = 100
>
<PARAM NAME = «StartColor» VALUE = «#ffffff»>
<PARAM NAME = «EndColor» VALUE = «#000000»>
<PARAM NAME = «Direction» VALUE = «0»>
</OBJECT>
</BODY>
</HTML>
```

Открытие такого файла в броузере Internet Explorer станет причиной довольно заметной паузы, во время которой в строке состояния появится надпись «Installing components...». При этом броузер связывается с сервером, который упомянут в атрибуте CODEBASE, и перекачивает с него файл, в котором находится компонент ActiveX (перед этим нужно подключиться к сети).

49б няется, если работают некоторые эмуляторы системы DOS, например модуль совместимости с DOS системы OS/2 версий до 1.3).

2. Команды пересылки с распространением нуля или знакового бита (MOVZX или MOVSX).

3. 64-разрядные сдвиги (в сдвоенных регистрах) — команды SHLD и SHRD.

4. Установка в байте параметров «истина» или «ложь» по содержимому флажков центрального процессора, что дает возможность избавиться от команд условного перехода (SETZ, SETC и т. д.).

5. Команды проверки, установки, сброса, инвертирования и просмотра битов (BT, BTC, BTR, BTS, BSF и BSR).

6. Обобщенная индексная адресация и режимы адресации с масштабированием индексов.

7. Быстрое умножение с помощью команды LEA с применением масштабированной индексной адресации.

8. «Дальние» условных переходов.

9. Перемножение 32-разрядных чисел и деление 64-разрядных чисел на 32-разрядные.

10. Дополнительные сегментные регистры (FS и GS).

11. Команды загрузки сегментных регистров SS, FS и GS (LSS, LFS и LGS).

51б отправит на сервер после заполнения пользователем HTML-бланка.

SHAPES

Такой пустой атрибут заставляет броузер наложить на прямоугольник, который занимает объект, карту (map), так, чтобы отдельные части этого прямоугольника являлись рабочими частями гипертекстовых ссылок. Координаты этих частей и URL-адреса ссылок для них записываются с помощью тегов <A> со специальными дополнительными атрибутами, которые должны находиться между <OBJECT> и соответствующим ему </OBJECT>.

STANDBY = текст

В данном атрибуте можно приводить текстовую надпись, которая будет находиться в прямоугольнике, отведенном объекту, пока сам объект загружается и запускается.

Переменные свойства объекта всегда одинаковы (по значению и по обозначению) с теми параметрами, которые предоставляются ему с помощью тегов <PARAM>. Методами объекта являются функции, вызов которых дает возможность выполнить определенные действия, специфические для этого класса объектов.

53a

53. Сценарии и документы

Двуязычие броузера Internet Explorer делает осмысленным или необходимым применение атрибута LANGUAGE тега <SCRIPT> для указания языка сценария. При этом, поддержка двух языков вызвала введение дополнительного необязательного аргумента в тех функциях, одним из аргументов которых служит строка кода. Например, функция setTimeout (), которая имеет в Netscape Navigator два аргумента, теперь может использовать третий аргумент, который сообщает данной функции, как интерпретировать строку кода в одном из аргументов — как программу на JavaScript или на VBScript.

Необходимо помнить, что хотя вариант языка JavaScript, применяемый Internet Explorer, и называется «JScript» во всех официальных документах Microsoft, на синтаксис языка это никак не повлияло — во всех местах, где необходимо указать один из двух языков, допустимыми вариантами служат либо «JavaScript», либо «VBScript».

В целом JScript является довольно сильным облегченным вариантом JavaScript. Особенно если сравнивать его с тем JavaScript, который поддерживает Netscape Navigator 3.0. Многих методов, событий, объектов нет вообще, некоторые копировались из Netscape Navigator в большой спешке, и даже в синтаксисе имеются некоторые значительные упрощения — в частности, числовые значения больше не объекты. Очевидно, много сценариев для Netscape Navigator вызовут переделки, прежде чем их сможет осуществить Internet Explorer. Но в настоящий момент Micro-

54a

54. Netscape Navigator

По мнению пользователей, данный модуль работает очень прилично и иногда даже превосходит в скорости работы органов управления ActiveX сам Internet Explorer. Но надежность его вызывает нарекания. Если очень долго и интенсивно работать со страницами, которые насыщены компонентами ActiveX, то в итоге можно встретить ошибку, которая приводит к зависанию Netscape.

