УВАЖАЕМЫЕ СТУДЕНТЫ! Законспектируйте в своей рабочей тетради по дисциплине приведенную лекцию (объемом 4-5 страницы), ответьте *письменно* на контрольные вопросы.

Результаты работы, фотоотчет, предоставить преподавателю на e-mail: r.bigangel@gmail.com до 30.01.2023.

При возникновении вопросов по приведенному материалу обращаться по следующему номеру телефона: (072)111-37-59, (Viber, WhatsApp), vk.com: https://vk.com/daykini

ВНИМАНИЕ!!! При отправке работы, не забывайте указывать ФИО студента, наименование дисциплины, дата проведения занятия (по расписанию).

Лекция № 34
Тема «Модульное программирование, как метод разработки программы»

Модульное программирование – это такой способ программирования, при котором вся программа разбивается на группу компонентов, называемых модулями, причем каждый из них имеет свой контролируемый размер, четкое назначение и детально проработанный интерфейс с внешней средой. Единственная альтернатива модульности – монолитная программа, что, конечно, неудобно. Таким образом, наиболее интересный вопрос при изучении модульности – определение критерия разбиения на модули.

Концепции модульного программирования. В основе модульного программирования лежат три основных концепции:

Принцип утаивания информации Парнаса. Всякий компонент утаивает единственное проектное решение, т.е. модуль служит для утаивания информации. Подход к разработке программ заключается в том, что сначала формируется список проектных решений, которые особенно трудно принять или которые, скорее всего, будут меняться. Затем определяются отдельные модули, каждый из которых реализует одно из указанных решений.

Аксиома модульности Коуэна. Модуль – независимая программная единица, служащая для выполнения некоторой определенной функции программы и для связи с остальной частью программы. Программная единица должна удовлетворять следующим условиям:

– блочность организации, т. е. возможность вызвать программную единицу из блоков любой степени вложенности;

– синтаксическая обособленность, т. е. выделение модуля в тексте синтаксическими элементами;

– семантическая независимость, т. е. независимость от места, где программная единица вызвана;

– общность данных, т. е. наличие собственных данных, сохраняющихся при каждом обращении;

– полнота определения, т. е. самостоятельность программной единицы.

Сборочное программирование Цейтина. Модули – это программные кирпичи, из которых строится программа. Существуют три основные предпосылки к модульному программированию:

– стремление к выделению независимой единицы программного знания. В идеальном случае всякая идея (алгоритм) должна быть оформлена в виде модуля;

– потребность организационного расчленения крупных разработок;

– возможность параллельного исполнения модулей (в контексте параллельного программирования).

Определения модуля и его примеры. Приведем несколько дополнительных определений модуля.

Модуль – это совокупность команд, к которым можно обратиться по имени.

Модуль – это совокупность операторов программы, имеющая граничные элементы и идентификатор (возможно агрегатный).

Функциональная спецификация модуля должна включать:

– синтаксическую спецификацию его входов, которая должна позволять построить на используемом языке программирования синтаксически правильное обращение к нему;

– описание семантики функций, выполняемых модулем по каждому из его входов.

Разновидности модулей. Существуют три основные разновидности модулей:

1) «Маленькие» (функциональные) модули, реализующие, как правило, одну какую-либо определенную функцию. Основным и простейшим модулем практически во всех языках программирования является процедура или функция.

2) «Средние» (информационные) модули, реализующие, как правило, несколько операций или функций над одной и той же структурой данных (информационным объектом), которая считается неизвестной вне этого модуля. Примеры «средних» модулей в языках программирования:

a) задачи в языке программирования Ada;

b) кластер в языке программирования CLU;

c) классы в языках программирования C++ и Java.

3) «Большие» (логические) модули, объединяющие набор «средних» или «маленьких» модулей. Примеры «больших» модулей в языках программирования:

a) модуль в языке программирования Modula-2;

b) пакеты в языках программирования Ada и Java.

