Российская академия наук
Сибирское отделение
Институт систем информатики
им. А. П. Ершова

Л.В. Городняя

ПАРАДИГМЫ ПРОГРАММИРОВАНИЯ

Часть 5
Учебные языки и системы программирования

Препринт
176

Новосибирск 2015

Препринт является пятой (завершающей) частью серии «Парадигмы программирования», посвященной исследованию парадигм программирования. Представлены результаты анализа особенностей учебных языков программирования. Содержание препринта представляет интерес для системных программистов, студентов и аспирантов, специализирующихся в области системного и теоретического программирования, и для всех тех, кто интересуется проблемами современной информатики, программирования и информационных технологий.

Siberian Division of the Russian Academy of Sciences
A. P. Ershov Institute of Informatics Systems

L.V. Gorodnyaya

PROGRAMING PARADIGMS

Part 5
Educational Programming Languages

Preprint
176

Novosibirsk 2015

The work describes research and specification of basic paradigms of programming. The author analyzes and compares special features of educational programming languages. A functional model to comparative description of implementation semantics of basic paradigms is proposed. The author proposes a scheme of describing and defining paradigm features of programming languages. The approach is illustrated with fragments of programming languages of different levels, which belong to machine-oriented, system, imperative, object-oriented, and productive programming.

ВВЕДЕНИЕ

Много лет назад корреспондент[1] газеты «Поиск» при обсуждении проблем компьютерного обучения задала ряд вопросов, включая следующие:

1. С какого возраста имеет смысл учить детей компьютеру?
2. Какие есть недостатки в нашей системе компьютерного обучения, и есть ли достоинства?
3. Про заграничный опыт: насколько он приложим на нашей почве, что стоит брать, что не брать?

Возможные ответы зависят от точки зрения на сущность понятия «компьютер» и цели обучения:

- Если компьютер понимать как готовый к использованию прибор, производимый за рубежом, и целью обучения считать его применение, то, по-видимому, достаточно игрового обучения, снимающего «стартовый барьер» перед техникой, что выполнимо в любом возрасте, почти одновременно с овладением грамотой, но целесообразно не форсировать такое обучение, чтобы не породить проблему повторного обучения при смене поколений техники, а «учить компьютеру» на уровне выпускных классов средних учебных заведений, когда приобретаемые навыки и знания уже не слишком консервативны.
- Если нацелиться на информатику как на часть культуры и науки, способствующую открытому развитию общеобразовательной системы и общества, то следует «учить компьютеру» всех независимо от возрастного и квалификационного ценза, чтобы не обострять проблему отцов и детей, а также сократить технологический разрыв с к экономически развитыми странами. При этом имеющим трудности в учебе детям необходимо предложить хорошие обучающие программные средства в индивидуальном порядке и пораньше, с дошкольного возраста, а общее освоение информатики отложить на уровень 7–8 класса, когда ученики склонны к критицизму. Этот уровень предшествует выбору профессии и специализации, и, следовательно, есть основания для надежды, что изучаемые элементы информатики будут распространяться в разные ви-

[1] Ольга Колесова

ды деятельности. Более раннее образование в области информатики станет целесообразным лишь по мере проверки временем содержания обучения.

- Если компьютер и его ПО рассматривать как индивидуально применяемый инструмент интеллектуальной коллективной деятельности и целью обучения считать проявление и развитие способностей личности в работающем коллективе, то общего и обязательного обучения компьютеру в школе, наверное, не надо вообще, а нужна факультативная система, какой была разработанная учениками академика А. П. Ершова система ШЮП (школа юных программистов) – летних, заочных, районных, в которой учатся строго на добровольной основе, учатся не только компьютеру, но и разработке программ, причем обязательно приходят к участию в коллективном проекте. «Учиться компьютеру» в такой системе лучше всего с 10–14 лет, когда многие дети уже научились учиться в школе, и факультатив им не помешает, к тому же обычно в этом возрасте дети любят играть в секреты, шифровки, изобретают игры, сюжеты, с удовольствием конструируют.

Системы компьютерного обучения пока еще нет. Есть лишь отдельные подходы и множественные эксперименты, потерявшие системообразующий фактор. Основной недостаток большинства подходов – нереальная оценка предстоящих трудозатрат и стремление к захвату или созданию «экологических ниш». Традиционно оттеснение квалифицированных специалистов, наивное доверие иностранным мощным фирмам, внедрение зарубежных разработок, непонимание, что технологии бывают отечественные. Главное достоинство – в самой попытке решения проблемы модернизации технологий через информатизацию системы образования. Это давало шанс, хотя он слабо реализован.

