Тема 1.3 Отладка и тестирование программного продукта на уровне модулей
Тема 1.3.2 Трансляторы

План:

1. Основные принципы построения трансляторов.
2. Генерация и оптимизация кода.
3. Современные системы программирования.

1 Основные принципы построения трансляторов

Транслятор — это программа, которая переводит входную программу на исходном (входном) языке в эквивалентную ей выходную программу на результирующем (выходном) языке.

В работе транслятора ***участвуют*** всегда три программы.

Во-первых, сам *транслятор является программой* — обычно он входит в состав системного программного обеспечения вычислительной системы. Все составные части транслятора представляют собой фрагменты или модули программы со своими входными и выходными данными.

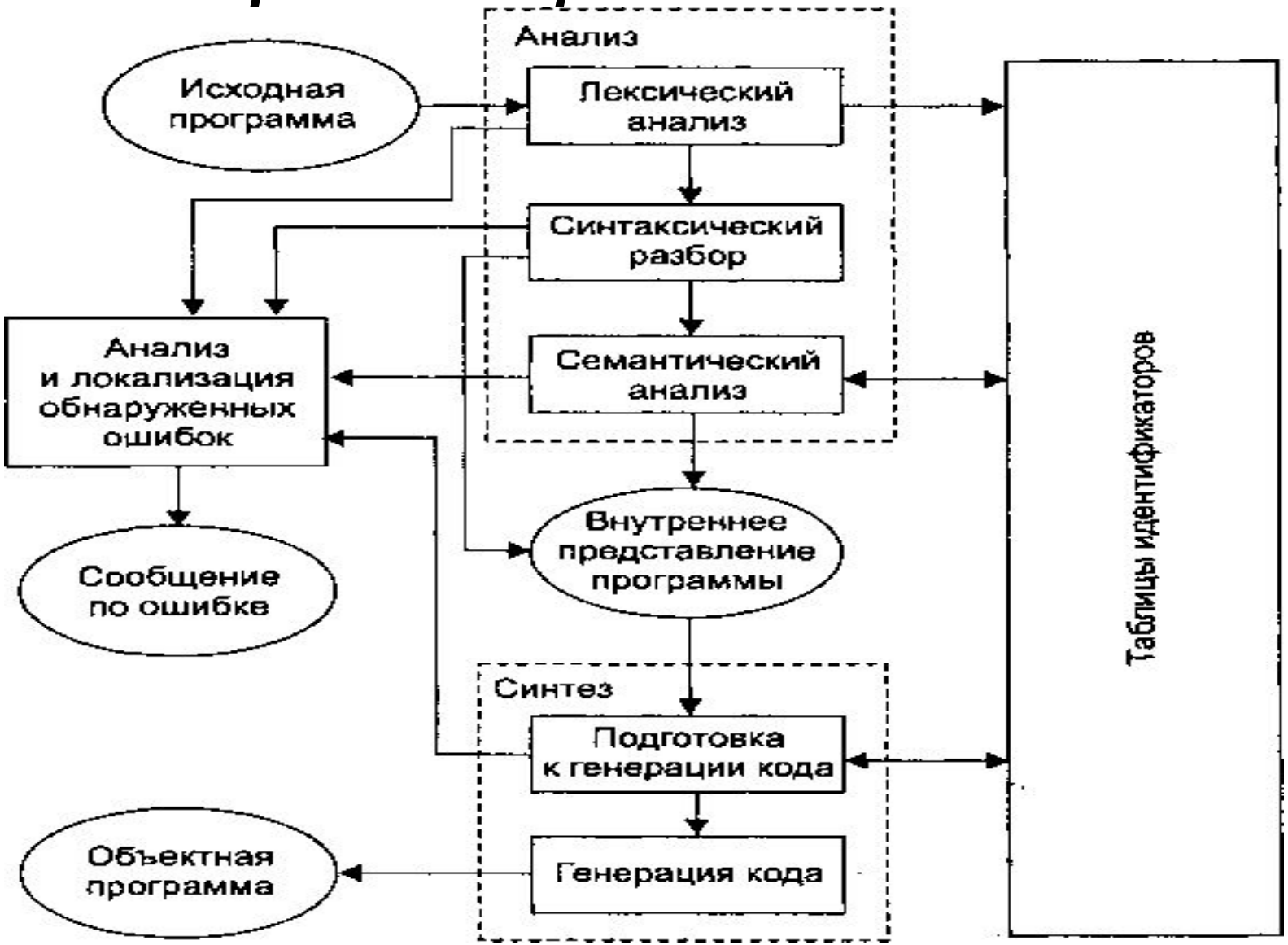
Во-вторых, *исходными данными для работы транслятора служит текст входной программы* — некоторая последовательность предложений входного языка программирования. Обычно это символьный файл, но этот файл должен содержать текст программы, удовлетворяющий синтаксическим и семантическим требованиям входного языка.

В-третьих, *выходными данными транслятора является текст результирующей программы*. Результирующая программа строится по синтаксическим правилам, заданным в выходном языке транслятора, а ее смысл определяется семантикой выходного языка.

Компилятор **—** это транслятор, который осуществляет перевод исходной программы в эквивалентную ей объектную программу на языке машинных команд или на языке ассемблера.

Интерпретатор — это программа, которая воспринимает входную программу на исходном языке и выполняет ее. В отличие от трансляторов интерпретаторы не порождают результирующую программу — и в этом принципиальная разница между ними. Интерпретатор анализирует текст исходной программы, преобразует ее в язык машинных кодов.

Этапы трансляции. Общая схема работы транслятора



Компилятор выполняет две основные функции:

1) является распознавателем для языка исходной программы. Генератором цепочек входного языка выступает пользователь — автор входной программы.

2) компилятор является генератором для языка результирующей программы. Распознавателем этой цепочки выступает вычислительная система, под которую создается результирующая программа.

Синтаксический разбор — это основная часть компилятора на этапе анализа. Она выполняет выделение синтаксических конструкций в тексте исходной программы, обработанном лексическим анализатором. На этой же фазе компиляции проверяется синтаксическая правильность программы.

