С И С Т Е М Ы П Р О Г Р А М М И Р О В А Н И Я

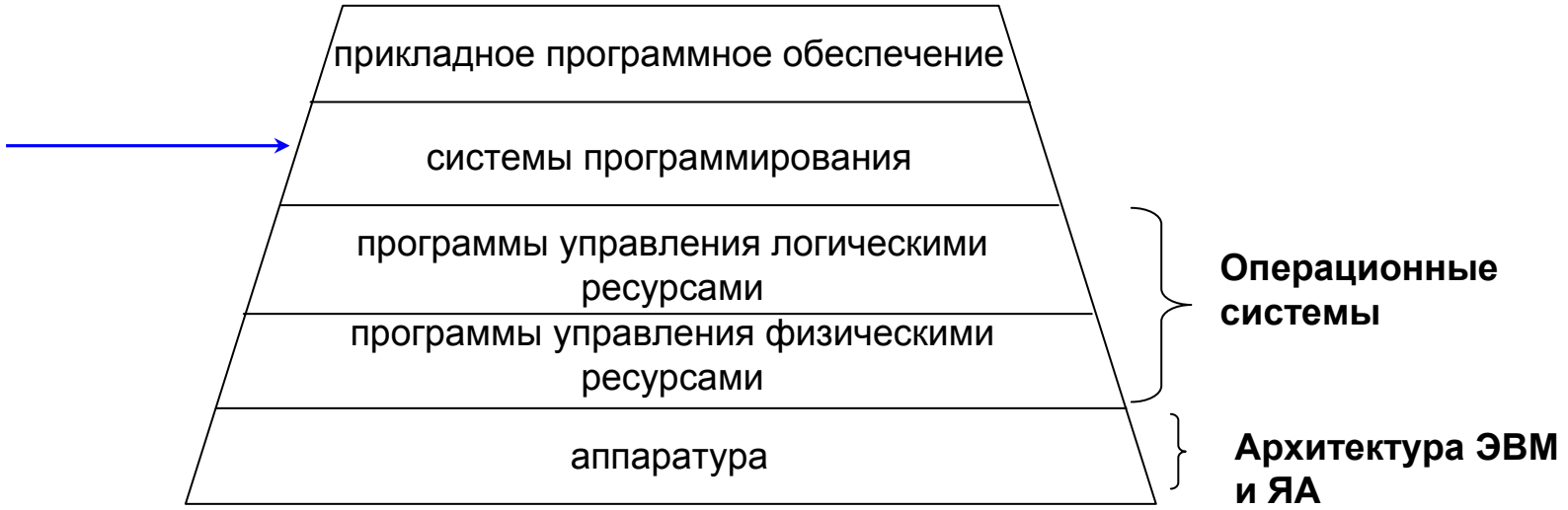
- Основные понятия, назначение, структура и функционирование СП
- Принципы ООП на примере языка C++ и СП, поддерживающие ООП
- Элементы теории трансляции

Коллоквиум по ООП и C++ (в третьей декаде марта)

- Письменная работа (обязательная) Баллы учитываются на экзамене

Темы на экзамене (письменный): СП и основы трансляции

С И С Т Е М Ы П Р О Г Р А М М И Р О В А Н И Я



Иерархия вычислительной системы

Развитие СП:

- программирование в машинных кодах
- автокоды, языки ассемблера
- трансляторы с языков высокого уровня
- визуальные средства автоматизации и проектирования

Определение: *системой программирования называется комплекс программных средств (инструментов, библиотек) , предназначенных для поддержки разработки программного продукта на протяжении всего жизненного цикла этого продукта*