Но с этим вполне можно было бы мириться, если бы не некоторая техническая трудность. Netscape Navigator не различает тег <OBJECT>, и потому, даже если подключен модуль ScriptActive, это не вызывает органы управления, вводимые этим тегом. Вызов подключаемых модулей Netscape осуществляется с помощью тега <EMBED>. Это означает, что авторам, которые стремятся, чтобы органы управления на их страницах работали в обоих броузерах, следует продублировать информацию тега <OBJECT> в теге <EMBED>. А для того, чтобы заставить Internet Explorer не обращать внимания на этот не нужный ему <EMBED> (как известно, Internet Explorer поддерживает этот тег и даже может работать с подключаемыми модулями Netscape), этот тег располагают внутри соответствующей пары тегов <OBJECT> ... </OBJECT>.

Подобный выход из положения используется в HTML довольно часто — применяя какой-нибудь новый тег, автор в целях совместимости располагает между этим тегом и парным ему закрывающим что-нибудь, что будет видеть только старый броузер, игнорирующий новый тег. Новый броузер, который распоз-

55a

55. Понятие системы VBA

VBA представляет собой подмножество VB и включает средства образования приложений VB, его структуры данных и управляющие структуры, возможность создания пользовательских типов данных. Также как и VB, VBA является системой визуального программирования, управляемого событиями. В нем имеется возможность создания форм со стандартным набором элементов управления и написания процедур, обрабатывающих события, которые возникают при тех или иных действиях системы и конечного пользователя. Также он позволяет использовать элементы ActiveX и автоматизации. VBA представляет собой полноценную систему программирования, но не имеет полный набор возможностей, которыми обладает последняя версия VB.

Программирование в среде VBA обладает рядом особенностей. В частности, в ней нельзя создавать проект независимо от этих приложений.

Из-за того что VBA является визуальной системой, программист способен создавать видимую часть приложения, которая является основой интерфейса «программа — пользователь». Благодаря ему производится взаимодействие пользователя с программой. На принципах объектно-ориентированного подхода, который реализуется в VBA применительно к приложениям, выполняемым под управлением Windows, разрабатывается программный интерфейс.

Характерным для данных приложений является то, что на экране в любой момент присутствует множество объектов (окон, кнопок, меню, текстовых и диалоговых окон, линеек прокрутки). С учетом алгоритма программы пользователь обладает определенной свободой выбора относительно использования этих объектов. То есть он может сделать щелчок по кнопке, перенести

56a

56. Язык программирования VBA

Алфавит и лексемы языка

Язык программирования VBA предназначен для написания кода программы. Он обладает своим алфавитом, который включает:

1) строчные и прописные буквы латинского алфавита (A, B,..., Z, a, b,..., z);
2) строчные и прописные буквы кириллицы (А — Я, а — я);
3) неотображаемые символы, используемые для отделения лексем друг от друга;
4) специальные символы, участвующие в построении конструкций языка:
+ -*/\ ^ = ><[](). :{}'&©;
5) цифры от 0 до 9;
6) символ подчеркивания «_»;
7) составные символы, воспринимаемые как один символ: <= >= <>.

Программный код VBA является последовательностью лексических единиц (лексем), которые записаны в соответствии с принятыми синтаксическими правилами, которая реализует нужную семантическую конструкцию.

Лексема является единицей текста программы, имеющей определенный смысл для компилятора, и которая не может быть разбита в дальнейшем.

Идентификатор представляет собой последовательность букв, цифр и символов подчеркивания.