Семантический анализ — это часть компилятора, проверяющая правильность текста исходной программы с точки зрения семантики входного языка. Кроме непосредственно проверки, семантический анализ должен выполнять преобразования текста, требуемые семантикой входного языка.

- ***Генерация кода*** — это фаза, непосредственно связанная с порождением команд, составляющих предложения выходного языка и в целом текст результирующей программы. Это основная фаза на этапе синтеза результирующей программы.
- ***Таблицы идентификаторов*** («таблицы символов») — это специальным образом организованные наборы данных, служащие для хранения информации об элементах исходной программы, которые затем используются для порождения текста результирующей программы. Таблица идентификаторов в конкретной реализации компилятора может быть одна, или же таких

Проход — это процесс последовательного чтения компилятором данных из внешней памяти, их обработки и помещения результата работы во внешнюю память. Результатом промежуточных проходов является внутреннее представление исходной программы, результатом последнего прохода — результирующая объектная программа.

Реальные компиляторы выполняют, как правило, от двух до пяти проходов. Наиболее распространены двух- и трехпроходные компиляторы, например: первый проход — *лексический анализ*, второй — *синтаксический разбор* и *семантический анализ*, третий — *генерация и оптимизация кода*.

Трансляторы с языка ассемблера («ассемблеры»)

Язык ассемблера — это язык низкого уровня. Структура и взаимосвязь цепочек этого языка близки к машинным командам целевой вычислительной системы, где должна выполняться результирующая программа.

Применение языка ассемблера позволяет разработчику управлять ресурсами (процессором, оперативной памятью, внешними устройствами и т. п.) целевой вычислительной системы на уровне машинных команд.

Компиляторы с языка ассемблера реализуются чаще всего по двухпроходной схеме.

На первом проходе компилятор выполняет разбор исходной программы, ее преобразование в машинные коды и одновременно заполняет таблицу идентификаторов.

Но на первом проходе в машинных командах остаются незаполненными адреса тех операндов, которые размещаются в оперативной памяти. На втором проходе компилятор заполняет эти адреса и одновременно обнаруживает неописанные идентификаторы.

Макроопределения и макрокоманды

Макрокоманда представляет собой текстовую подстановку, в ходе выполнения которой каждый идентификатор определенного вида заменяется на цепочку символов из некоторого хранилища данных. Процесс выполнения макрокоманды называется макрогенерацией, а цепочка символов, получаемая в результате выполнения макрокоманды, — макрорасширением.

Процесс выполнения макрокоманд заключается в последовательном просмотре текста исходной программы, обнаружении в нем определенных идентификаторов и их замене на соответствующие строки символов.

Для того чтобы указать, какие идентификаторы на какие строки необходимо заменять, служат макроопределения.

Макроопределения присутствуют непосредственно в тексте исходной программы. Они выделяются специальными ключевыми словами либо разделителями, которые не могут встречаться нигде больше в тексте программы.

В процессе обработки все макроопределения полностью исключаются из текста входной программы, а содержащаяся в них информация запоминается для обработки при выполнении макрокоманд.

2 Генерация и оптимизация кода

С целью повысить эффективность компиляторов разбор цепочек входного языка выполняется в два этапа:

- первый — синтаксический разбор на основе распознавателя одного из известных классов КС-языков;
- второй — семантический анализ входной цепочки.

Для проверки семантической правильности входной программы необходимо иметь всю информацию о найденных лексических единицах языка. Эта информация помещается в таблицу лексем на основе конструкций, найденных синтаксическим распознавателем.

Примерами таких конструкциями являются блоки описания констант и идентификаторов или операторы, где тот или иной идентификатор встречается впервые.

Входными данными для семантического анализа служат:

- таблица идентификаторов;
- результаты разбора синтаксических конструкций входного языка.

Семантический анализ обычно выполняется на двух этапах компиляции: на этапе синтаксического разбора и в начале этапа подготовки к генерации кода.

Семантический анализатор выполняет следующие основные действия:

- проверка соблюдения во входной программе семантических соглашений входного языка;
- дополнение внутреннего представления программы в компиляторе операторами и действиями, неявно предусмотренными семантикой входного языка;
- проверка элементарных семантических (смысловых) норм языков программирования, напрямую не связанных с входным языком

Проверка соблюдения во входной программе семантических соглашений входного языка заключается в сопоставлении входных цепочек программы с требованиями семантики входного языка программирования.

Каждый ЯП имеет четко заданные и специфицированные семантические соглашения, которые не могут быть проверены на этапе синтаксического разбора.

Примерами таких соглашений являются следующие требования:

- каждая метка, на которую есть ссылка, должна один раз присутствовать в программе;
- каждый идентификатор должен быть описан один раз, и ни один идентификатор не может быть описан более одного раза (с учетом блочной структуры описаний);
- все операнды в выражениях и операциях должны иметь типы, допустимые для данного выражения или операции;
- типы переменных в выражениях должны быть согласованы между собой;

- при вызове процедур и функций число и типы фактических параметров должны быть согласованы с числом и типами формальных параметров.

Преобразование типов — это только один вариант операций, неявно добавляемых компилятором в код программы на основе семантических соглашений.

Другим примером такого рода операций могут служить операции вычисления адреса, когда происходит обращение к элементам сложных структур данных.

Проверка элементарных смысловых норм языков программирования, напрямую не связанных с входным языком, — это сервисная функция, которую предоставляют большинство современных компиляторов. Эта функция обеспечивает проверку компилятором некоторых соглашений.

Примерами таких соглашений являются следующие требования:

- каждая переменная или константа должна хотя бы один раз использоваться в программе;

- каждая переменная должна быть определена до ее первого использования при любом ходе выполнения программы (до использования переменной должно быть присвоено значение);
- результат функции должен быть определен при любом ходе ее выполнения;
- каждый оператор в исходной программе должен иметь возможность хотя бы один раз выполниться;
- операторы условия и выбора должны предусматривать возможность хода вы полнения программы по каждой из своих ветвей;

Идентификация лексических единиц языков программирования

Идентификация переменных, типов, процедур, функций и других лексических единиц языков программирования — это установление однозначного соответствия между данными объектами и их именами в тексте исходной программы.