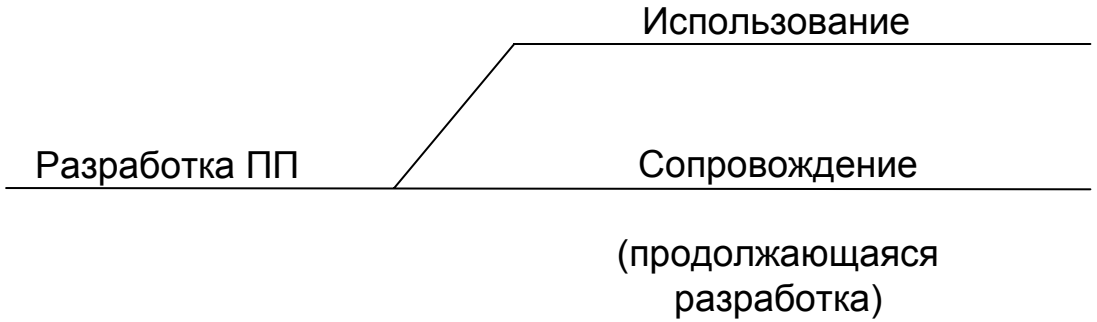
Программа -- создается для решения отдельной задачи автором программы и используется в некоторой конкретной операционной среде. Для других пользователей программы может потребоваться консультация автора.

Программный продукт -- программа, которая работает без авторского надзора в рамках некоторого набора операционных сред. Может исполняться, тестироваться и модифицироваться без участия автора (отчуждён от автора). Качество программного продукта должно быть существенно выше качества обычной программы. Программный продукт должен обладать “дружественным” интерфейсом пользователя, иметь полную пользовательскую (эксплуатационную) и техническую документацию, например, для модификации продукта. Тестирование программных продуктов должно проходить более тщательно и полно. Программный продукт должен быть настраиваемым и конфигурируемым.

Системный программный продукт (*интегрированный*) -- комплекс (пакет) программных продуктов. Например, пакет офисных программ.

Жизненный цикл программного продукта

Основные фазы жизненного цикла ПП :



Этапы жизненного цикла

1) Постановка задачи. Анализ (определение) требований
2) Проектирование
3) Написание текста программ (программирование, “кодирование”)
4) Компоновка или интеграция программного комплекса
5) Верификация, тестирование и отладка
6) Документирование
7) Внедрение
8) Тиражирование
9) Сопровождение, повторяющее все предыдущие этапы

Формализация определения требований (1-й этап ЖЦ)

-- Таблицы решений

Отображают связь входных данных с выходными. Для их построения входные данные разбиваются на группы, представители которых обрабатываются программным продуктом практически одинаково. Используются для данных, которые легко разбиваются на немногочисленные группы.

-- Функциональные диаграммы

Представляют собой графы с узлами, соответствующими входным данным. Задаются условия и/или ограничениями на данные или их сочетания. В диаграммах также описывается эффект обработки соответствующих данных.

-- Языки спецификаций, применяемые для формулирования требований (язык CLU , SDL и другие).

Проектирование (2-й этап ЖЦ)

Метод «разделяй и властвуй» для борьбы со сложностью

Разбиение (декомпозиция) задачи на отдельные модули (подсистемы). Спецификации для модулей. Дальнейшая декомпозиция модулей в соответствии со спецификациями.

Два подхода к декомпозиции:

-- алгоритмическая (структурное программирование «сверху-вниз», данные играют пассивную роль)

-- объектно-ориентированная (выделение отдельных объектов, которые способны воспринимать направляемые им сообщения и отвечать выполнением тех или иных ответных действий. Данные (объекты) играют активную роль, алгоритмы превращаются в операции над выделенными объектами)

CASE-технологии (*Computer Aided Software Engineering*) – средства автоматизированного проектирования
(охватывают все, от подсистем верхнего уровня до отдельных модулей с обязательной спецификацией всех механизмов межмодульного взаимодействия)

Результаты второго этапа:

-- схема иерархии подсистем (или модулей, для алгоритмической декомпозиции)

-- функциональность и интерфейсы каждой подсистемы

-- внутренняя структура каждого модуля: выбирается алгоритм работы модуля и способ внутреннего представления

Кодирование (3-й этап ЖЦ)

написание текстов программ на алгоритмическом языке

Компоновка (4-й этап ЖЦ)

интеграционный процесс комплексирования (комбинирования), то есть связывания отдельных частей программы, написанных разными людьми или группами, в одну большую систему программного обеспечения.

