УВАЖАЕМЫЕ СТУДЕНТЫ! Изучите приведенную лекцию, законспектируйте основные тезисы.

Ответы на вопросы, фотоотчет, предоставить преподавателю на e-mail: r.bigangel@gmail.com до 13.02.2023.

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 № 36

Тема «Теоретические предпосылки структурного программирования»

В теории схем программ было замечено, что некоторые случаи блок-схем легче поддаются анализу. Поэтому естественно было выделить такой *класс* блок-схем, что и сделали итальянские ученые С. Бем и К. Джакопини в 1966 г. Они доказали, что любую блок-схему можно привести к структурированному виду, использовав несколько дополнительных булевых переменных. *Э. Дейкстра* подчеркнул, что программы в таком виде, как правило, являются легче понимаемыми и модифицируемыми, так как каждый блок имеет один вход и один *выход*.

В качестве методики структурного программирования *Э. Дейкстра* предложил пользоваться лишь конструкциями *цикла* и условного оператора, изгоняя go to как концептуально противоречащее этому стилю

Пожалуй, это первое *концептуальное противоречие*, явно отмеченное и учтенное в теории и практике программирования (и даже во всей современной науке). Но, поскольку не было даже наметок теории неформализуемых понятий, и на работу с ними переносили *опыт* работы с малыми формализациями, структурное противоречие было воспринято следующим образом.

К несчастью, оператор go to формально совместим с другими конструкциями традиционных (тогда говорили – универсальных)

алгоритмических языков. Но реально он плохо взаимодействует с ними. Значит, он плох сам по себе.

Структурное *программирование* основано главным образом на теоретическом аппарате теории рекурсивных функций. *Программа* рассматривается как частично-рекурсивный операторнад библиотечными подпрограммами и исходными операциями. Структурное *программирование* базируется также на теории доказательств, прежде всего на естественном выводе. Структура программы соответствует структуре простейшего математического рассуждения, не использующего сложных лемм и абстрактных понятий[2].

Средства структурного программирования в первую *очередь* включаются во все языки программирования традиционного типа и во многие нетрадиционные языки. Они занимают основное *место* в учебных курсах программирования и в теоретических работах.

При структурном программировании присваивания и локальные действия становятся органичной частью программы. Достаточно лишь внимательно следить, чтобы каждая *переменная* в модуле использовалась для одной конкретной цели, и не допускать "экономии", при которой ненужная в данном месте *переменная* временно используется под совсем другое *значение*. Такая "экономия" запутывает структуру информационных зависимостей, которая при данном стиле должна быть хорошо согласована со структурой программы.

Структурное *программирование* естественно возникает во многих классах задач, прежде всего в таких, где **задача естественно расщепляется на подзадачи, а информация – на достаточно независимые структуры данных**. Основной его *инвариант*:

действия и условия локальны.

Необходимой чертой хорошей реализации структурного стиля программирования является соблюдение согласованности, а в идеале и единства, следующих компонентов программы:

1. **Структура информационного пространства**. Содержательно любую задачу можно описать как переработку объектов, полный набор которых называется **информационным пространством** задачи.

Для структурного стиля программирования требуется следующее. Задача разбивается на подзадачи, и таким образом выстраивается дерево вложенности подзадач. Информационное пространство структурируется в точном соответствии с деревом вложенности: для каждой подзадачи оно состоит из ее **локальных объектов**, определяемых вместе с подзадачей и для нее, и так называемых **глобальных объектов**, определяемых как информационное пространство непосредственно объемлющей подзадачи. Таким образом, информационное пространство всей задачи (подзадачи самого верхнего уровня) расширяется по мере перехода к подзадачам за счет их локальных объектов. Для различных дочерних подзадач одной подзадачи оно имеет общую часть - информационное пространство родительской подзадачи[3].

2. **Структуры управления**. Стиль структурного программирования в его общепринятом варианте предполагает использование строго ограниченного набора *управляющих конструкций*: последовательность операторов, условные и выбирающие операторы, все вычислительные ветви которых сходятся в одной точке программы, а также процедуры, вычисления которых всегда заканчиваются возвратом управления в точку вызова.