Идентификация лексических единиц языка чаще всего выполняется на этапе семантического анализа.

Распределение памяти. Принципы распределения памяти

Распределение памяти — это процесс, который ставит в соответствие лексическим единицам исходной программы адрес, размер и атрибуты области памяти, необходимой для этой лексической единицы.

Область памяти — это блок ячеек памяти, выделяемый для данных, каким-то образом объединенных логически.

Логика таких объединений задается семантикой исходного языка.

Исходными данными для процесса распределения памяти в компиляторе служат таблица идентификаторов.

Процесс распределения памяти в современных компиляторах работает с относительными, а не абсолютными адресами ячеек памяти.

Распределение памяти выполняется перед генерацией кода результирующей программы, потому что его результаты должны быть использованы в процессе генерации кода.

Память можно разделить на локальную и глобальную память, динамическую и статическую память.

Глобальная область памяти — это область памяти, которая выделяется один раз при инициализации результирующей программы и действует все время выполнения программы. Доступна из любой точки исходной программы.

Локальная область памяти — это область памяти, которая выделяется в начале выполнения некоторого фрагмента результирующей программы и может быть освобождена по завершении выполнения данного фрагмента. Доступна только в определенном фрагменте программы.

Статическая область памяти — это область памяти, размер которой известен на этапе компиляции.

Динамическая область памяти — это область памяти, размер которой на этапе компиляции программы не известен. Размер динамической области памяти будет известен только в процессе выполнения результирующей программы. Динамические области памяти можно разделить на динамические области памяти, выделяемые пользователем, и динамические области памяти, выделяемые непосредственно компилятором.

Динамические области памяти сами по себе

Дисплей памяти процедуры (функции). Стековая организация дисплея памяти

Дисплей памяти процедуры (функции) — это область данных, доступных для обработки в этой процедуре (функции).

Дисплей памяти процедуры включает следующие составляющие:

- глобальные данные (переменные и константы) всей программы;
- формальные аргументы процедуры;
- локальные данные (переменные и константы) данной процедуры.

Стековая организация дисплея памяти процедуры позволяет организовать рекурсию вызовов и имеет все преимущества перед динамической схемой, но недостаток — память, отводимая под стек параметров, используется неэффективно.

Стек параметров должен иметь размер, достаточный для хранения всех параметров, локальных данных и адресов при самом глубоком вложенном вызове процедуры в результирующей программе.

Размер стека обычно выбирается разработчиком программы «с запасом», чтобы обеспечить любую возможную глубину вложенных вызовов процедур и функций.

Память для типов данных (RTTI-информация)

Результирующая программа может обрабатывать не только переменные, константы и другие структуры данных, но и информацию о типах данных, описанных в исходной программе. Эта информация получила название RTTI — Run Time Type Information — «информация о типах во время выполнения».

Состав RTTI-информации определяется семантикой входного языка и реализацией компилятора.

RTTI-информация хранится в результирующей программе в виде RTTI-таблицы. RTTI-таблица это глобальная статическая структура данных,

Каждому типу данных в RTTI-таблице соответствует своя область данных, в которой хранится вся необходимая информация об этом типе данных, его взаимосвязи с другими типами данных в общей иерархии типов данных программы, а также все указатели на код виртуальных процедур и функций, связанных с этим типом.

Компилятор всегда сам автоматически строит код, ответственный в результирующей программе за создание и заполнение RTTI-таблицы и за ее взаимосвязь с экземплярами объектов (классов) различных типов.

Генерация кода. Методы генерации кода

Генерация объектного кода — это перевод компилятором внутреннего представления исходной программы в цепочку символов выходного языка.

Внутреннее представление программы может иметь любую структуру в зависимости от реализации компилятора, в то время как результирующая программа всегда представляет собой линейную последовательность команд.

Компилятор выполняет генерацию результирующего кода поэтапно, на основе законченных синтаксических конструкций входной программы.

В качестве анализируемых законченных синтаксических конструкций выступают операторы, блоки операторов, описания процедур и функций. Смысл (семантику) каждой такой синтаксической конструкции входного языка можно определить, исходя из ее типа на основании грамматики входного языка. Примерами типов синтаксических конструкций могут служить операторы цикла, условные операторы, операторы выбора и т. д.

Одни и те же типы синтаксических конструкций характерны для различных ЯП, при этом они различаются *синтаксисом* (который задается грамматикой языка), но имеют *схожий смысл* (который определяется семантикой).

Для семантически схожих конструкций различных входных языков программирования может порождаться типовой результирующий код.

Чтобы компилятор мог построить код результирующей программы для синтаксической конструкции входного языка используется метод, называемый синтаксически управляемым переводом — *СУ-переводом.*

СУ-перевод — это основной метод порождения кода результирующей программы на основании результатов синтаксического разбора.

Для удобства понимания сути метода можно считать, что результат синтаксического разбора представлен в виде дерева синтаксического разбора (дерева операций), хотя в реальных компиляторах это не всегда так.

Общие принципы оптимизации кода

Эффективность результирующей программы важна для ее разработчика, т.к. современные компиляторы выполняют еще один этап компиляции — оптимизацию результирующей программы.

Важно отметить два момента:

1) выделение оптимизации в отдельный этап генерации кода — это вынужденный шаг. Компилятор вынужден производить оптимизацию построенного кода, поскольку он не может выполнить семантический анализ всей входной программы в целом и построить результирующую программу.

Оптимизация нужна, поскольку результирующая программа строится не вся сразу, а поэтапно.

2) оптимизация — это необязательный этап

Компилятор может вообще не выполнять оптимизацию, и при этом результирующая программа будет правильной, а сам компилятор будет полностью выполнять свои функции.

Оптимизация программы — это обработка, связанная с переупорядочиванием и изменением операций в компилируемой программе с целью получения более эффективной результирующей объектной программы.

Оптимизация выполняется на этапах подготовки к генерации и непосредственно при генерации объектного кода.

В качестве показателей эффективности результирующей программы можно использовать два критерия:

- объем памяти, необходимый для хранения всех данных и кода результирующей программы
- скорость выполнения (быстродействие) программы.