Верификация, тестирование, отладка (5-й этап ЖЦ)

Верификация -- проверка правильности программы

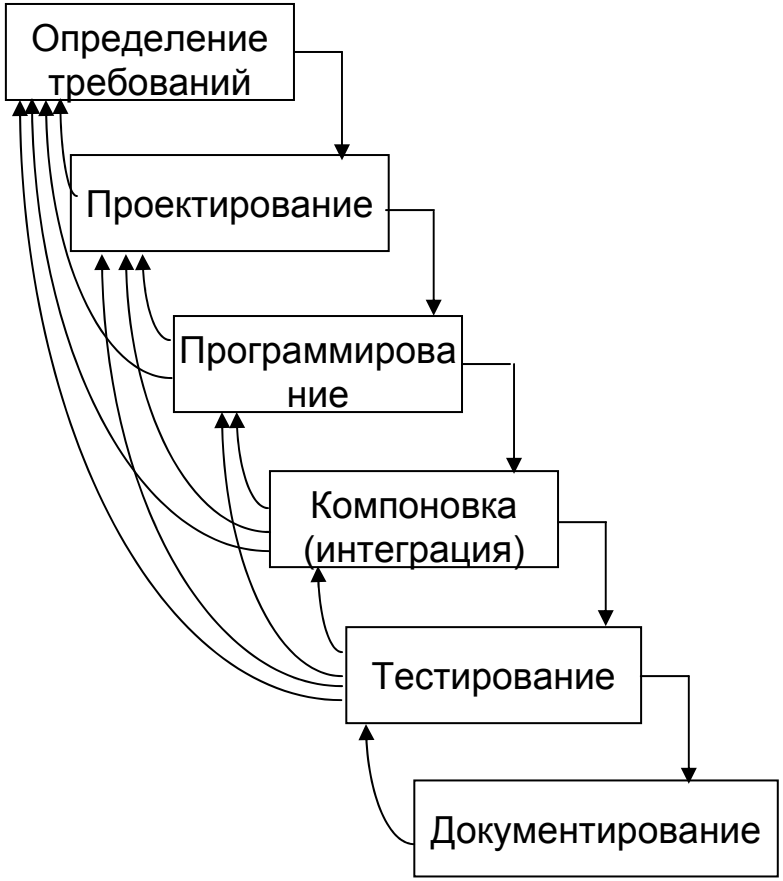
Тестирование -- процесс обнаружения дефектов в созданных программах, обнаружение несоответствия спецификациям.

Тест - это задание программе с заранее известным ответом. Существуют методы «белого» и «черного» ящиков.

Отладка -- выявление причин дефектов, а также их устранение

Регрессионное тестирование -- повторное тестирование после внесения исправлений , которое позволяет убедиться, что внесение исправлений в одни фрагменты программ, не нарушило функционирование других компонентов

Комплексное тестирование -- проверяется правильность взаимодействия всех автономно разработанных составных частей между собой



