[4. Основные понятия.](#_Toc515769877)

[4.1. История языка С.](#_Toc515769878)

[4.1.1. Взаимоотношения с другими языками.](#_Toc515769879)

[4.1.2. Достоинства языка С.](#_Toc515769880)

[4.1.2.1. Малый размер.](#_Toc515769881)

[4.1.2.2. Набор команд языка.](#_Toc515769882)

[4.1.2.3. Быстродействие.](#_Toc515769883)

[4.1.2.4. Язык со слабой типизацией.](#_Toc515769884)

[4.1.2.5. Структурированный язык.](#_Toc515769885)

[4.1.2.6. Поддержка модульного программирования.](#_Toc515769886)

[4.1.2.7. Простой интерфейс с ассемблерными подпрограммами.](#_Toc515769887)

[4.1.2.8. Поразрядная обработка.](#_Toc515769888)

[4.1.2.9. Переменные-указатели.](#_Toc515769889)

[4.1.2.10. Гибкие структуры.](#_Toc515769890)

[4.1.2.11. Эффективность использования памяти.](#_Toc515769891)

[4.1.2.12. Переносимость.](#_Toc515769892)

[4.1.2.13. Библиотеки специальных функций.](#_Toc515769893)

[4.1.3. Недостатки языка С.](#_Toc515769894)

[4.1.3.1. Слабая типизация.](#_Toc515769895)

[4.1.3.2. Отсутствие проверок на этапе исполнения.](#_Toc515769896)

[4.1.4. Использование языка Си.](#_Toc515769897)

[4.1.5. Будущее языка Си.](#_Toc515769898)

[4.2. Исходные файлы и выполняемые файлы.](#_Toc515769899)

[4.3. Принципы программирования.](#_Toc515769900)

[4.4. Стандарт ANSI С.](#_Toc515769901)

[4.5. Эволюция языка C++ и объектно-ориентированное программирование.](#_Toc515769902)

[4.6. История C++.](#_Toc515769903)

[4.6.1. Использование объектов C++ для быстрого создания программы.](#_Toc515769904)

[4.6.2. Некоторые усовершенствования по сравнению с языком С.](#_Toc515769905)

[4.6.2.1. Комментарии.](#_Toc515769906)

[4.6.2.2. Имена перечисляемых типов.](#_Toc515769907)

[4.6.2.3. Имена структуры или класса.](#_Toc515769908)

[4.6.2.4. Блочные объявления.](#_Toc515769909)

[4.6.2.5. Операция уточнения области действия (scope).](#_Toc515769910)

[4.6.2.6. Описатель const.](#_Toc515769911)

[4.6.2.7. Анонимные объединения.](#_Toc515769912)

[4.6.2.8. Явное преобразование типов.](#_Toc515769913)

[4.6.2.9. Объявления функций.](#_Toc515769914)

[4.6.2.10. Перегруженные функции.](#_Toc515769915)

[4.6.2.11. Значения параметров функций по умолчанию.](#_Toc515769916)

[4.6.2.12. Функции с неуказанным числом параметров.](#_Toc515769917)

[4.6.2.13. Ссылочные параметры функции.](#_Toc515769918)

[4.6.2.14. Операторы newи delete.](#_Toc515769919)

[4.6.2.15. Указатели void и функции, возвращающие void.](#_Toc515769920)

[4.6.3. Основные усовершенствования по сравнению с языком С.(часто повторяется)](#_Toc515769921)

[4.6.3.1. Конструкторы классов и инкапсуляция данных.](#_Toc515769922)

[4.6.3.2. Класс struct.](#_Toc515769923)

[4.6.3.3. Конструкторы и деструкторы.](#_Toc515769924)

[4.6.3.4. Сообщения.](#_Toc515769925)

[4.6.3.5. "Дружественные" классы.](#_Toc515769926)

[4.6.3.6. Перегрузка операций.](#_Toc515769927)

[4.6.3.7. Производные классы.](#_Toc515769928)

[4.6.3.8. Полиморфизм при использовании виртуальных функций.](#_Toc515769929)

[4.6.3.9. Библиотеки потоков.](#_Toc515769930)

[4.7. Базовые элементы программы на С.](#_Toc515769931)

[4.7.1. Пять основных компонентов программы.](#_Toc515769932)

[4.7.2. Различия между программами в ЕХЕ - и СОМ - файлах](#_Toc515769933)

1. Основные понятия.

История языка С начинается с обсуждения операционной системы UNIX,поскольку сама эта система и большинство работающих на ней программ написаны на С. Это, однако, не означает, что С привязан к UNIX или любой другой операционной системе или аппаратной платформе. Общая Среда разработки UNIX/C создала языку С репутацию языка системного программирования, так как он удобен для написания компиляторов и операционных систем. Но С также удобен и для создания важнейших программ в самых различных областях.

Первоначально UNIX была разработана в 1969 году на компьютере, относящемся по нынешним меркам к мини-ЭВМ, DECPDP-7 в лаборатории фирмы Белл, Мюррей Хилл, штат Нью-Джерси (BellLaboratories — MurrayHill, NewJersey). UNIX была написана исключительно на языке ассемблера PDP-7. По идее, эта операционная система должна была быть "дружественной" для программиста, обеспечивая мощные средства разработки, компактные команды и сравнительно открытую среду. Вскоре после разработки UNIX, Кен Томпсон(KenThompson) создал компилятор для нового языка, названного В.

Здесь интересно исследовать истоки и историю созданного Томпсоном языка В, непосредственного предшественника С. Далее следует полная родословная языка С:

Начиная с этой темы, вы будете знакомиться с происхождением, синтаксисом и использованием языков С и C++. Полезно узнать историю языка С, так как она поможет вам понять причины успеха этого языка, и то, почему С и C++ могут стать вашим выбором на ближайшие годы.

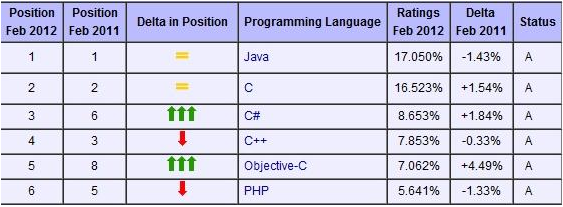
Среди современных языков программирования язык СИ является одним из самых распространённых. Язык СИ универсален, однако наиболее эффективно его применение в задачах системного программирования – разработке трансляторов, операционных систем, экранных интерфейсов, инструментальных средств, игр. Язык СИ хорошо зарекомендовал себя эффективностью, лаконичностью записи алгоритмов, логической стройностью программы. Во многих случаях программы, написанные на языке СИ, сравнимы по скорости с программами, написанными на языке ассемблера, при этом они более наглядны и просты в сопровождении.

Удобство языка СИ основано на том, что он является одновременно и языком высокого уровня, имеющий полный набор конструкций структурного программирования, поддерживающий модульность, блочную структуру программ, возможность раздельной компиляции модулей.