Оптимизацию можно выполнять на любой стадии генерации кода, начиная от завершения синтаксического разбора и вплоть до последнего этапа, когда порождается код результирующей программы.

Оптимизация в компиляторе может выполняться несколько раз на этапе генерации кода.

Принципиально различаются два основных вида оптимизирующих преобразований:

- преобразования исходной программы (в форме ее внутреннего представления в компиляторе), не зависящие от результирующего объектного языка;
- преобразования результирующей объектной программы.

Первый вид преобразований не зависит от архитектуры целевой вычислительной системы, на которой будет выполняться результирующая программа.

Второй вид преобразований может зависеть от свойств объектного языка и архитектуры вычислительной системы, на которой будет выполняться результирующая программа.

Оптимизация может выполняться для следующих типовых синтаксических конструкций:

- линейных участков программы;
- логических выражений;
- циклов;
- вызовов процедур и функций;
- других конструкций входного языка.

Во всех случаях могут использоваться как машинно-зависимые, так и машинно-независимые методы оптимизации.

Существуют методы, позволяющие снизить затраты кода и времени выполнения на передачу параметров в процедуры и функции и повысить в результате эффективность результирующей программы:

- передача параметров через регистры процессора;
- подстановка кода функции в вызывающий объектный код.

В большинстве современных компиляторов метод передачи параметров в процедуры и функции выбирается самим компилятором автоматически в процессе генерации кода для каждой процедуры и функции. Однако разработчик имеет возможность запретить передачу параметров через регистры процессора либо для определенных функций, либо для всех

Некоторые языки программирования (C++) позволяют разработчику исходной программы явно указать, какие параметры или локальные переменные процедуры он желал бы разместить в регистрах процессора. Тогда компилятор стремится распределить свободные регистры в первую очередь именно для этих параметров и переменных.

Метод подстановки кода функции в вызывающий объектный код основан на том, что объектный код функции непосредственно включается в вызывающий объектный код всякий раз в месте вызова функции.

Для разработчика исходной программы такая функция ничем не отличается от любой другой функции, но для вызова

3 Современные системы программирования

Понятие и структура системы программирования

Компилятор является составной частью системного ПО. *Компиляторы — это средства, служащие для создания ПО на этапах кодирования, тестирования и отладки.*

Основные технические средства, используемые в комплексе с компиляторами, включают в себя следующие программные модули:

- текстовые редакторы;
- компоновщики, позволяющие объединять несколько объектных модулей в единое целое;
- библиотеки прикладных программ, содержащие функции и подпрограммы в виде готовых объектных модулей;
- загрузчики, обеспечивающие подготовку готовой программы к выполнению;
- отладчики, выполняющие программу в заданном

В задачу разработчика программы входило обеспечить взаимосвязь всех используемых технических средств:

- подать входные данные в виде текста исходной программы на вход компилятора;
- получить от компилятора результаты его работы в виде набора объектных файлов;
- подать весь набор полученных объектных файлов вместе с необходимыми библиотеками подпрограмм на вход компоновщику;
- получить от компоновщика единый файл программы (исполняемый файл) и подготовить его к выполнению с помощью загрузчика;
- поставить программу на выполнение, при необходимости использовать отладчик для проверки правильности выполнения программы.

Все эти действия выполнялись с помощью последовательности команд, инициировавших запуск соответствующих программных модулей с передачей им всех необходимых параметров. Пользователи могли выполнять эти команды последовательно вручную, а с развитием средств командных процессоров ОС они стали объединять их в командные файлы.

Для написания командных файлов компиляции был предложен специальный командный язык — язык Makefile. Он позволял в достаточно гибкой и удобной форме описать весь процесс создания программы от порождения исходных текстов до подготовки ее к выполнению. Это было удобное, но достаточно сложное техническое средство, требующее от разработчика высокой степени подготовки и профессиональных знаний, поскольку сам командный язык Makefile был по сложности сравним с простым языком программирования.

Язык Makefile стал стандартным средством, единым для компиляторов всех разработчиков.

Следующим шагом в развитии средств разработки стало появление так называемой «интегрированной среды разработки». Интегрированная среда объединила в себе возможности текстовых редакторов исходных текстов программ и командный язык компиляции. Создание интегрированных сред разработки стало возможным благодаря бурному развитию персональных компьютеров и появлению развитых средств интерфейса пользователя. Их появление на рынке определило дальнейшие развитие такого рода технических средств.

Дальнейшее развитие средств разработки также тесно связано с повсеместным распространением развитых средств графического интерфейса пользователя. Такой интерфейс стал неотъемлемой составной частью многих современных ОС и так называемых графических оболочек.

В состав ПО были сначала включены соответствующие библиотеки, обеспечивающие поддержку развитого графического интерфейса пользователя и взаимодействие с функциями API (прикладной программный интерфейс) операционных систем.

Для описания графических элементов программ потребовались соответствующие языки. На их основе сложилось понятие «ресурсов» (resources) прикладных программ.

Ресурсами прикладной программы будем называть множество данных, обеспечивающих внешний вид интерфейса пользователя этой программы и не связанных напрямую с логикой выполнения программы.

Примеры ресурсов: тексты сообщений, выдаваемых программой; цветовая гамма элементов интерфейса; надписи на таких элементах, как кнопки и заголовки окон и т. п.