Реальный ход разработки программного обеспечения

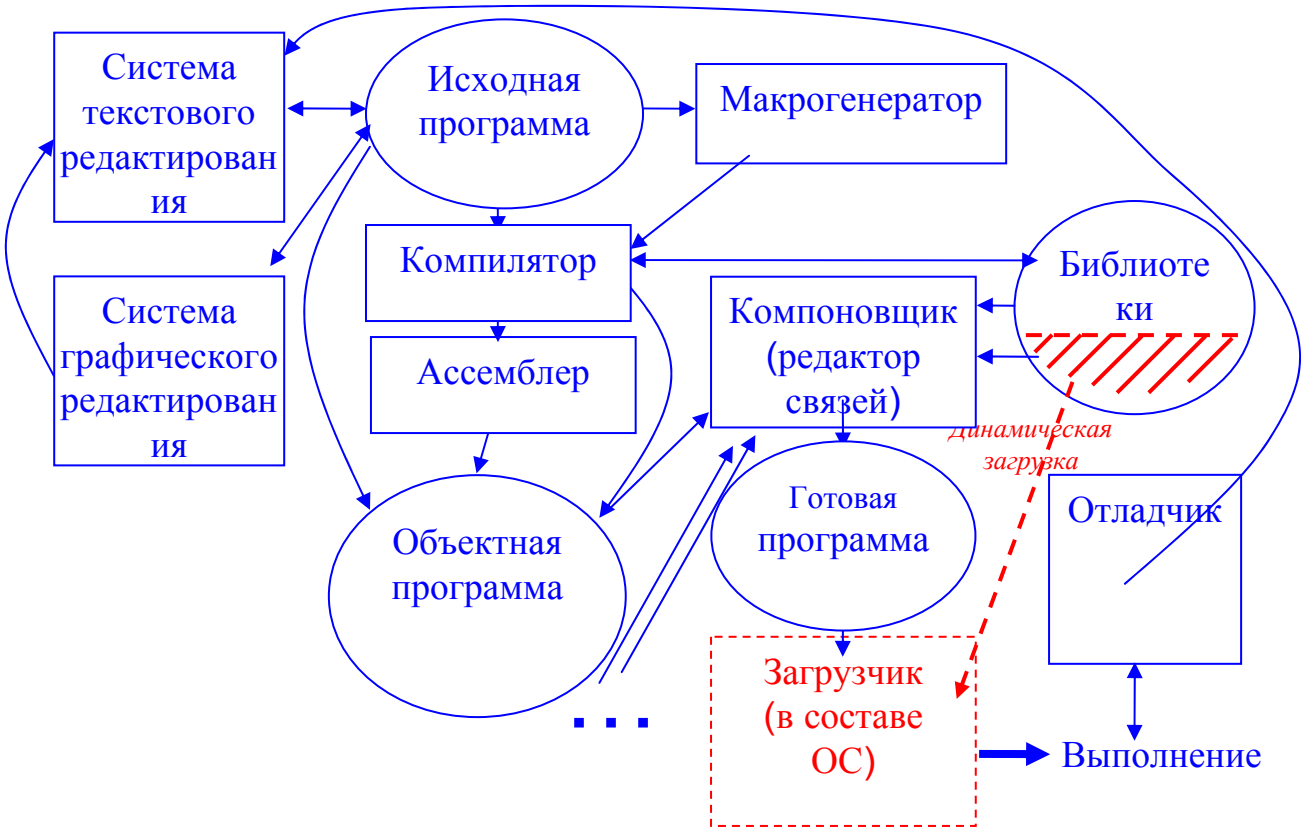
Существует и другие модели разработки, напр. «спиральная»

Примеры систем программирования:

СП Turbo Pascal для ОС MS-DOS (продукт компании Borland) и

СП Си для ОС UNIX (версия Free BSD 4.3).

Схема классической системы программирования



(В СП Си под UNIX одна и та же команда CC является программной оболочкой и компилятора и редактора связей)

Для фиксации, удобной визуализации и редактирования материалов, возникающих на каждом из этапов создания ПП желательны дополнительные средства СП на основе базы данных, обеспечивающие:

согласованность *(изменения в одном месте должны инициировать изменения там, где используется измененный фрагмент ПП),*

непротиворечивость *(принимаемые решения на одном из этапов должны быть согласованы с решениями, принимаемыми на других этапах создания ПП),*
корректность изменений *(вносимые изменения не должны нарушать общие требования к ПП),*

"историю" развития проекта *(предыдущие версии принимаемых решений).*

возможность работы группы разработчиков в реальном времени

автоматическое или автоматизированное кодирование и документирование ПП

автоматизацию календарного планирования сроков создания ПП*,* ***разбиение общей постановки задачи на подзадачи, распределения подзадач по конкретным разработчикам ПП*** **(**пример: MS Project**)**

Дополнительные средства В СП Си (C++), базирующейся на ОС UNIX:

Координатор (программа управления сборкой) Make (GNU Make).

Make существенно упрощает процесс сборки проектов.
Make отслеживает изменения файлов и перекомпилирует при обращении к ней только изменившиеся файлы и файлы, связанные с ними по компиляции.

Система SCCS (Source Code Control System)

SCCS предназначена для отслеживания изменений различных версий файлов проекта (создаваемого под ОС UNIX, не только в СП С) и разделения доступа к ним.

Основные функции SCCS:

- документирование модификаций проекта,
- контроль полномочий пользователей-разработчиков проекта,
- сопровождение параллельных версий проекта,
- восстановление старых версий проекта.