Одним из основных достоинств языка СИ считается высокая переносимость написанных на нём программ между компьютерами с различными архитектурами, между различными операционными системами.

Язык СИ был разработан в США сотрудниками фирмы BellLaboratories на рубеже 70-х годов. Первое описание языка СИ было дано его авторами, Б.Керниганом и Д.Ритчи.

Гибкость и универсальность языка СИ обеспечивает его широкое применение. Популярность и продвинутость языка следует из интернет ресурсов



## История языка С.

История языка С начинается с обсуждения операционной системы UNIX,поскольку сама эта система и большинство работающих на ней программ написаны на С. Это, однако, не означает, что С привязан к UNIX или любой другой операционной системе или аппаратной платформе. Общая Среда разработки UNIX/C создала языку С репутацию языка системного программирования, так как он удобен для написания компиляторов и операционных систем. Но С также удобен и для создания важнейших программ в самых различных областях.

Первоначально UNIX была разработана в 1969 году на компьютере, относящемся по нынешним меркам к мини-ЭВМ, DECPDP-7 в лаборатории фирмы Белл, Мюррей Хилл, штат Нью-Джерси (BellLaboratories — MurrayHill, NewJersey). UNIX была написана исключительно на языке ассемблера PDP-7. По идее, эта операционная система должна была быть "дружественной" для программиста, обеспечивая мощные средства разработки, компактные команды и сравнительно открытую среду. Вскоре после разработки UNIX, Кен Томпсон (KenThompson) создал компилятор для нового языка, названного В.

Здесь интересно исследовать истоки и историю созданного Томпсоном языка В, непосредственного предшественника С. Далее следует полная родословная языка С:

|  |  |
| --- | --- |
| Алгол 60 | Разработан международным комитетом в начала 1960-х |
| CPL | (CombinedProgrammingLanguage — Объединенный Язык Программирования) Разработан параллельно в Кембридже и Лондонском Университете в 1963 |
| BCPL | (BasicCombinedProgrammingLanguage — Базовый PCL) Разработан в Кембрижде Мартином Ричардсом (MartinRichards) в 1967 |
| В | Разработан Кеном Томпсоном, BellLabs, в 1970 |
| С | Разработан ДэннисомРитчи (DennisRitchie), BellLabs, в 1972 |
| C++ | Бьярн Страуструп 1980 г. |

Затем, в 1983 году, в целях создания стандарта языка С (ANSI С), был образован Американский институт национальных стандартов (ANSI).

Алгол 60 появился всего несколькими годами позже того, как был представлен язык ФОРТРАН. Этот новый язык был сложнее и имел большое влияние на проектирование последующих языков программирования. Его авторы уделили громадное внимание регулярности синтаксиса, модульной структуре и другим особенностям, связываемым обычно со структурированными языками высокого уровня. К сожалению, Алгол 60 никогда не был по-настоящему популярным в США. Многие утверждают, что это произошло из-за абстрактности языка и его обобщенности.

Разработчики CPL намеревались совместить возвышенность идей Алгола 60 и прагматизм реального компьютера. Однако, так же как и Алгол 60, CPL оказался сложным для изучения и реализации. Ориентируясь на лучшие средства CPL, авторы языка BCPL попытались сузить CPL, не потеряв при этом его основных достоинств.

Когда Кен Томпсон разрабатывал язык В, предназначенный первоначально для проектирования UNIX, он пытался еще более упростить CPL. Ему удалось создать очень компактный язык, который хорошо подходил для использования на имевшемся в распоряжении автора компьютере. Однако, попытки обобщения и BCPL и языка В зашли слишком далеко: эти языки стали ограниченными, пригодными только для решения определенных проблем.

Например: как только Томпсон создал язык В, появился новый компьютер, названный PDP-11. UNIX и компилятор.Вбыли немедленно перенесены на эту новую машину. Хотя PDP-11 и превосходил своего предшественника, PDP-7, все же это был довольно слабый, по сегодняшним меркам, компьютер. Он имел только 24 Кб памяти, из которых 16 Кб занимала система, и один жесткий диск 512 Кб. Были потрачены некоторые усилия на то, чтобы переписать UNIX на языке В, однако этот язык оказался медленным в силу своей интерпретирующей природы. Была также и другая проблема: язык В был ориентирован на слова данных, а PDP-11 на байты. По этим причинам в 1971 году была начата работа по созданию преемника языка В, названного, соответственно, С.

Заслуги по созданию С принадлежат ДэннисуРитчи, создавшему язык, в котором была восстановлена некоторая обобщенность, потерянная в языках BCPL и В. Это удалось благодаря продуманному использованию типов данных при сохранении простоты и непосредственного доступа к аппаратуре, что было первоначальной задачей при разработке CPL.

Многие языки, разработанные одним человеком (С, Pascal, Lisp и APL), имеют внутреннюю связанность, которая отсутствует в языках, созданных большой группой программистов (Ада, PL/1 и Алгол 68). Для языков, написанных одним человеком, также характерно то, что они отражают профессиональную ориентацию автора. В своей работе ДэннисРитчи отвечал за системное программное обеспечение — машинные языки, операционные системы и генераторы программ.

Принимая во внимание вышеизложенное, легко понять, почему язык С выбран для проектирования системных программ. С — сравнительно низкоуровневый язык, позволяющий учитывать каждую деталь алгоритма для достижения максимальной производительности компьютера. Но С — одновременно и высокоуровневый язык, который может не учитывать особенности архитектуры компьютера, что повышает эффективность программирования.

### Взаимоотношения с другими языками.

Ваc может заинтересовать положение, которое занимает язык С по отношению к другим языкам. Возможная иерархия показана на рисунке. Если начать снизу и двигаться вверх, вы придете от материальных и эмпирических категорий к неуловимым и теоретическим понятиям. Точки предоставляют собой основные этапы развития, многие шаги пропущены. Прародители компьютера — ткацкий станок Жаккарда (Jacquard, 1805) и "думающая машина" Чарльза Бэббиджа (CharlesBabbage, 1834) программировались аппаратно. А может быть, придет день, когда для программирования машины нужно будет вставить преобразователь нейронных сигналов в гнездо, имплантированное в височной доле (языковая память) или в зоне Брока (Вгоса) (языковая моторная область) коры головного мозга.

|  |
| --- |
| Прямые нейронные связи |
| Искусственный интеллект  Командные языки операционных систем  Проблемно-ориентированные языки  Машинно-ориентированные языки  Язык ассемблера |
| Реальная аппаратура |

Рис. Возможная иерархия языков программирования

Языки ассемблера, чья история начинается с первых дней появления электронных компьютеров, обеспечивали работу непосредственно с набором встроенных команд ЭВМ и были сравнительно простыми. Поскольку языки ассемблера заставляли мыслить в категориях аппаратного обеспечения, необходимо было задавать каждое действие при помощи средств машины. Следовательно, вся работа сводилась к повторяющимся пересылкам битов между регистрами, сложению и сдвигам содержимого регистров с последующим запоминанием результатов в памяти. Эти процедуры были утомительными и сопровождались множеством ошибок.