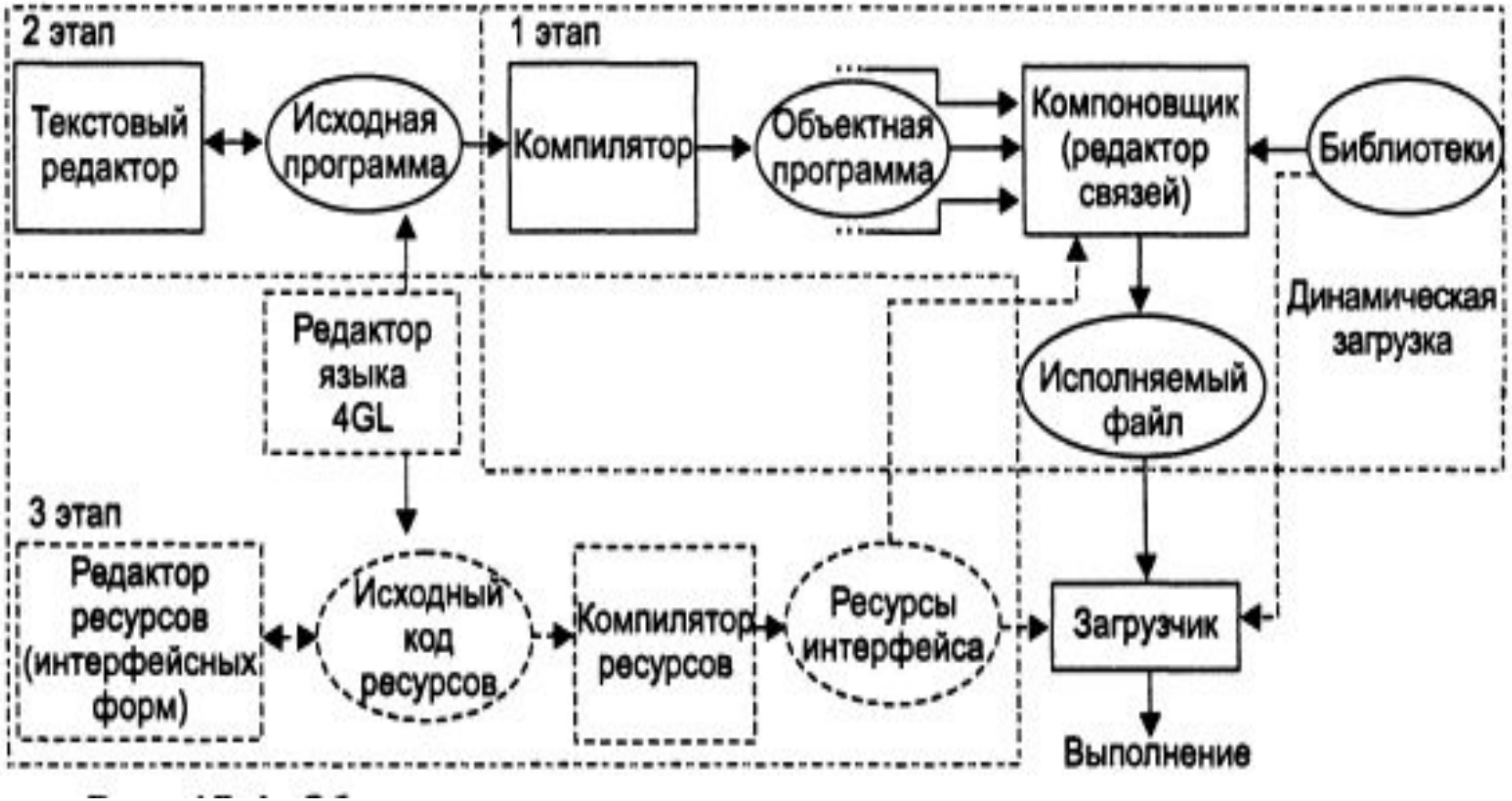
Для формирования структуры ресурсов потребовались редакторы ресурсов, а затем и компиляторы ресурсов, обрабатывающие результат их работы.

Ресурсы, полученные с выхода компиляторов ресурсов, стали обрабатываться компоновщиками и загрузчиками.

Весь этот комплекс программно-технических средств в настоящие время составляет новое понятие, которое здесь названо **«системой программирования».**

Структура современной системы программирования

Системой программирования будем называть весь комплекс программных средств, предназначенных для кодирования, тестирования и отладки ПО.



В качестве основных тенденций в развитии современных систем программирования следует указать внедрение в них средств разработки на основе так называемых «языков четвертого поколения» — 4GL (four generation languages), — а также поддержка систем «быстрой разработки программного обеспечения» — RAD (rapid application development).

Языки четвертого поколения — 4GL — представляют собой широкий набор средств, ориентированных на проектирование и разработку программного обеспечения.

Они строятся на основе графических образов.

4GL составляют часть средств автоматизированного проектирования и разработки программного обеспечения, поддерживающих все этапы жизненного цикла — CASE-систем.

Принципы функционирования систем программирования

Функции текстовых редакторов в системах программирования

Текстовый редактор в системе программирования — это программа, позволяющая создавать, изменять и обрабатывать исходные тексты программ на языках высокого уровня.

Возникновение интегрированных сред разработки на определенном этапе развития средств разработки программного обеспечения позволило непосредственно включить текстовые редакторы в состав этих средств.

В итоге в современных системах программирования текстовый редактор стал важ ной составной частью, которая позволяет подготавливать исходные тексты программ и выполняет все интерфейсные и сервисные функции.

Компилятор как составная часть системы программирования

Компиляторы являются основными модулями в составе любой системы программирования (СП).

Без компилятора никакая СП не имеет смысла, а все остальные ее составляющие на самом деле служат лишь целям обеспечения работы компилятора и выполнения им своих функций.

Именно технические характеристики компилятора, прежде всего, влияют на эффективность результирующих программ, порождаемых СП.

Компоновщик. Назначение и функции компоновщика

Компоновщик (редактор связей) предназначен для связывания между собой объектных файлов, порождаемых компилятором, а также файлов библиотек, входящих в состав системы программирования.

Объектный файл не может быть исполнен до тех пор, пока все модули и секции не будут в нем увязаны между собой. Это и делает редактор связей (компоновщик). Результатом его работы является единый файл - «исполняемый файл», который содержит весь текст результирующей программы на языке машинных кодов. Компоновщик может порождать сообщение об ошибке, если при попытке собрать объектные файлы в единое целое он не смог обнаружить какой-либо необходимой составляющей.

Обычно компоновщик формирует простейший программный модуль, создаваемый как единое целое.

Загрузчики и отладчики. Функции загрузчика

Компилятор и компоновщик не могут знать точно, в какой реальной области памяти компьютера будет располагаться программа в момент ее выполнения. Поэтому они работают не с реальными адресами ячеек ОЗУ, а с некоторыми относительными адресами. Такие адреса отсчитываются от некоторой условной точки, принятой за начало области памяти, занимаемой результирующей программой.