3. К структурным операторам добавляются **либо** *циклы*, **либо** *рекурсии*.

Внимание!

Этой альтернативы Вы не встретите в традиционных изложениях структурного программирования. *Концептуальное противоречие* между *циклами* и *рекурсиями* намного мягче, чем между операторами структурного программирования и *структурными переходами*, и оно отмечается лишь в виде изредка встречающихся прагматических указаний (благих пожеланий) не смешивать их произвольно.

4. **Потоки передачи данных**. Разбивая задачу на подзадачи, программист предусматривает их взаимодействие по данным: одни подзадачи передают другим данные для переработки.

5. **Структуры данных**. Данные объединяются в логически связанные фрагменты, соответствующие структурам задачи либо вспомогательных конструкций, вводимых для ее решения.

6. **Призраки**. Часто даже сама программа не может быть объяснена через понятия, которые используются внутри нее. Еще чаще это происходит для ее связей с внешним миром. Понимание программы возможно лишь после сопоставления реальных внутрипрограммных объектов с идеальными внепрограммными. Эти идеальные внепрограммные объекты (*призраки*) часто не просто не нужны, но даже вредны для исполнения программы[4].

Первым обратил внимание на необходимость введения *призраков* для логического и концептуального анализа программ *Г. С. Цейтин* в 1971 г. В Америке это "независимо" открыли заново в 1979 г., хотя упомянутая статья *Цейтина* была опубликована на английском языке в общедоступном издании. Даже название сущностям было дано то же самое... Этому важнейшему и традиционно игнорируемому понятию посвящена отдельная лекция в курсе "Основания программирования".

7. ***Подпорки*** в программе - значения, конструкции и сущности, которые не нужны для понимания задачи и программы, не определяются сущностью задачи и алгоритма, но вынужденно вставляются для реализации данного алгоритма на конкретной системе.

Подпорки противоположны *призракам*. На самом деле они являются той формой, в которой материя проникает в программу, и в этом качестве противостоят всей совокупности идеальных сущностей, порождающих структуру программы: как реальных, так и призрачных, - и порою грубо ее искажают. Но без *подпорок* программа просто не будет работать или будет работать неэффективно.

Для структурного программирования весьма важно требование:

Все структуры подчиняются структуре информационного пространства.

Это общее требование конкретизируется в следующие.

1. Необходимо, чтобы структура управления программы была согласована со структурой ее информационного пространства. Каждой структуре управления соответствуют согласующиеся с ней структуры данных и часть информационного пространства. Это условие позволяет человеку легко отслеживать порядок выполнения конструкций в программе.

2. Подзадачи могут обмениваться данными только посредством обращения к объектам из общей части их информационных пространств (в современных языках чаще всего к глобальным).

3. Информационные потоки должны протекать согласно иерархии структур управления; мы должны четко видеть для каждого блока программы, что он имеет на входе и что дает на выходе. Таким образом, **свойства каждого логически завершенного фрагмента программы должны ясно осознаваться и в идеале четко описываться в самом тексте программы и в сопровождающей ее** документации[5].

4. Описание переменных, представляющих перерабатываемые объекты, а также других, вспомогательных переменных при структурном программировании строго подчиняется разбиению задачи на подзадачи.

5. Все *призраки* действуют на своем структурном месте и соответствуют идеальным сущностям, которые, согласно парадоксу изобретателя, должны вводиться для эффективного решения задачи.

6. Все *подпорки* строго локализованы в том месте, где их вынуждены ввести. Желательно даже обозначать их по-другому, чем идеальные сущности, например, оставляя мнемонические имена лишь для идеальных сущностей, а *подпорки* именовать джокерами типа x или i. Необходимо строго следить за тем, чтобы *подпорки* не искажали идеальную структуру программы.

Структурное *программирование* лучше всего описано теоретически, но частные описания не сведены в единую систему. Одни книги трактуют его с точки зрения программиста, другие - с точки зрения теоретика. Так что даже здесь единой системы взглядов еще нет, хотя, видимо, все основания для ее формирования уже имеются.