Аналогичной системой под ОС Windows является RCS (Revision Control System).

Программа lint

Lint применяется для более жесткого контроля программ, написанных на Си, по сравнению с компилятором. Например, она

- обнаруживает объекты, по-разному определенные в разных файлах,
- обнаруживает переменные, которые используются раньше, чем им присвоено значение или не используются вовсе,
- осуществляет контроль соответствия формальных и фактических параметров для функций, декларированных как f( )

*Главное достоинство -- может проконтролировать **всю** программу, что особенно ценно для языка, допускающего независимую компиляцию.*

- ***Серия Manual.***

Программа **man** в интерактивном режиме предоставляет пользователю документацию по всем компонентам системы программирования и операционной системы.

- **Форматер nroff**

Nroff (работает с любой СП под ОС UNIX) используется для создания печатной документации. Она преобразует неформатированный текстовый файл в форматируемый (команды форматирования вставляются непосредственно в текст).

- **Форматер tbl**

Tbl (работает с любой СП под ОС UNIX) также используется для создания печатной документации. Она форматирует табличные данные.

Другие компоненты классической системы программирования
Интегрированная среда разработки (ИСР)

Современная ИСР - комплекс программных средств, поддерживающих полный жизненный цикл ПП. Комплекс инструментальных средств, входящих в состав ИСР, иногда называют интегрированным CASE-средством.

Простая ИСР объединяет в себе возможности текстовых редакторов исходных текстов программ, отладчиков и командный язык компиляции

Примеры ИСР: Turbo Pascal, Delphi, Visual Studio (под ОС Windows);
K-develop для K Desktop Environment под ОС LINUX (в приведённой терминологии они будут простыми ИСР).

“Продвинутая” ИСР *:*

-- репозиторий*,* являющийся основой ИСР.
Он должен обеспечивать хранение версий проекта и его отдельных компонентов, синхронизацию поступления информации от различных разработчиков при групповой разработке, контроль данных о проекте на полноту и непротиворечивость*;*

-- графические средства анализа и проектирования*,*
обеспечивающие создание и редактирование иерархически связанных диаграмм, образующих модели ПП, а также графический пользовательский интерфейс, обеспечивающий взаимодействие с функциями API (Application Program Interface - прикладного программного интерфейса ОС).

-- средства разработки приложений, включая языки
-- редактор текстов;
-- средства документирования;
-- средства тестирования и отладки;
-- средства управления проектом;
-- средства реинжиниринга (позволяют восстановить логику и структуру программы по ее исходным текстам, с целью модификации или перенесения на другую платформу).

Редакторы текстов

-- строковые, позволяющие смотреть и редактировать текст только на одной строке, заканчивающейся признаком "конец строки";

-- потоковые, воспринимающие текст в виде потока символов, признак конца строки также является символом;

-- экранные, позволяющие перемещать по тексту окно, по которому можно перемещать курсор и устанавливать его в нужные позиции; многие экранные редакторы позволяют видеть на экране текст в том виде, в котором он будет напечатан.

Основные функции текстового редактора в рамках ИСР программного обеспечения:

1) подготовка текста программы (обычные действия по созданию, редактированию, сохранению файла с текстом программы),
2) многооконный интерфейс с поддержкой режима "буксировки" фрагментов текста мышкой (drag&drop),
3) интеграция с компилятором:
 -- визуализация текста с выделением лексем (синтаксическая подсветка элементов языка),
 -- дополнение кода, интерактивная подсказка (a. $\Rightarrow$ получаем список всех членов структры, f($\Rightarrow$ контекстный список параметров функции f),
 -- шаблоны кода (на "горячих" клавишах - часто используемые программные конструкции),
 -- всплывающие подсказки об атрибутах идентификаторов, если на них установить курсор, отображение ошибок, обнаруженных на этапе компиляции, в тексте программы,
4) интеграция с отладчиком:
 -- отображение контрольных точек останова при отладке,
 -- отображение текущего значения объекта, при наведении курсора на идентификатор.

Средства тестирования. Отладчики.