Программа не может быть исполнена в относительных адресах. Поэтому требуется модуль, который бы выполнял преобразование относительных адресов в реальные (абсолютные) адреса непосредственно в момент запуска программы на выполнение. Этот процесс называется *трансляцией адресов* и выполняет его специальный модуль, называемый *загрузчиком*. Методы трансляции адресов могут быть основаны на сегментной, страничной и

Отладчик — это программный модуль, который позволяет выполнить основные задачи, связанные с мониторингом процесса выполнения результирующей прикладной программы. Этот процесс называется *отладкой* и включает в себя следующие основные возможности:

- последовательное пошаговое выполнение результирующей программы на основе шагов по машинным командам или по операторам входного языка;
- выполнение результирующей программы до достижения ею одной из заданных точек останова (адресов останова);
- выполнение результирующей программы до наступления некоторых заданных условий, связанных с данными и адресами, обрабатываемыми этой программой;

Библиотеки подпрограмм как составная часть систем программирования

Библиотеки подпрограмм составляют существенную часть систем программирования. Они состоят из двух основных компонентов: файл(ы) библиотеки, содержащий объектный код, и набор файлов описаний функций, подпрограмм, констант и переменных, составляющих библиотеку. Описания оформляются на соответствующем входном языке (для языка C++ это будет набор заголовочных файлов). Набор файлов описания библиотеки служит для информирования компилятора о составе входящих в библиотеку функций. Обрабатывая эти файлы, компилятор получает всю необходимую информацию о составе библиотеки с точки зрения входного ЯП. Эти файлы предназначены только для того, чтобы избавить разработчика от необходимости постоянного описания библиотечных функций, подпрограмм, констант и переменных.

Динамические библиотеки в отличие от традиционных (статических) библиотек подключаются к программе не в момент ее компоновки, а непосредственно в ходе выполнения, как только программа затребовала ту или иную функцию, находящуюся в библиотеки. Широкий набор динамических библиотек поддерживается всеми современными ОС и содержат системные функции ОС и общедоступные функции программного интерфейса (API).

Дополнительные возможности систем программирования

Лексический анализ «на лету» — это функция текстового редактора в составе СП. Она заключается в поиске и выделении лексем входного языка в тексте программы непосредственно в процессе ее создания разработчиком. Система подсказок и справок в настоящее время является составной частью многих

Разработка программ в архитектуре «клиент—сервер»

Среди всего множества компонентов прикладной программы можно было выделить две логически цельные составляющие: первая — обеспечивающая «нижний уровень» работы приложения, отвечающая за методы хранения, доступа и разделения данных; вторая — организующая «верхний уровень» работы приложения, включающий в себя логику обработки данных и интерфейс пользователя.

Тогда сложилось понятие приложения, построенного на основе архитектуры «клиент—сервер».

В первую (серверную) составляющую такого приложения относят все методы, связанные с доступом к данным. Чаще всего их реализует сервер БД (сервер данных) из соответствующей СУБД (системы управления базами данных) в комплекте с драйверами доступа к нему.

Во вторую (клиентскую) часть приложения относят все методы обработки данных и представления их пользователю. Клиентская часть взаимодействует, с одной стороны, с сервером, получая от него данные, а с другой стороны — с пользователем, ресурсами приложения и ОС, осуществляя обработку данных и отображение результатов.

Разработка программ в трехуровневой архитектуре. Серверы приложений

Трехуровневая архитектура разработки приложений явилась логическим продолжением идей, заложенных в архитектуре «клиент—сервер».

Многие приложения стали нуждаться в предоставлении пользователю возможности доступа к данным посредством сети.

Поэтому дальнейшим развитием архитектуры «клиент—сервер» стало разделение клиентской части в свою очередь еще на две составляющих: *сервер приложений,* реализующий обработку данных, и *«тонкий клиент»,* обеспечивающий интерфейс и доступ к результатам обработки.

Разделение клиентской части на две составляющих потребовало организации взаимодействия между этими составляющими.

Стали появляться новые интерфейсы обмена данными. Среди них можно выделить семейство стандартов COM/DCOM, предложенных фирмой Microsoft для ОС семейства Microsoft Windows, а также семейство стандартов CORBA (Common Object Request Broker Architecture), поддерживаемое широким кругом производителей и разработчиков программного обеспечения для большого спектра различных ОС.

Системы программирования в настоящее время ориентируются на поддержку средств разработки приложений в трехуровневой (трехзвенной) архитектуре.

Примеры современных систем программирования

1 Системы программирования компании Borland/Inprise

Turbo Pascal. Pascal был предложен Н. Виртом в конце 70-х годов как хорошо структурированный учебный язык. В Borland Pascal были внесены компанией Borland расширения, которые преследовали две основные цели:

1. упрощение обработки в языке структур, представляющих распространенные типы данных — строки и файлы;
2. реализация в языке основных возможностей объектно-ориентированных языков программирования.

Borland Delphi. Система программирования Borland Delphi явилась логическим продолжением и дальнейшим развитием идей, заложенных компанией-разработчиком еще в системе программирования Turbo Pascal.

В Borland Delphi появились принципиальные изменения:

- новый язык программирования — Object Pascal;
- компонентная модель среды разработки, ориентированная на технологию разработки RAD.

Система программирования Borland Delphi предназначена для создания результирующих программ, выполняющихся в среде ОС Windows различных типов. Основу системы программирования Borland Delphi и ее компонентной модели составляет библиотека VCL. В этой библиотеке реализованы в виде компонентов все основные органы управления и интерфейса ОС. Для поддержки разработки результирующих программ для архитектуры «клиент-сервер» в состав Borland Delphi входит средство BDE. Оно обеспечивает результирующим программам возможность доступа к широкому диапазону серверов БД посредством классов библиотеки VCL.

2 Системы программирования фирмы Microsoft

Microsoft Visual Basic. Система программирования Microsoft Visual Basic является одним из эффективных средств для создания результирующих программ, ориентированных на выполнение под управлением ОС типа Microsoft Windows. Эта система программирования ориентирована на технологию разработки RAD.