Первые языки высокого уровня, например, ФОРТРАН, создавались как альтернатива ассемблерным языкам. Высокоуровневые языки являются значительно более обобщенными и абстрактными и позволяют программисту формулировать задачу, используя проблемные понятия, а не аппаратные термины компьютера.

К сожалению, создатели языков высокого уровня сделали одно ошибочное предположение: будто бы каждый пользователь, который придерживается некоторого стандарта, предпочтет автоматически, так сказать, ему следовать. Озабоченные задачей обеспечения легкости программирования, разработчики языков забыли о некоторых необходимых возможностях. ФОРТРАН и Алгол слишком абстрактны для системного программирования; это проблемно-ориентированные языки, используемые для решения проблем инженерных областей, науки и бизнеса. Программисты системного уровня по-прежнему были вынуждены пользоваться машинным языком ассемблера.

Реагируя на такую ситуацию, некоторые разработчики системного обеспечения сделали шаг назад — или, придерживаясь иерархии, спустились ниже — и создали класс машинно-ориентированных языков. Как видно из генеалогии языка С, BPCL и В входят в этот класс программных средств очень низкого уровня. Эти языки превосходно подходят к конкретной ЭВМ, но не пригодны для чего-нибудь еще; они слишком тесно связаны с определенной архитектурой.Язык С находится чуть выше машинно-ориентированных языков, но все же ниже, чем большинство проблемных языков. Язык С достаточно близок с компьютеру и дает возможность детально описывать реализацию некоторого приложения, при этом он достаточно абстрактен и позволяет игнорировать нюансы аппаратного обеспечения. Вот почему С рассматривается одновременно как язык высокого и низкого уровней.

### Достоинства языка С.

Все машинные языки имеют определенный внешний вид. APL выглядит как набор иероглифов, ассемблер напоминает колонки мнемонических обозначений. Паскаль имеет легко читаемый синтаксис. Наконец появляется С. При первом знакомстве многие программисты сочтут его синтаксис зашифрованным и пугающим. В С очень мало дружественных синтаксических конструкций естественного языка, имеющихся во многих других языках программирования. Вместо них системный программист получает необычно выглядящие операторы и множество указателей. Затем новичок языка С открывает для себя различные языковые средства, корни которых уходят к аппаратно-программным истокам С. В последующих разделах рассматриваются достоинства С.

Малый размер.

В С меньше синтаксических правил, чем во многих других языках, и можно написать первоклассный компилятор С, работающий всего в 256 Кб памяти. По сути, в С больше операций и их комбинаций, чем ключевых слов.

Набор команд языка.

Изначально язык С, написанный ДэннисомРитчи, имел всего 27 ключевых слов. Несколько зарезервированных слов было добавлено в стандарте ANSI С. В MicrosoftC/C++ добавлено еще 19 команд, в результате чего в этой версии число ключевых слов стало равным 66.

Многие функции, обычно описываемые в других языках программирования как их часть, не включены в С. Например, в С нет встроенных средств ввода и вывода, он не содержит арифметических операций (кроме базовых: сложения и вычитания) или строковых функций. Поскольку мало пользы от языка, не имеющего таких средств, в языке С имеется богатый набор библиотечных функций для ввода-вывода, арифметических и строковых операций. Этот общепринятый набор библиотек настолько широко используется, что он может практически рассматриваться как часть самого языка. Однако, одно из достоинств С — его свободная структура, позволяющая легко записывать все эти функции.

Быстродействие.

Код, создаваемый большинством компиляторов С, весьма эффективен. Компактный язык, небольшая исполняющая система и тот факт, что язык С близок к аппаратным средствам, — все это позволяет программам на С работать со скоростью, близкой к быстродействию эквивалентных программ, написанных на ассемблере.

Язык со слабой типизацией.

В отличие от Паскаля, языка с сильной типизацией, С рассматривает типы данных менее строго. Эта черта унаследована от языка В, который также свободно манипулировал с типами данных. Благодаря этому, данные можно рассматривать с разных сторон. Например, в одной точке программы переменную можно рассматривать как символ, а в другой — для преобразования символа в верхний регистр (путем вычитания 32) — эту же ячейку памяти можно рассматривать как ASCI-код символа.

Структурированный язык.

В С имеются все управляющие конструкции, которые присущи современному языку. Это произведет впечатление, если вспомнить, что в 1971 году еще только зарождавшийся язык С предвосхитил появление формального структурного программирования. В языке реализованы: оператор for — для циклов, конструкции if и if-else, оператор-переключатель (switch) и циклы while. ВС также обеспечивается, благодаря управлению их видимостью, изолированность кода и данных друг от друга. Например, для этих целей в С имеются локальные переменные и вызов по значению для обеспечения целостности собственных данных подпрограммы.

Поддержка модульного программирования.

В С поддерживается концепция модульного программирования, заключающаяся в раздельной компиляции и компоновке. При этом можно перекомпилировать только те части программы, которые менялись при проектировании. Эта возможность чрезвычайно важна при разработке больших программ или даже при создании программ среднего размера на медленных системах. При отсутствии поддержки модульного программирования затраты времени, необходимые для компиляции всей программы, могут непозволительно удлинить цикл ее изменения, компиляции, проверки и коррекции.

Простой интерфейс с ассемблерными подпрограммами.

В большинстве компиляторов С существует хорошо описанный метод вызова ассемблерных подпрограмм. С учетом раздельной компиляции и компоновки, эта возможность делает язык С очень сильным инструментом для приложений, требующих совмещения программ высокого уровня и ассемблерных программ. На многих системах процедуры, написанные на С, можно также интегрировать в ассемблерные программы.

Поразрядная обработка.

Зачастую при системном программировании необходимо манипулировать с объектами на уровне двоичных разрядов. Естественно, что язык С, связанный происхождением с операционной системой UNIX, располагает большим набором команд поразрядной обработки.

Переменные-указатели.

Одним из требований, предъявляемых к языку операционной системой, является возможность адресации определенной области памяти. Эта возможность также ускоряет выполнение программы. В языке С для этих целей служат указатели. Хотя указатели имеются и в других языках, С отмечен за возможность выполнять арифметические действия с указателями. Например, если переменная index указывает на первый элемент массива student\_records, значение index+1 будет адресом второго элемента массива student\_records.

Гибкие структуры.

Все массивы в С — одномерные. Многомерные структуры создаются на базе этих одномерных массивов. Массивы и структуры (записи) могут объединяться любым необходимым образом, образуя базы данных, ограничиваемые только возможностями программиста.

Эффективность использования памяти.

Во многом по тем же причинам, по которым программы С работают быстро, эти же программы очень эффективно используют память. Отсутствие утроенных функций позволяет отказаться в программах от тех функций, которые не требуются в данном приложении.

Переносимость.