Отладчик — программа, помогающая анализировать поведение отлаживаемой программы, обеспечивая ее трассировку.

Основные функции отладчика в рамках ИСР программного обеспечения:

1) пошаговое выполнение программы (шаг = строка; с трассировкой внутри вызываемой функции и без нее),
2) выполнение программы до строки, в которой в редакторе стоит курсор,
3) выделение выполняемой в данный момент строки,
4) приостановка выполнения программы:
   -- можно запросить значение переменной,
   -- можно заказать вычисление некоторого выражения,
   -- можно изменить значение переменной и продолжить выполнение программы (!но редко какой отладчик позволяет изменять программный код, т.е. поддерживает частичную перекомпиляцию),
5) расстановка/снятие точек останова, которые визуализируются в текстовом редакторе,
6) выдача всей информации в терминах исходной программы.

Автоматизировать процесс тестирования и отладки помогают следующие средства:

- средства отслеживания тестового покрытия, определяющие участки кода, пройденные и пропущенные при тестировании;
- профилировщики , определяющие линейные участки кода;
- средства построения срезов программы, т.е. всех операторов, которые могут влиять на значение некоторой переменной в некоторой точке программы.

Редакторы связей

Редактор связей (компоновщик) предназначен для связывания между собой (по внешним данным) объектных файлов, порождаемых компилятором, а также файлов (статических) библиотек, входящих в состав СП. Создает таблицы: 1) для трансляции относительных адресов (она используется загрузчиком) ; 2) для точек вызова функций динамически подключаемых библиотек.

Загрузчики

Чтобы программа могла быть исполнена, необходим модуль, преобразующий относительные адреса в реальные, абсолютные адреса непосредственно в момент запуска программы на выполнение. Этот процесс называется трансляцией адресов и выполняет его модуль, называемый загрузчиком. В современных вычислительных системах загрузчик обычно является частью ОС, а не СП.

Средства конфигурирования

Конфигурирование и управление версиями программного продукта облегчает работу при разработке и сопровождении ПП

-- конфигурирование из командной строки,
-- использование командных файлов,
-- работа в интегрированных средах с проектами программных комплексов,
-- использование систем управления версиями программных комплексов.

Системы управления жизненным циклом программ: Peforce SCM (Software Configuration Management) и IBM Rational ClearCase. В задачу этих систем входит создание и изменение конфигураций программ, их комплексирование, регистрация поставок, а также обеспечение повторного использования программ. На клиентских местах разрешается использовать все наиболее распространенные операционные системы (Windows, UNIX, Linux), а также наиболее современные системы разработки, включая Rational Application Developer, WebSphere Studio, Microsoft Visual Studio .NET, Eclipse.

Свободно распространяемые системы управления версиями, например система CVS (Concurrent Versions System).

Серверная часть системы может работать под управлением любого варианта операционной системы UNIX – FreeBSD, Linux и др. Клиентские части работают под управлением UNIX систем, а также системы Windows.

Система CVS поддерживает историю дерева каталогов (репозитория) с исходным кодом, работая с последовательностью изменений. Каждое изменение в файлах репозитория маркируется моментом времени, когда оно было сделано, и именем пользователя, совершившим изменение. Обычно человек, совершивший изменение, также предоставляет текстовое описание причины, по которой произошло изменение. Система CVS может отвечать на такие вопросы, как
-- Кто совершил данное изменение?
-- Когда они его совершили?
-- Зачем они это сделали?
-- Какие еще изменения произошли в то же самое время?

Профилировщики

строит профиль программы, где выделены линейные участки кода (фрагменты программы, где нет передачи управления), с указанием времени и частоты их исполнения; используется для более эффективной оптимизации программы (редко исполняемые линейные участки можно не оптимизировать, сосредоточив основные усилия на оптимизации критичных циклических участков). Позволяет получать информацию о вызовах функций ядра ОС, аппаратных прерываниях, состояниях потоков ввода/вывода, сообщениях и деятельности планировщика.

В состав операционных систем UNIX входит программа *prof.* (Т.е. профилировщик *prof* можно считать компонентом СП Си/Си++ на базе Unix.)