Microsoft Visual Basic 6.0 содержит интегрированные средства визуальной работы с БД, поддерживающие проектирование и доступ к базам данных SQL Server, Oracle и т. п. MS Visual Basic 6.0 обеспечивает простое создание приложений, ориентированных на данные. Система программирования Microsoft Visual Basic ориентирована на создание клиентской части приложений.

Visual Basic поддерживает универсальный интерфейс доступа к данным Microsoft при помощи технологии ADO.

Microsoft Visual C++. Эта система программирования в настоящее время построена в виде интегрированной среды разработки, включающей в себя все необходимые средства для разработки результирующих программ, ориентированных на выполнение под управлением ОС типа Microsoft Windows различных версий.

Основу системы программирования Microsoft Visual C++ составляет библиотека классов MFC. В этой библиотеке реализованы в виде классов C++ все основные органы управления и интерфейса ОС. Также в ее состав входят классы, обеспечивающие разработку приложений для архитектуры «клиент—сервер» и трехуровневой архитектуры. Система программирования Microsoft Visual C++ позволяет разрабатывать любые приложения, выполняющиеся в среде ОС типа Microsoft Windows, в том числе серверные или клиентские результирующие программы, осуществляющие взаимодействие между собой по одной из указанных выше архитектур.

Концепция .NET - это не система программирования, а новейшая технология, предложенная фирмой Microsoft с целью унификации процесса разработки ПО с помощью различных систем программирования.

Microsoft .NET — это целостный взгляд компании Microsoft на новую эпоху в развитии Интернета. В рамках этой концепции самые разнообразные программные приложения предоставляются пользователям и разработчикам как сервисы, которые взаимодействуют между собой в соответствии с конкретными потребностями бизнеса.

С точки зрения систем программирования основные идеи архитектуры .NET заключаются в том, что в ОС типа Windows организуется специальная виртуальная машина, исполняющая команды некоторого промежуточного низкоуровневого языка.

3 Системы программирования под ОС Linux и UNIX

Системы программирования в составе ОС типа UNIX. Язык программирования Си появился как базовый язык программирования в ОС типа UNIX. ОС типа UNIX поставляется в виде исходного текста на языке Си. Всякий раз при изменении основных параметров ОС происходит компиляция и компоновка ядра ОС заново, а при перезапуске ОС вновь созданное ядро активируется. Этот принцип характерен для всех ОС типа UNIX.

ОС типа UNIX фактически не может существовать без наличия в ее составе компилятора и компоновщика для ЯП Си. Все производители ОС типа UNIX включают в ее состав и СП языка Си.

Прикладная программа, ориентированная на ОС типа UNIX, поставляется в виде набора исходных кодов (на ЯП Си).

Системы программирования проекта GNU

Проект GNU был начат в 1984 году. Идея проекта GNU заключается в свободном распространении всех программ. Программы распространяются в виде исходного кода, причем пользователь получает этот код в пользование, имеет право вносить в него изменения, а также распространять исходный или модифицированный код.

GNU — это название полной UNIX-совместимой программной системы, которая разрабатывается и безвозмездно предоставляется всем желающим ее использовать в рамках проекта.

Система GNU способна исполнять UNIX-программы. В качестве языка в системе доступны Си и Lisp. Поддерживаются коммуникационные протоколы UUCP, MIT Chaosnet и сети Интернет. Базовой системой программирования в проекте GNU стала система программирования на языке C++.

Проект Borland Kylix

В рамках проекта Borland Kylix компания Borland создала и распространяет на рынке программного обеспечения одноименную СП, основанную на языке программирования Object Pascal. Данный язык известен в среде разработчиков для системы программирования Borland Delphi, ориентированной на ОС типа Windows.

Компания Borland реализовывает проект Borland Kylix таким образом, чтобы перенести в созданную СП под ОС Linux все черты, присущие системе программирования Borland Delphi, уже существующей под ОС типа Microsoft Windows.

Основой проекта Borland Kylix стала библиотека компонентов CLX, в которой компания Borland реализовала все основные органы управления пользовательского интерфейса и средства обработки данных,

Библиотека построена в виде компонентов на основе иерархии классов языка Object Pascal.

В рамках этого проекта разработчик, пользуясь только средствами библиотеки CLX, сможет создавать исходный код на языке Object Pascal или C++ таким образом, что результирующие программы, построенные на основе этого кода с помощью Borland Kylix, будут выполняться в ОС типа Linux, а построенные с помощью Borland Delphi или Borland C++ Builder — в ОС типа Microsoft Windows.

Проект Borland Kylix только начал развиваться.

3 Разработка программного обеспечения для сети Интернет

При осуществлении взаимодействия по сети двух компьютеров один из них выступает в качестве источника данных (Интернет-сервера), а другой — приемника данных (Интернет-клиента).

Основной особенностью программирования в сети Интернет является использование в качестве основного средства программирования интерпрети руемых языков. При интерпретации исполняется не объектный код, а сам исходный код программы, и уже непосредственно интерпретатор на стороне клиента отвечает за то, чтобы этот исходный код был исполнен всегда одним и тем же образом вне зависимости от архитектуры вычислительной системы.

Язык HTML. Программирование статических Web-страниц

Суть его достаточно проста: Интернет-сервер создает текст на языке HTML и передает его в виде текстового файла на клиентскую сторону сети по специальному протоколу обмена данными HTTP. Клиент, получая исходный текст на языке HTML, интерпретирует его и в соответствии с результатом интерпретации строит соответствующие интерфейсные формы и изображения на экране клиентского компьютера.

Программирование динамических Web-страниц

Основная идея динамической генерации Web-страниц заключается в том, что большая часть HTML-страницы не хранится в файле на сервере, а порождается непосредственно каждый раз при обращении клиента к серверу. Тогда сервер формирует страницу и сразу же по готовности передает ее клиенту.

Таким образом, всякий раз при новом обращении клиент получает новый текст HTML.

CGI (общедоступный шлюзовой интерфейс) — это интерфейс для запуска внешних программ на сервере в ответ на действия клиента, установившего соединение с ним через глобальную сеть. Пользуясь этим интерфейсом, приложения могут получать информацию от удаленного пользователя, анализировать ее, формировать HTML-код и отсылать его клиенту. CGI-приложения могут получать данные из заполненной формы, построенной с помощью HTML, либо из командной строки описания URL (универсальный указатель ресурса). CGI-приложения по каким действиям пользователя должны выполняться на сервере, указывается непосредственно в коде HTML-страницы, которую сервер передает клиенту.