Переносимость — это возможность переноса некоторой программы, работающей на одном компьютере или в одной операционной системе, на другой компьютер или в другую операционную систему. Программы, написанные на С, являются самыми мобильными в современном компьютерном мире. В особенности, этоотносится к мини- и микрокомпьютера.

Библиотеки специальных функций.

Для всех популярных компиляторов С имеется множество коммерчески доступных библиотек функций. Имеются библиотеки функций для графики, работы с файлами, поддержки баз данных, экранных окон, ввода данных, коммуникаций и общего назначения. Используя эти библиотеки, можно значительно сократить время проектирования.

### Недостатки языка С.

Не существует идеальных языков программирования. Разные программные проблемы требуют различных решений. Дело программиста — выбрать наилучший язык для конкретного проекта. В любом проекте — это одно из первых необходимых для принятия решений, и его практически не изменить после начала работы над программой. Выбор языка программирования может также непосредственно повлиять на успех или неуспех всего проекта. Далее описываются некоторые недостатки языка С, что позволит лучше понять, когда стоит, а когда не стоит использовать С для конкретного приложения.

Слабая типизация.

Тот факт, что в С слабая типизация, является и достоинством языка, и одним из его недостатков. На практике, типизация определяет, в какой степени в данном языке используются типы переменных. (Например, целые числа и числа с плавающей запятой точкой относятся к двум различным типам числовых данных.) В некоторых языках запрещается присваивать значения одного типа переменным другого типа, не используя некоторой функции преобразования. При этом данные защищены от искажений, вызываемых неожиданными округлениями.

Как указывалось выше, в С можно присвоить целое значение символьной переменной и наоборот. Для программиста это означает необходимость следить за правильным использованием переменных. Для опытных разработчиков это не представляет проблемы. Однако, новички предпочли бы, чтобы им напоминали об этом источнике возможных побочных эффектов. Побочный эффект в языке — это неожиданное изменение значения переменной или другого объекта. Поскольку в С слабая типизация, этот язык предоставляет большую гибкость в манипулировании данными. К примеру, операция присваивания (=) может появиться несколько раз в одном и том же выражении. Эта гибкость, которую вы можете использовать себе на пользу, означает, что можно записать выражения, не имеющие ясного и определенного значения. Ограничения, наложенные на использование операции присваивания и ей подобных, или устранение всех побочных эффектов и непредсказуемых результатов лишили бы язык С многих его достоинств и качеств ассемблерного языка высокого уровня.

Отсутствие проверок на этапе исполнения.

Поскольку в исполняющей системе С отсутствуют проверки, это может привести к тому, что многие загадочные и неустойчивые ошибки останутся необнаруженными. Например, исполняющая система не предупредит вас о том, что в вашей программе индекс массива превышает его размер. Это — расплата за скорость компилятора, во имя быстродействия и эффективности приложения.

### Использование языка Си.

Си — язык «компилируемого» типа. Не огорчайтесь, если это звучит для вас пока как непонятный набор слов; вы поймете, что это значит, когда мы опишем этапы процесса создания работающей Си-программы.

_02_01

Если вы привыкли использовать какой-нибудь язык программирования компилируемого типа, например, Паскаль или Фортран, вам будут понятны основные этапы «сборки» программ, написанных на Си. Но если ваш опыт основан на работе с такими языками интерпретируемого типа, как Бейсик и Лого, или у вас совсем нет соответствующей подготовки, то процесс сборки может показаться вам поначалу необычным. К счастью, мы можем объяснить вам все детали этого процесса, и вы увидите, что на самом деле он достаточно понятен и прост.

Чтобы дать вам первое представление о процессе создания программы, ниже приводится упрощенная схема того, что необходимо сделать — начиная от написания программы и кончая ее выполнением.

1. Используйте «редактор текстов» для создания программы на языке Си.
2. Попытайтесь осуществить трансляцию вашей программы с помощью удобного для вас компилятора. Он проведет проверку правильности вашей программы и, если обнаружит ошибки, выдаст сообщение об этом. В противном случае компилятор выполнит перевод программы в некоторый внутренний язык ЭВМ и поместит результат в новый файл.
3. Набрав имя этого нового файла на клавиатуре дисплея, вы можете запустить вашу программу.

В некоторых вычислительных системах второй этап может быть разбитым на два или три шага, но его суть от этого не изменится. Давайте рассмотрим теперь каждый этап более подробно.

### Будущее языка Си.

Язык Си уже занимает доминирующее положение в мире мини-компьютеров, работающих под управлением ОС UNIX. Сейчас он распространяется на область персональных ЭВМ. Многие фирмы, производящие программное обеспечение, все чаще обращаются к Си, как к удобному языку для реализации своих проектов: программ обработки текстов, составления крупноформатных таблиц, компиляторов и т. д., поскольку известно, что Си позволяет получить компактные и эффективные программы. Что еще важнее, эти программы могут быть легко модифицированы и адаптированы к новым моделям ЭВМ.

Другой причиной, способствующей проникновению Си в программное обеспечение персональных ЭВМ, является желание пользователей, работающих в системе UNIX, отлаживать свои программы дома. Поскольку уже созданы варианты компилятора с языка Си для некоторых моделей персональных ЭВМ, пользователи имеют возможность это делать.

По нашему мнению, то, что подходит для фирм и программистов с большим опытом работы на Си, хорошо и для остальных пользователей. Все больше и больше программистов останавливают свой выбор на языке Си, чтобы воспользоваться его преимуществами. Поэтому вам совсем не обязательно быть, профессиональным программистом, чтобы следовать их примеру.

Короче говоря, Си суждено стать одним из наиболее важных языков программирования в 80—90-е годы. Он уже применяется на мини-компьютерах и персональных ЭВМ. Он используется фирмами, производящими программное обеспечение, студентами, обучающимися программированию, и различными энтузиастами. И если вы хотите работать в сфере программотехники, то один из первых вопросов, накоторый вы должны будете отвечать «да», — «Умеете ли вы программировать на Си?».

## Исходные файлы и выполняемые файлы.

Наша замечательная программа, несмотря на свою лаконичность и простоту, для компьютера является совершенно бессмысленным набором символов, так как он «не понимает» директив типа #include или printf. Он понимает только специальный язык, называемый машинным кодом, — набор последовательностей двоичных цифр, например, 10010101 и 01101001. Если мы хотим, чтобы компьютер выполнил программу, мы должны осуществить перевод (трансляцию) нашего кода (исходного) в ее код (машинный). В результате этих действий будет получен выполняемый файл 2), т. е. файл, который содержит весь необходимый машинный код, требующийся компьютеру для выполнения задания.

02_02

Если вышеприведенные рассуждения выглядят скучными и непонятными, не огорчайтесь. Дело в том, что процесс перевода удалось переложить на сам компьютер! «Умные» программы, называемые компиляторами, выполняют весь объем работы, связанный с этим переводом. Детали процесса зависят от особенностей конкретной системы. Ниже кратко описано несколько способов перевода.