Объявление переменных. Переменные являются объектами, которые предназначены для хранения данных. Перед применением переменных в программе необходимо их объявлять (декларировать). Правиль-

54б нает этот тег, наоборот, должен игнорировать все, что расположено внутри парного тега. Так выглядит тег <OBJECT>, информация которого по-вторена во вложенном теге <EMBED>:

```
<OBJECT
WIDTH = 320
HEIGHT = 240
CLASSID = «clsid:0D5C3F21-6DF8-11CF-AAEB-
02608C9EA5BF»
CODEBASE = «http://www.ncompasslabs.com/Acti-
veX/
                ocx/nbillbrd.ocx»
DATA = «http://www.ncompasslabs.com/ActiveX/inli-
ne/
                billboard.ods»
>
<PARAM NAME = «Slideshow» VALUE = «1»>
<PARAM NAME = «LocalButtons» VALUE = «0»>
<PARAM NAME = «Delay» VALUE = «1»>
<EMBED
WIDTH = 320
HEIGHT = 240
SRC = «BillBoard.ods»
CODE = «http://www.ncompasslabs.com/ActiveX/
            ocx/nbillbrd.ocx»
Slideshow = 1  <!- параметры стали атрибутами ->
LocalButtons = 0
Delay = 1
>
</OBJECT>
```

53б soft даже не подготовила официальной документации на свой вариант JavaScript; все, что можно найти на сервере Microsoft, — это документ, называемый «Microsoft Internet Explorer Scripting Object Model», содержащий перечисление поддерживаемых JScript объектов и их свойств и методов.

Рассмотрим, что такое «ActiveX Documents». «Документы ActiveX» дают Интернету то, к чему для рядового пользователя Windows и сводится технология OLE. К примеру, когда в окне Microsoft Word возникает обычная таблица Excel, с которой можно делать все то же самое, что и в самом Excel. Так же HTML-документ способен теперь включать в себя документы любого другого формата, для которых существуют программы просмотра, которые удовлетворяют стандартам OLE.

Для того чтобы человек мог, не покидая своего броузера, познакомиться с содержимым документа в определенном формате X, должны выполняться два условия. Первое — для данного формата должна иметься программа просмотра, которая способна играть роль OLE-сервера. Второе — такая программа должна присутствовать на компьютере пользователя. Ясно, что оба эти условия значительно ограничивают практическое применение этой идеи. Даже несмотря на то, что для любых приложений, которые входят в Microsoft Office, существуют бесплатно распространяемые программы просмотра (Word Viewer, Excel Viewer и т. п.), рассчитывать на их наличие на каждом компьютере не способна даже корпорация Microsoft, не говоря уже о других, не известных фирмах, форматах и программах. Ясно, что о переносе на другую операционную систему здесь не может быть и речи.

56б ный выбор типа переменной обеспечивает эффективное использование памяти компьютера.

Объекты, значения которых не изменяются и не могут быть изменены во время выполнения программы, называются константами: именованными и неименованными.

Перечни используются для декларации группы констант, которые объединяются общим именем, к тому же они могут быть объявлены только в разделе глобальных объявлений модуля или формы.

Декларация массивов. Выделяют два вида переменных — простые переменные и переменные структурного вида. Массивы бывают одномерными и многомерными.

Операция присваивания. После декларации значение переменной может оказаться произвольным, а для присвоения переменной необходимого значения применяется операция присваивания.

Математические операции используются для записи формулы, представляющей собой программный оператор, который содержит числа, переменные, операторы и ключевые слова.

Операции отношения могут привести к появлению значения, причем существует только два результирующих значения: истина и ложно.

Логические операции используются в логических выражениях, это происходит при существовании нескольких условий выбора в операциях отношения.

55б объект, ввести данные в окно и т. п. При создании программы программист не должен ограничивать действия пользователя, он должен разрабатывать программу, правильно реагирующую на любое действие пользователя, даже неправильное.

Для любого объекта определяется ряд возможных событий. Одни из событий происходят от действий пользователя (щелчок или двойной щелчок мыши, перенос объекта, нажатие клавиши клавиатуры и т. п.). Случается, что некоторые события происходят в результате свершения других событий: окно открывается или закрывается, элемент управления становится активным или теряет активность.

Любое из событий проявляется в определенных действиях программы, а виды возможных действий можно разделить на две группы.

1. Действия первой группы, которые являются следствием свойств объекта, устанавливающихся из некоторого стандартного перечня свойств, которые задаются системой программирования VBA и самой системой Windows. К примеру, свертывание окна после щелчка по кнопке Свернуть.

2. Вторую группу действий на события может определить только программист. Для любого возможного события отклик обеспечивается созданием процедуры VBA. Теоретически возможно создать процедуру для каждого события, но практически программист заполняет кодом процедуры только для событий, представляющих в данной программе интерес.

Свойствами-участниками являются свойства, которые задают вложенные объекты.

Объекты способны реагировать на события — инициируемые пользователем и генерируемые системой. События, инициируемые пользователем, появляются, например, при нажатии клавиши, щелчке кнопками мыши.