Кроме интерфейса CGI существуют и другие варианты интерфейсов, позволяющие динамически создавать HTML-код путем запуска на сервере приложений в ответ на действия клиента. Например, интерфейс ISAPI (интерфейс прикладных программ Интернет-сервера). Отличие ISAPI от CGI заключается в том, что для поддержки CGI создаются отдельные приложения, выполняющиеся в виде самостоятельных программ, а ISAPI поддерживается с помощью библиотек, динамически подключаемых к серверу.

Существует несколько языков и соответствующих им интерпретаторов, которые нашли применение в этой области и успешно служат цели порождения HTML-страниц. Язык Perl лежит в основе различных версий Web-технологии PHP и язык сценариев, на котором основана Web-технология ASP, — последний предложен и поддерживается известным производителем ПО—

Текст на интерпретируемых языках, которые поддерживаются такими Web-технологиями, как ASP или PHP, представляет собой часть текста обычных HTML-страниц со встроенными в них сценариями (script). Эти сценарии можно писать на любом языке, поддерживаемом сервером; Интернет-сервер обрабатывает их при поступлении запроса о URL-адресе соответствующего файла. Он разбирает текст HTML-страницы, находит в нем тексты сценариев, вырезает их и интерпретирует в соответствии с синтаксисом и семантикой данного языка. В результате интерпретации получается выходной текст на языке HTML, который сервер вставляет непосредственно в то место исходной страницы, где встретился сценарий. Так обрабатывается динамическая Web-страница на любом интерпретируемом языке, ориентированном на работу в глобальной сети.

Все эти языки сценариев обладают присущими им характерными особенностями.

Во-первых, они имеют мощные встроенные функции и средства для работы со строками, поскольку основной задачей программ, написанных с помощью таких языков, является обработка входных параметров (строковых) и порождение HTML-кода (который также является текстом).

Во-вторых, все они имеют средства для работы в архитектуре «клиент—сервер» для обмена информацией с серверами БД, а многие современные версии таких языков (например, язык, поддерживаемый Web-технологией ASP) — средства для функционирования в трехуровневой архитектуре для обмена данными с серверами

Языки программирования Java и Java Script

Основная идея, отличающая язык Java от многих других ЯП, заключается в том, что Java не является полностью компилируемым языком. Исходная программа, созданная на языке Java, не преобразуется в машинные коды. Компилятор языка порождает некую промежуточную результирующую программу на специальном низкоуровневом двоичном коде. Именно этот код интерпретируется при исполнении результирующей Java-программы. Такой метод исполнения кода делает его независимым от архитектуры целевой вычислительной системы. Для исполнения Java-программы необходимы две *составляющие*: компилятор промежуточного двоичного кода, порождающий его из исходного текста Java-программы, и интерпретатор, исполняющий этот промежуточный двоичный код. Такой интерпретатор получил название виртуальной Java-машины.

Одной из отличительных особенностей данного языка является использование специального механизма распределения памяти (менеджера памяти), который должен сам организовывать своевременное выделение областей памяти при создании новых классов и объектов, а затем освобождать области памяти, которые уже больше не используются. В последнем случае должна решаться непростая задача сборки мусора — поиска неиспользуемых фрагментов в памяти и их освобождение.

При выполнении Java-программы в глобальной сети компилятор, порождающий промежуточный низкоуровневый код, находится на стороне Интернет-сервера, а интерпретатор, выполняющий этот код, — на стороне клиента.

По сети от сервера к клиенту передается только уже скомпилированный код.

JBuilder. Необходимость иметь в составе архитектуры вычислительной системы клиентского компьютера виртуальную Java-машину, а также довольно высокие требования к производительности компьютеров на клиентской стороне в ряде случаев ограничивают возможности применения языка Java. Чтобы решить эти проблемы, был предложен командный язык Java Script, который можно назвать упрощенным вариантом языка Java.

Синтаксис и семантика Java Script в целом соответст вуют языку Java, но возможности его выполнения сильно ограничены — они не выходят за рамки той программы, которая интерпретирует HTML-страницу. В таком случае эта программа выступает и в роли виртуальной Java-машины для выполнения операторов Java Script. Чаще всего это программа навигации по сети — браузер. Он находит в тексте HTML-страницы операторы языка Java Script, выделяет их и исполняет по всем правилам языка Java. Операторы Java Script сейчас широко используются для организации динамических Web-страниц.

Вопросы для самопроверки

1. Что такое трансляция, компиляция, транслятор, компилятор? Из каких процессов состоит компиляция?
2. Какую роль выполняет лексический анализ в процессе компиляции? Как могут быть связаны между собой лексический и синтаксический анализ?
3. Компилятор можно построить, не используя лексический анализатор. Объясните, почему все-таки подавляющее большинство современных компиляторов используют фазу лексического анализа.
4. От чего зависит количество проходов, необходимых компилятору для построения результирующей объектной программы на основе исходной программы?
5. Почему для языка ассемблера, который имеет простую грамматику, чаще всего используется двухпроходный компилятор? Что мешает свести компиляцию исходного кода на языке ассемблера в один проход?
6. Что такое таблица идентификаторов и для чего она предназначена? Какая информация может храниться в таблице идентификаторов?

Вопросы для самопроверки

7. Чем различаются таблица лексем и таблица идентификаторов?

8. Какие задачи в компиляторе решает семантический анализ?

9. Какие специфические функции выполняет текстовый редактор в составе системы программирования?

10. В чем особенности функционирования компилятора в составе системы программирования по сравнению с его функционированием в виде отдельного программного модуля?

11. Почему компоновщик носит также название «редактор связей»? В чем заключается процедура разрешения внешних ссылок объектного файла?

12. Какие дополнительные возможности предоставляет пользователю лексический анализ исходного текста программы «на лету»?

13. В чем заключаются основные особенности разработки программного обеспечения для глобальной сети Интернет? Почему в этой сети часто используются интерпретируемые языки?