Описанное ниже разбиение процесса компиляции программы на последовательные шаги зависит как от операционной системы, так и от самого компилятора. Конкретный пример, который мы здесь рассматриваем, — это функционирование компилятора Microsoft С под управлением операционной системы(компилятор LatticeQ лежащий в основе версии, реализованной фирмой Microsoft, запускается по аналогичным правилам, только вместо команд mc1 и mс2 необходимо использовать команды lс1 lс2)

Так же как и прежде, мы считаем, что исходная программа содержится в файле inform.с. Наша первая команда выглядит следующим образом:

mc1inform

(Компилятор интерпретирует строку символов inform как inform.с). Если ошибок нет, то в результате будет получен промежуточный файл с именем inform.q. Затем мы набираем на клавиатуре следующую команду:

mc2inform,

в результате выполнения которой будет создан файл с именем inform.obj, содержащий так называемый «объектный код» (код на языке машины), соответствующий нашей исходной программе. (Объяснения приведены ниже). После этого вводится команда

link с inform,

по завершении которой создается файл inform.exe. Наша цель достигнута — получен файл, содержащий выполняемую программу. Если мы затем введем команду

inform.exe или просто

inform, то наша программа начнет выполняться.

На самом деле вы можете не знать, что происходит, когда вы пользуетесь вышеописанной процедурой, но, если вам интересно, мы кратко опишем выполняемые при этом действия.

Что здесь нового? Во-первых, новым является то, что вводится файл с именем inform.obj. Поскольку в нем содержится машинный код, непонятно, почему мы не остановились в этом месте? Ответом может служить то, что полная программа включает в себя части, которые мы не писали. Например, мы использовали команду printf, являющуюся программой, помещенной в Си-библиотеку. Вообще говоря, может возникать необходимость использовать в программе стандартные процедуры, помещенные в различные библиотеки. Эта потребность приводит к использованию второго нового понятия — команды link.

Программа link является частью операционной системы IBMDOS. Она связывает наш объектный код (находящийся в файле inform.obj) с некоторыми стандартными процедурами, содержащимися в файле с.obj, и, кроме того, осуществляет поиск требуемых объектных модулей в той библиотеке, которую мы указываем (программа link запрашивает требуемое имя во время выполнения); в данном случае это будет библиотека с именем Ic.lib. Затем указанная программа объединяет все найденные модули в одну полную программу.

_02_04

Программа cc, работающая под управлением ОС UNIX, во время выполнения проходит аналогичную последовательность шагов; отличие состоит только в том, что она «скрывает» этот факт от нас, уничтожая файл с объектным модулем после его использования для получения полной программы. (Но в случае необходимости в ответ на соответствующий запрос компилятор выдаст нам объектный файл под именем inform.о).

## Принципы программирования.

Громадный диапазон возможностей языка С — от поразрядной обработки до высокоуровневого форматируемого ввода/вывода — и его относительная похожесть на разных компьютерах обеспечили его использование для создания приложений в науке, инженерных областях и бизнесе. Это также объясняется широким распространением операционной системы UNIX на компьютерах всех типов и размеров.

Однако, подобно любому другому мощному инструменту, язык С налагает большую ответственность на пользователей. Программисты на С должны очень быстро освоить принципы языка, принять различные правила и соглашения, чтобы их программы были понятны как им самим (спустя некоторое время после их написания), так и остальным, тем кто впервые знакомится с программой. В языке С дисциплина программирования весьма важна. К счастью, она приходит почти автоматически с опытом проектирования.

## Стандарт ANSI С.

Комитет Американского института национальных стандартов (ANSI) разработал стандарты для языка С. В этом разделе описываются некоторые существенные изменения, предложенные и осуществленные данным комитетом. Некоторые из изменений направлены на увеличение гибкости языка, другие являются попытками стандартизировать те языковые средства, которые ранее оставлялись на рассмотрение разработчика компилятора.

Ранее, единственным стандартом была книга Кернигана и Ритчи "Язык программирования С" (В. Kernighan, D. Ritchie — "The С ProgrammingLanguage", Prentice-Hall, MurrayHill, NewJersey, 1988). В этой книге не были подробно рассмотрены некоторые особенности языка, что привело к расхождениям в реализациях компиляторов. В стандарте ANSI сделана попытка снять эти неоднозначности. Хотя некоторые предложенные изменения могут создать проблемы в некоторых написанных ранее программах, они не влияют на большинство существующих программ.

Стандарт ANSI С обеспечивает даже большую, чем была ранее, возможность написания переносимого С-кода. Однако, в стандарте устранены не все неоднозначности языка, и, поскольку С активно взаимодействует с аппаратным обеспечением компьютеров, многие программы всегда требуют некоторой доработки при переносе в иную среду. Разработавший стандарт комитет ANSI принял в качестве директив некоторые постулаты, которые коллективно были названы "духом языка С". Вот некоторые из них:

* Доверять программисту
* Не мешать программисту делать то, что ему нужно
* Поддерживать компактность и простоту языка

Кроме того, были проведены согласования с международным сообществом, призванные обеспечить соответствие ANSI (американского) стандарта С и стандарта ISO (Международной организации стандартов). Благодаря этим усилиям, С — единственный язык, реально имеющий альтернативные таблицы для сортировки, множество символьных таблиц, и учитывающий различные национальные особенности. В таблице перечислены только некоторые вопросы, затронутые комитетом ANSI.

Таблица. Свойства языка С, стандартизованные комитетом ANSI

|  |  |
| --- | --- |
| **Свойство** | **По стандарту** |
| Типы данных | Четыре: символьный, целый, с плавающей запятой и перечисляемый |
| Комментарии | /\* — начало комментария, \*/ — окончание; альтернативный вариант — //, остаток строки, следующий за этим символом, компилятор игнорирует |
| Длина идентификатора | 31 символ этого достаточно для обеспечения уникальности идентификатора |
| Стандартные идентификаторы и заголовки файлов | Принят минимальный набор идентификаторов и заголовочных файлов, необходимый для выполнения основных операций, например, для ввода/вывода |
| Операторы препроцессора | Директива препроцессора # может иметь перед собой разделительные символы (любая комбинация пробелов и символов табуляции), позволяющие выделять в абзацы команды препроцессора в целях наглядности. В некоторых ранних версиях компиляторов требовалось, чтобы все директивы препроцессора начинались в первой колонке. |
| Новые директивы | Добавлены две новые директивы препроцессора: выражение # ifdefined и #elif |
| Смежные строки | Смежные строки литералов должны объединяться. Это позволяет, к примеру, продолжить директиву #define на следующих строках. |
| Стандартные библиотеки | Базовый набор системных и внешних процедур: например, read() и write() |
| Управление выводом | Принятый набор управляющих (escape) кодов, представляющих собой коды управления форматированием, например, символы новой строки, новой страницы и табуляции. |
| Ключевые слова | Принятый минимальный набор слов, использующихся для построения правильных операторов Си |
| Sizeof() | Функция Sizeof() возвращает значения типа size\_t, а не системно-зависимое значение размера integer |
| Прототипы | Все компиляторы СИ должны работать с программами, в которых используются или не используются прототипы. |
| Аргументы командной строки | Для того, чтобы компилятор правильно обрабатывал аргументы командной строки, разработан особый синтаксис. |
| Указатель типа void | Ключевое слово void может использоваться с функциями, не возвращающими значения. Функция, возвращающая значение, может иметь описание void, сообщающее компилятору о том, что тип возвращаемого значения преднамеренно игнорируется. |
| Обработка структур | Обработка структур значительно улучшена. Имена элементов структуры и описания объединений могут не быть уникальными. Структуры могут передаваться в качестве аргументов функций, возвращаться функциями и присваиваться структурам того же типа. |
| Объявления функций | Объявления функций могут включать в список типов аргументов (прототипы функции) для передачи компилятору сведений о количестве и типе аргументов. |
| Шестнадцатеричные символьные константы | Шестнадцатеричные константы могут задаваться при помощи начального символа \x, за которым следует от одного до трех шестнадцатеричных цифр (0-9, a-f, A-F); например, десятичное 16=\x10, что может быть записано как 0x10, если использовать традиционную нотацию Си. |
| Триплеты | Триплеты (триграммы) определяют последовательности стандартных символов, которые могут изображаться не на всех клавиатурах. Например, ??<можно заменить на более изящный символ {}. |