Набор характеристик модуля предложен Майерсом [Майерс 1980]. Он состоит из следующих конструктивных характеристик:

1) размера модуля;

В модуле должно быть 7 (+/-2) конструкций (например, операторов для функций или функций для пакета). Это число берется на основе представлений психологов о среднем оперативном буфере памяти человека. Символьные образы в человеческом мозгу объединяются в «чанки» – наборы фактов и

связей между ними, запоминаемые и извлекаемые как единое целое. В каждый момент времени человек может обрабатывать не более 7 чанков.

Модуль (функция) не должен превышать 60 строк. В результате его можно поместить на одну страницу распечатки или легко просмотреть на экране монитора.

2) прочности (связности) модуля;

Существует гипотеза о глобальных данных, утверждающая, что глобальные данные вредны и опасны. Идея глобальных данных дискредитирует себя так же, как и идея оператора безусловного перехода goto. Локальность данных дает возможность легко читать и понимать модули, а также легко удалять их из программы.

Связность (прочность) модуля (cohesion) – мера независимости его частей. Чем выше связность модуля – тем лучше, тем больше связей по отношению к оставшейся части программы он упрятывает в себе. Можно выделить типы связности, приведенные ниже.

Функциональная связность. Модуль с функциональной связностью реализует одну какую-либо определенную функцию и не может быть разбит на 2 модуля с теми же типами связностей.

Последовательная связность. Модуль с такой связностью может быть разбит на последовательные части, выполняющие независимые функции, но совместно реализующие единственную функцию. Например, один и тот же модуль может быть использован сначала для оценки, а затем для обработки данных.

Информационная (коммуникативная) связность. Модуль с информационной связностью – это модуль, который выполняет несколько операций или функций над одной и той же структурой данных (информационным объектом), которая считается неизвестной вне этого модуля. Эта информационная связность применяется для реализации абстрактных типов данных.

Обратим внимание на то, что средства для задания информационно прочных модулей отсутствовали в ранних языках программирования (например, FORTRAN и даже в оригинальной версии языка Pascal). И только позже, в языке программирования Ada, появился пакет – средство задания информационно прочного модуля.

3) сцепления модуля с другими модулями;

Сцепление (coupling) – мера относительной независимости модуля от других модулей. Независимые модули могут быть модифицированы без переделки других модулей. Чем слабее сцепление модуля, тем лучше. Рассмотрим различные типы сцепления.

Независимые модули – это идеальный случай. Модули ничего не знают друг о друге. Организовать взаимодействие таких модулей можно, зная их интерфейс и соответствующим образом перенаправив выходные данные одного модуля на вход другого. Достичь такого сцепления сложно, да и не нужно, поскольку сцепление по данным (параметрическое сцепление) является достаточно хорошим.

Сцепление по данным (параметрическое) – это сцепление, когда данные передаются модулю, как значения его параметров, либо как результат его обращения к другому модулю для вычисления некоторой функции. Этот вид сцепления реализуется в языках программирования при обращении к функциям (процедурам). Две разновидности этого сцепления определяются характером данным.

· Сцепление по простым элементам данных.

· Сцепление по структуре данных. В этом случае оба модуля должны знать о внутренней структуре данных.

4) рутинности (идемпотентность, независимость от предыдущих обращений) модуля.

Рутинность – это независимость модуля от предыдущих обращений к нему (от предыстории). Будем называть модуль рутинным, если результат его

работы зависит только от количества переданных параметров (а не от количества обращений).

Модуль должен быть рутинным в большинстве случаев, но есть и случаи, когда модуль должен сохранять историю. В выборе степени рутинности модуля пользуются тремя рекомендациями.

В большинстве случаев делаем модуль рутинным, т. е. независимым от предыдущих обращений.

Зависящие от предыстории модули следует использовать только в тех случаях, когда это необходимо для сцепления по данным.

В спецификации зависящего от предыстории модуля должна быть четко сформулирована эта зависимость, чтобы пользователи имели возможность прогнозировать поведение такого модуля.