Справочные системы

содержит справочные материалы по языку программирования и компонентам

-- справочные системы в виде базы данных с индексами (алфавитным и тематическим), облегчающими поиск информации;

-- контекстный поиск в ИСР

-- удаленная работа с документацией (через Интернет, документация на сайте разработчика)

Библиотеки

По техническому составу делятся на две категории: библиотеки функций исходного языка и библиотеки функций операционной системы, в составе которой должна будет работать обрабатываемая программа. Эта операционная система может отличаться от той, в составе которой функционирует сама система программирования.

Статические библиотеки

Представляют собой процедуры и функции, встраиваемые внутрь программ. Подключаются к программам, готовящимся к выполнению, ровно один раз в момент формирования редактором связей полной программы. Готовая программа далее не зависит от библиотеки, т.к. содержит внутри себя весь необходимый код. Однако объем кода может быть весьма велик, разные программы, использующие одни и те же библиотечные функции, содержат внутри себя один и тот же код , что увеличивает объем занимаемой дисковой и оперативной памяти. При изменениях библиотеки приходится перекомпилировать основанные на ней программы

.

Динамически загружаемые библиотеки

Компоненты динамических библиотек (динамически загружаемые компоненты и библиотеки) подключаются к программам во время выполнения этих программ. Компоненты динамических библиотек не связаны с программами, которые к ним обращаются, и распространяются отдельно от них. На этапе компоновки программы редактор связей формирует таблицу точек вызова функций библиотеки для последующей операции динамического связывания (то есть компоновщик не помещает в программу тела функций). Различные программы могут пользоваться кодом одной и той же библиотеки, содержащейся в оперативной памяти. При модификации библиотечных подпрограмм не требуется перекомпилировать программы, их использующие, однако поведение программ может измениться из-за изменений в библиотечных подпрограммах.

По функциональному наполнению все используемые в составе современных систем программирования библиотеки можно классифицировать следующим образом:

-- библиотеки функций, процедур и макроопределений,
-- библиотеки классов,
-- библиотеки компонентов.

Библиотеки функций (подпрограмм) во многом определяют возможности системы программирования в целом. Различают два вида:

Стандартные библиотеки системных подпрограмм для ЯП (ввод/вывод и т.п.)

Библиотеки прикладных программ (например, функции, реализующие численные методы). Библиотеки функций представляют собой откомпилированные объектные модули.

Подключаются статически во время работы редактора связей и динамически во время выполнения программы с помощью загрузчика.

Библиотеки классов - для объектно-ориентированных ЯП. Все их классы должны быть написаны на том же ЯП, на котором пишется программа, в которой используются библиотечные классы.

Может включать разные виды: *конкретные классы*, *абстрактные классы, шаблоны классов*.

Библиотеки классов интегрируются в программу на этапе компиляции и компилируются со всей программой вместе.

Библиотеки компонент - это библиотека скомпилированных программных модулей, которые можно использовать в качестве составной части программы, и которыми можно манипулировать во время разработки программы. Компоненты бывают локальные (находящиеся на том же компьютере, где создается прикладная программа) и распределенные (расположенные на сервере и доступные по сети).

Технологии, использующие библ. компоненты:, COM, DCOM, CORBA, Java Beans, ActiveX

Критерии проектирования стандартных библиотек

-- поддержка свойств языка, например, управление памятью и предоставление информации об объектах во время выполнения программ;

-- предоставлять информацию о зависящих от реализации аспектах языка, например, о максимальных размерах целых значений;

-- предоставлять функции, которые не могут быть написаны оптимально для всех вычислительных систем на данном языке программирования, например, функции вычисления квадратного корня *sqrt()* или пересылок блоков памяти *memmove()*; т.е. обеспечивать эффективность для каждой конкретной ВС

-- нетривиальные средства программирования (работа со списками, функции сортировки, потоки ввода/вывода); т.е. общезначимость предоставляемых средств

-- предоставлять основу для расширения собственных возможностей, в частности, соглашения и средства поддержки, позволяющие обеспечить операции для данных, имеющих определяемые пользователями типы, в том же стиле, в котором обеспечиваются операции для встроенных типов (например, ввод/вывод);

-- служить основой создания других библиотек.