## Эволюция языка C++ и объектно-ориентированное программирование.

Попросту говоря, C++ — это надмножество языка С.В (языке) C++ сохранены все достоинства С, включая его мощные и гибкие средства для работы с аппаратурой, для возможности низкоуровневого системного программирования, эффективность, экономность и богатые выразительные возможности. Но кроме этого, C++ вывел С в динамичный мир объектно-ориентированного программирования и сделал его платформой для высокоуровневых проблемных абстракций, в чем он даже превзошел язык Ада. Все эти возможности C++ сочетаются с простотой; поддерживая модульнoе программирование, как в языке Модула-2, C++ сохраняет компактность и эффективность исполняемого кода языка С.

Этот новый гибридный язык объединяет конструкции обычного процедурного языка, хорошо знакомые многим программистам, и объектно-ориентированную модель, которую вы можете использовать в полной мере для получения чисто объектно-ориентированного решения некоторой проблемы. (На практике, приложение C++ может проявлять этот дуализм, совмещая в себе модели процедурного программирования и новейшего объектно-ориентированного.) Такая двуликость языка создает особые проблемы для начинающего программиста на C++: необходимо не только изучать новый язык, но нужен также новый стиль мышления и решения проблем.

## История C++.

Не удивительно, что C++ имеет те же корни, что язык С. Напоминая в чем-то BCPL и Алгол 68, C++ также имеет компоненты языка Симула 67. В C++ есть свойства Алгола 68: возможность перегружать операции и объявлять переменные в том месте, где они впервые используются в программе. Концепция подклассов (или производных классов) и виртуальных функций взята из языка Симула 67. Подобно многим другим популярных языкам программирования, C++ представляет собой результат эволюции и накопления некоторых наилучших свойств предыдущих языков. Но безусловно ближе всего он к языку С.

Заслуга разработки в начале 1980-х годов языка C++ принадлежит Бъярну Страуструпу(BjarneStroustrup) из BellLabs. (Честь названия этого нового языка доктор Страуструп уступил РикуМаскити — RickMascitti.) Первоначально C++ был разработан для решения некоторых задач событийного моделирования, в которых соображения эффективности не позволили использовать другие языки. В 1983 году C++ был впервые использован за пределами группы программистов доктора Страуструпа, но до лета 1987 года язык все еще находился на стадии естественной эволюции и уточнений. Одной из ключевых задач в C++ было обеспечение совместимости с языком С. Было желание сохранить в целости миллионы ранее написанных и отлаженных строк программ языка С, многие существующие библиотеки С и полезные инструментальные средства, написанные ранее на С. Поскольку эта задача была в значительной мере успешно решена, многие программисты считают переход на C++ значительно более простым, чем совершенный ими ранее переход с других языков, например с ФОРТРАНа, на С.

C++ в полной мере обеспечивает разработку программ. Поскольку в нем усилена проверка типов, невозможными стали многие побочные эффекты, возникающие при написании слабо типизированных приложений С. Самым главным достоинством языка C++ является поддержка объектно-ориентированного программирования (ООП). Для того чтобы использовать все преимущества C++, необходимо пересмотреть подход к решению проблем. Например, нужно описывать объекты и связанные с ними операции, а также создавать все необходимые классы и подклассы.

### Использование объектов C++ для быстрого создания программы.

Далее показан пример, в котором видны преимущества абстрактного объекта данных языка C++ перед ограниченными встроенными конструкциями и возможностями устаревшего языка. К примеру, пусть программисту на ФОРТРАНе потребовался список сотрудников. Его можно реализовать при помощи многочисленных массивов скалярных данных, каждый из которых представляет из себя набор сведений о сотруднике. Все массивы необходимо объединить общим индексом. Если для каждого сотрудника выделить десять полей информации, то для получения массива записей понадобится десять обращений, использующих один и тот же индекс. Решение на C++ включает в себя следующее: объявление простого объекта *employee\_database*(база данных сотрудников), который может получать сообщения типа *add\_employee*(добавить сотрудника), *delete\_employee*(удалить сотрудника), *access\_employee*(изменить сотрудника) или *display\_employее*(показать сотрудника), касающиеся информации, содержащейся в объекте. Манипулировать с объектом *employee\_database* можно в понятиях естественного языка. Для добавления новой записи в *employee\_database* достаточно указать следующее: employee\_database.add\_employee(new\_recruit)

Предполагается, что объект *employee\_database* был соответствующим образом объявлен, функция add\_employee() — это метод, должным образом описанный в классе, поддерживающем объекты *employee\_database,* а параметр *new\_recruit*(новоприбывший) — конкретная добавляемая информация. Обратите внимание на то, что класс объектов, названный *employee\_database,* не является частью самого языка. Вместо этого программист расширяет язык для решения конкретной проблемы. Описывая новый класс объектов или модифицируя существующие классы (создавая подклассы), можно более естественно перейти из проблемной области в программную область (или в область решений). Главной задачей является освоение этого мощного средства.

### Некоторые усовершенствования по сравнению с языком С.

В следующем разделе описаны небольшие изменения (не объектно-ориентированные) по сравнению с языком С.

Комментарии.

В C++ введен символ комментарии//, действующий до конца строки. Однако, по-прежнему можно использовать скобки комментария/\* и \*/.

Имена перечисляемых типов.

Имя перечисления является именем типа. Это упрощает нотацию, поскольку не нужно помещать описатель enum перед именем перечисляемого типа.

Имена структуры или класса.

Имя структуры или класса является именем типа. Такая возможность отсутствует в С. В C++ описатели struct или classперед названием структуры или класса использовать не обязательно.

Блочные объявления.

