Методический материал

В качестве основной методологии разработки программных средств для проекта ЭВМ пятого поколения было избрано логическое программирование, ярким представителем которого является язык Пролог. Думается, что и в настоящее время Пролог остается наиболее популярным языком искусственного интеллекта в Японии и Европе (в США, традиционно, более распространен другой язык искусственного интеллекта — язык функционального программирования Лисп).

Известна классификация языков программирования по их близости либо к машинному языку, либо к естественному человеческому языку. Те, что ближе к компьютеру, относят к языкам **низкого уровня**, а те, что ближе к человеку, называют языками **высокого уровня**. В этом смысле декларативные языки можно назвать языками **сверхвысокого** или **наивысшего** уровня, поскольку они очень близки к человеческому языку и человеческому мышлению.

Еще одной особенностью языков высокого уровня была возможность повторного использования ранее написанных программных блоков, выполняющих те или иные действия, посредством их идентификации и последующего обращения к ним, например по имени. Такие блоки получили название **функций** или процедур, и программирование приобрело более упорядоченный характер.

Одним из путей развития декларативного стиля программирования стал функциональный подход, возникший после создания языка LISP.

Отличительной особенностью данного подхода является то, что любая программа, написанная на таком языке, может интерпретироваться как функция с одним или несколькими аргументами. Такой подход даст возможность прозрачного моделирования текста программ математическими средствами, а значит, весьма интересен с теоретической точки зрения.

Сложные программы при таком подходе строятся посредством агрегирования функций. При этом текст программы представляет собой функцию, некоторые аргументы которой можно также рассматривать как функции. Таким образом, повторное использование кода сводится к вызову ранее описанной функции, структура которой, в отличие от процедуры императивного языка, математически прозрачна.

Более того, типы отдельных функций, используемых в функциональных языках, могут быть переменными. Таким образом, обеспечивается возможность обработки разнородных данных (например, упорядочение элементов списка по возрастанию для целых чисел, отдельных символов и строк) или полиморфизм.

Еще одним важным преимуществом реализации языков функционального программирования является автоматизированное динамическое распределение памяти компьютера для хранения данных. При этом программист избавляется от обязанности контролировать данные, а при необходимости может запустить функцию "сборки мусора" – очистки памяти от тех данных, которые больше не потребуются программе (обычно этот процесс периодически инициируется компьютером).

Таким образом, при создании программ на функциональных языках программист сосредотачивается на области исследований (предметной области) и в меньшей степени

заботится о рутинных операциях (обеспечении правильного с точки зрения компьютера представления данных, "сборке мусора" и т.д.).

Автор языка Лисп – профессор математики и философии Джон Мак-Карти, выдающийся ученый в области искусственного интеллекта. Он предложил проект языка Лисп, идеи которого возбудили не утихающие до наших дней дискуссии о сущности программирования. Сформулированная Джоном Мак-Карти (1958) концепция символьной обработки информации восходит к идеям Чёрча и других видных математиков конца 20-ых годов предыдущего века. Выбирая лямбда-исчисление как основную модель, Мак-Карти предложил функции рассматривать как общее понятие, к которому могут быть сведены все другие понятия программирования.

Функцией называется правило, по которому каждому значению одного или нескольких аргументов ставится в соответствие конкретное значение результата.

Элементарные данные языка Лисп называются атомами. Атомы могут иметь вид имен, чисел или других объектов, неделимых базовыми средствами языка.

Атомы, выглядящие как имена, могут обладать характеристиками - свойствами, задаваемыми системой или программой. Значения, связанные с атомами и определения функций – примеры характеристик. Особый интерес представляют рекурсивные функции и методы их реализации в системах программирования.

Точечная нотация

При реализации Лиспа в качестве единой универсальной базовой структуры для конструирования сложных символьных выражений использовалась так называемая "точечная нотация" (dot-notation), согласно которой синтаксически сложную структуру, состоящую из двух атомов, можно записать (ATOM1 . ATOM2). В памяти такая структура представляется **бинарным узлом**, левая и правая части которого равноправны и хранят указатели на эти атомы. Вместо атомов могут быть использованы более сложные символьные выражения (представляющие данные любой природы). В результате, могут быть построены различные символьные выражения – **S-выражения**.

S-выражение - это или атом или заключенная в скобки пара из двух S-выражений, разделенных точкой.

Все сложные символьные выражения являются S-выражениями, а в памяти представляются структурами одинаково устроенных блоков - бинарных узлов, содержащих указатели на S-выражения. Каждый бинарный узел соответствует минимальному блоку памяти из двух указателей.

Переменные обозначают некоторый неизвестный объект из некоторого множества объектов.

В момент фиксации утверждений в программе не имеют значения. Значения для переменных могут быть установлены системой только в процессе поиска ответа на вопрос, т.е. реализации программы.

Виды переменных:

- Именованные – есть имя – комбинация символов;
- Анонимные – нет имени – символ подчеркивания;
- Связанная (конкретизирована) – имеется объект, который в данный момент обозначает данная переменная;
- Свободная (не конкретизирована).

Какое предложение БЗ сформулировано в более общей – абстрактной форме: содержащее или не содержащее переменных? – Не содержащее переменных.

4.6 Что такое подстановка?

Подстановка - множество пар вида $X_i = t_i$, где X_i – переменная, а t_i – терм.

4.7 Что такое пример терма? Как и когда строится? Как вы думаете, система строит и хранит примеры?

Терм B называется примером терма A, если существует такая подстановка θ, что $B = A\theta$, где $A\theta$ – результат применения подстановки к терму.

Примеры строятся во время алгоритма унификации.