В C++ допускаются объявления внутри блоков и после программных операторов. Это позволяет объявить некоторый идентификатор в том месте, где он впервые используется в программе. Можно даже объявить управляющую переменную цикла внутри формального описания управляющей структуры, как показано ниже:

// объявление переменной С++ на месте ее использования

for (int row; row<MAX\_ROWS; row++)

Операция уточнения области действия (scope).

Для разрешения конфликтов имен используется новая операция уточнения области действия (::).К примеру, если некоторая функция имеет локально объявленную переменную *vector\_location,* и существует глобальная переменная *vector\_location,* описатель ::*vector\_location* позволяет обращаться к этой глобальной переменной из локальной области действия функции. Обратное — невозможно.

Описатель const.

Можно использовать описатель const для определения фиксированного значения некоторой сущности в пределах области ее действия. Также этот описатель можно использовать для данных, на которые указывает некоторая переменная-указатель, для значения адреса указателя или одновременно—и для адреса указателя и для данных, на которые он указывает.

Анонимные объединения.

Объединения без имени можно описывать в любом месте, где допускается описание переменной или элемента структуры. Вы можете использовать эту возможность для экономии памяти, позволяя двум или более элементам структуры использовать память совместно.

Явное преобразование типов.

Можно использовать название предопределенного или описанного пользователем типа в качестве функции для преобразования данных из одного типа в другой. При определенных условиях, подобное явное преобразование типов может использоваться как альтернатива операции приведения типов.

Объявления функций.

Многие программисты, ранее писавшие на языках Паскаль, Модула-2 и Ада, могут порадоваться тому, что в C++ можно указывать имя и тип каждого параметра функции внутри скобок после названия функции. Например:

void \* dupmem(void \*dest,intc,unsignedcount)

{

}

Эквивалентная запись на С в стандарте ANSI выглядит точно так же. В данном случае C++ повлиял на разработку стандарта ANSI.Транслятор C++ будет выполнять проверку типа, контролируя наличие совпадения количества и типа переменных, переданных в функцию, и количества и типа формальных параметров, объявленных для этой функции.

Также выполняется проверка, гарантирующая, что тип, возвращаемый функцией, совпадает с типом переменной, использованной в выражении, вызывающем эту функцию. Подобная проверка типа параметров отсутствует в большинстве систем С.

Перегруженные функции.

При использовании операции перегрузки в C++ имя функции может оставаться неизменным, а однозначно распознаваться каждая перегруженная функция может по количеству и типу ее параметров.

Значения параметров функций по умолчанию.

Можно указывать значения по умолчанию для последовательного списка параметров функции C++. В этом случае функция может вызываться с меньшим числом параметров. Любой отсутствующий последующий параметр принимает значение по умолчанию.

Функции с неуказанным числом параметров.

Можно описывать функции C++, не указывая число и тип параметров, и вместо них записывается многоточие (...). При этом проверка типа параметров отключается, что позволяет гибко использовать функцию.

Ссылочные параметры функции.

При помощи операции определения адреса (&) формальный параметр функции может быть объявлен как параметр-ссылка. Например:

int i;

increment(i);

void increment(int&variable\_reference)

{

variable\_reference++;

}

Так как параметр *&variable\_reference* описан как ссылочный, при вызове функции increment() его адресу присваивается адрес переменной i. Значение i, переданное в функцию, увеличивается на единицу внутри функции increment() и возвращается переменной i вне функции increment(). Нет необходимости явно передавать адрес i в функцию increment(), как это было в С.

Операторы new и delete.

Операторы new и delete, появившиеся в C++, позволяют программисту выделять и освобождать динамически распределяемую область памяти.

Указатели void и функции, возвращающие void.

В C++ тип void используется для функций, не возвращающих значений. Можно объявить переменные-указатели, имеющие значение void. Затем такие указатели могут принимать значения других указателей, ссылающихся на произвольный базовый тип.

### Основные усовершенствования по сравнению с языком С.(часто повторяется)

Самое существенное улучшение по сравнению с языком С касается концепции объектно-ориентированного программирования. В следующих разделах кратко описаны все характеристики C++, позволяющие реализовать эту концепцию.

Конструкторы классов и инкапсуляция данных.

Конструкторы классов — это важнейший инструмент объектно-ориентированного программирования. Описание класса может инкапсулировать все объявления данных, начальные значения и набор операций (называемых методами), что обеспечивает абстрагированность данных. Могут объявляться объекты данного класса, этим объектам могут передаваться сообщения. Кроме того, каждый объект класса может иметь как собственные частные для этого класса, так и общие данные.

Класс struct.

Структура struct в C++ является подмножеством описания класса и не имеет разделов private и protected. Этот подкласс может содержать и данные (как это предусмотрено в ANSI С), и функции.

Конструкторы и деструкторы.

Использование конструкторов и деструкторов гарантирует инициализацию данных, описанных внутри объекта заданного класса. При объявлении объекта активизируется соответствующий инициирующий конструктор. Деструкторы автоматически освобождают память, занятую объектом, при выходе из объявленной области действия этого объекта.

Сообщения.

Как вы уже видели, объект — это основной элемент объектно-ориентированного программирования. Для манипулирования объектами используется передача сообщений. Для передачи объектам сообщений (объявленных переменных данного класса) используется механизм, аналогичный вызову функции. Набор сообщений, допустимых для передачи объекту, указывается в описании класса данного объекта. Каждый объект реагирует на сообщение, выполняя соответствующие действия, определяемые смыслом сообщения. Например, если имеется объект Palette\_CoIors и метод *SetNumColors\_Method,* имеющий единственный целочисленный параметр, послать сообщение объекту можно при помощи следующего оператора Palette\_Colors.SetNumColors\_Method(16) ;

"Дружественные" классы.

Концепция скрытности и инкапсуляции данных подразумевает, что доступ к внутренней структуре, образующей некий объект, запрещен. Частная область класса обычно полностью изолирована от всех функций, находящихся вне его. В C++ возможно, все-таки, объявлять другие функции, не входящие в число методов, или классы "дружественными" данному классу. Такие отношения ослабляют обычную закрытость класса и позволяют обращаться к его частным данным и методам.

Перегрузка операций.

В C++ программист может придать множество значений набору предопределенных операций и функций, поставляемых с компилятором или определенных пользователем. К примеру, различные функции обычно имеют разные имена, однако, для функций, выполняющих аналогичные действия с разными типами объектов, иногда удобнее иметь одно и то же имя. Когда типы их аргументов меняются, компилятор может распознать их и вызвать нужную функцию. Далее следует пример программы: можно иметь одну функцию average(), перегружаемую для массивов с элементами типа integer, float и double.

int average(intisize, intiarray[]);

float average (intisize, float farray[]);

double average(intisize, double darray[]);

Поскольку объявлены три различных функции с одним и тем же именем, компилятор может анализировать вызывающий оператор и автоматически выбирать функцию, соответствующую списку формальных параметров:

average(isize,iarray);

average(isize,farray);

average(isize,darray);

Производные классы.

Производный класс можно рассматривать как подкласс заданного класса, и создавать, таким образом, иерархию абстрактных сущностей. Объекты производного класса обычно наследуют все или некоторые методы родительского класса. Зачастую эти унаследованные методы дополняются в производном классе новыми методами, присущими этому классу. Все объекты подкласса содержат наборы данных родительского класса, а также свои собственные данные.

Полиморфизм при использовании виртуальных функций.

Полиморфизм характеризует древовидную структуру родительских классов и их подклассов. Каждый подкласс в этой структуре может получать одно или несколько сообщений с одним и тем же названием. Когда некоторый объект класса, принадлежащего этому дереву, получает сообщение, он определяет конкретный смысл этого сообщения, соответствующий объекту данного подкласса.

Библиотеки потоков.

В язык C++ включены дополнительные библиотечные (предопределенные) потоки. Для терминального и файлового ввода/вывода предусмотрены три класса: cin, cout и сеrr. Все операции внутри этих классов могут перегружаться в классе, описанном пользователем. Эта возможность позволяет легко настроить операции ввода и вывода согласно требованиям приложения.

## Базовые элементы программы на С.

Возможно, вам знакомо мнение, что язык С трудно изучать. Хотя можно согласиться с тем, что поверхностное знакомство с программой на С заставит вас почесать голову, но это только потому, что С имеет непривычный синтаксис, структуру и принципы структурированного расположения текста. К концу этой главы вы получите достаточно информации по практическому использованию языка С, что позволит вам написать короткую, но осмысленную программу. (В следующем разделе вы познакомитесь с пятью основными компонентами "хорошей" программы.)

### Пять основных компонентов программы.

Вы наверно знакомы с методом решения проблем, называемым IPO-диаграммой. IPO-диаграммы — это формализованный подход к решению старой программистской задачи ввода/обработки/вывода (Input/Process/Output). Следующий далее список получен из трех этих основных сущностей и включает в себя весь цикл разработки прикладной программы. Все программы должны содержать пять базовых компонентов:

* Программы должны получать информацию от некоторого источника ввода
* Программы должны определять способы форматирования введенной информации и ее хранения.
* Программы используют некоторый набор команд для обработки, введенной информации.
* Эти команды можно разбить на четыре основных категории: простые операторы, условные операторы, циклы и подпрограммы.
* Программы должны выдавать результаты обработки данных.

Хорошо написанная программа включает в себя все перечисленные выше компоненты, что предполагает наличие следующих признаков: хорошей модульной организации, самодокументированного текста программы (осмысленные имена переменных) и хорошо структурированного расположения текста

### Различия между программами в ЕХЕ - и СОМ - файлах

Несмотря на то, что существуют служебные программы, например EXE2BIN, которые преобразует ЕХЕ -файл в СОМ -файл, однако нужно знать определенные различия между программой, выполняемой как ЕХЕ -файл и программой, выполняемой как СОМ-файл.

**1. Размер программы.**

Программа в формате ЕХЕ может иметь любой размер, в то время как СОМ-файл ограничен размером одного сегмента и не превышает 64 К. Размер СОМ-файла всегда меньше, чем размер соответствующего ЕХЕ-файла; одна из причин этого - отсутствие в СОМ-файле 512-байтового заголовка ЕХЕ-файла (префикса программного сегмента, PSP).

**2. Сегмент стека.**

В ЕХЕ-программе определяется сегмент стека, в то время как СОМ-программа генерирует стек автоматически. Таким образом, при создании ассемблерной программы, которая будет преобразована в СОМ-файл, стек должен быть опущен.

**3. Сегмент данных.**

В ЕХЕ- программе обычно определяется сегмент данных, а регистр DS инициализируется адресом этого сегмента. В СОM-программе все данные должны быть определены в сегменте кода.

**4. Инициализация.**

В ЕХЕ-программе выполняются запись нулевого слова в стек и инициализация регистра DS. Так как в СОМ-программе стек и сегмент данных не определены, то эти шаги отсутствуют. Когда СОМ-программа начинает работать, все сегментные регистры содержат адрес префикса программного сегмента (PSP) - 256-байтовый (100Н) блок, который резервируется операционной системой DOS непосредственно перед СОМ- или ЕХЕ-программой в памяти. Так как адресация начинается со смещения 100Н от начала PSP, то в программе после оператора SEGMENT кодируется директива ORG 100Н.

**5. Обработка.**

Для программ в ЕХЕ- и СОМ-форматах выполняются:

*a) ассемблирование для получения OBJ-файла:*

Для TASM:

tasm /z /zi /l prog.asm

кроме файла: prog.obj дополнительно создается:

* файл.lst (листинг программы (опция /l);
* выводится исходный текст вместе с сообщением об ошибках (опция /z);
* дополнительная информация для дебагера(опция /zi).

*b) компоновка для получения ЕХЕ-файла:*

Для TLINK:

tlink /v prog.obj

Создает:

* файлы: prog.exe и prog.map
* дополнительную информацию для дебагера (опция /v).

Если программа создается для выполнения как ЕХЕ-файл, то ее уже можно выполнить.

Если же программа создается для выполнения как СОМ-файл, то компоновщиком будет выдано сообщение:

Warning: No STACK Segment\*

Это сообщение можно игнорировать, так как определение стека в программе не предполагалось.

**Пример программы типа СОМ**

Программа PROG, приведенная в листинге, написана согласно требованиям СОМ-формата. Обратите внимание на следующие особенности в этой СОМ-программе:

* Сегмент стека и сегмент данных отсутствуют.
* Директива ORG служит для резервирования 100H байт от начального адреса под PSP.
* Директива ORG 100H устанавливает относительный адрес для начала выполнения программы. Программный загрузчик устанавливает этот адрес в командном указателе.
* Команда JMP служит для обхода данных, определенных в программе.

Размеры ЕХЕ- и СОМ-программ - 788 и 20 байт соответственно. Учитывая такую эффективность СОМ-файлов, рекомендуется все небольшие программы создавать в СОМ-формате.

Некоторые программисты кодируют элементы данных после команд так, что первая команда JMP не требуется. Кодирование элементов данных перед командами позволяет ускорить процесс ассемблирования и является хорошим стилем, рекомендуемым в руководствах по ассемблеру.

Пример СОМ-программы

;Программа Prog

;Программа складывает два числа и завершается

.MODEL TINY ; модель памяти, используемая для СОМ .CODE [SEGMENT] ; начало сегмента кода ORG 100H ; начальное значение счетчика - 100h, конец PSP

BEGIN: JMP START ;обход через данные

FLDA DW 0F77Fh ;определение данных

FLDB DW 03219h

FLDC DW ?

START PROC NEAR

MOV AX,FLDA ;Переслать 1-е число в AX

ADС AX,FLDB ;Прибавить 2-е число к AX

MOV FLDC,AX ;Записать сумму в FLDC

RET ;Вернуться в DOS

ENDP

END BEGIN