Другая опасность – увлечение методами контроля в ущерб методам и содержанию обучения.

Заграничный опыт не всегда приложим на нашей почве. Многие проблемы нашей системы образования за рубежом не представляют интереса, а потому маловероятно, что там они будут для нас решены [1, 3, 15, 24, 25, 28]. Например, заметный уровень информационной культуры в экономически развитых странах сложился в докомпьютерный период и, следовательно, компьютеризация образования там наследует этот уровень, опирается на него. В нашем случае система образования вынуждена формировать в учебном процессе необходимый уровень культуры, без которого компьютеризация не

может быть полезна [26, 32]. Впрочем, со временем технология производства обучающих средств, наверное, сможет достичь независимости от культурного ценза, но неясно, на благо ли нам это будет [3, 16, 21, 29].

Переход к параллельным вычислениям намного осложняет изначально весьма непростую задачу обучения программировании. Ознакомление с образовательными проблемами параллельного программирования и моделями параллелизма, встречающимися в учебных и экспериментальных языках и системах программирования, показывает, что в практике обучения программированию предстоит преодолеть дистанцию от уровня базовых средств управления взаимодействующими процессами до уровня разработки программ высокопроизводительных вычислений [9].

Мир программирования и его техническая основа претерпели значительные изменения за последние двадцать лет [2, 5–7]. Возрастает актуальность обучения параллельному программированию, что требует развития языковой и системной поддержки введения в программирование. В середине 1970-х годов активное исследование методов параллельного программирования рассматривалось как ведущее направление преодоления кризиса технологии программирования. Теперь рост интереса к параллельному программированию связан с переходом к производству многоядерных архитектур. Прежде всего следует упомянуть бурное развитие суперкомпьютеров и распределенных информационных систем, а также переход на многоядерные процессоры и массовое распространение многопроцессорных графических ускорителей.

19. ДЕТСКОЕ ПРОГРАММИРОВАНИЕ

Задача детского программирования – сформировать пороговый уровень, обеспечивающий предстоящее систематическое обучение программированию. Здесь не требуется концептуальное и инструментальное разнообразие, а нужен взвешенный подход к оценке сложности учебных задач по информатике и качества их решения, что требует комплектов ранжированных серийных задач с вариантами типовых решений, соответствующих разным требованиям к их качеству. Важной особенностью разработки такого комплекта является использование прикладной информатики подобно именованным числам при знакомстве с арифметикой в начальной школе. Именованные числа обосновывают абстрактное понятие числа при переходе к общему изучению математики. Прикладная информатика так же мо-

жет служить обоснованием основных понятий информатики, что усиливает методическую обусловленность техники решения задач по информатике.

Следует отметить, что привлечение прикладной информатики порождает определенные проблемы, вызванные эмоциональным отношениям к игровым, музыкальным и изобразительным возможностям компьютерных средств и предвзятыми оценками трудоемкости или удобства применения программного обеспечения. Чтобы преодолеть связанный с такими аспектами субъективизм, предлагается ранжировать задачи по структуре конструкций, используемых в типовых решениях, с учетом числа вариантов решения и некоторых их характеристик. Для оценки сложности задач, решаемых без применения компьютера, по-видимому, достаточно сравнения их по аналогии с задачами из ранжированных серий.

Применение компьютера быстро выводит некоторых учащихся за пределы пространства легко ранжируемых задач и типовых вариантов решений. Более сложные задачи характеризуются резким возрастанием неожиданных вариантов решений с разным уровнем алгоритмизации. Программирование и отладка решений таких задач – итеративный процесс, каждый шаг которого повышает уровень изученности решаемой задачи и уточняет пространство ее возможных решений. На разных фазах этого процесса в зависимости от достигнутого уровня изученности действуют свои критерии качества программы, что увеличивает число оценочных критериев, влияющих на оценку решения задачи, и даже провоцирует преждевременный переход к производственным критериям. Для ориентации в таких оценках предлагаются рекомендации по построению вспомогательных ситуаций, в которых можно продемонстрировать роль выбранного критерия. Это позволяет сравнивать качество вариантов решения задачи относительно набора критериев.

Каждая серия задач символизирует собой восходящий процесс пошагового уточнения пространства изучаемых решений, начиная от визуализации результатов до полного вывода их из входных данных, включая удобство отладки. Основные шаги уточнения: построение результирующих текстов и изображений, прием входных данных и параметров, линейная обработка данных, ветвления, укрупнение используемых понятий (процедуры), структуры данных, повторяемость (циклы), унификация ввода-вывода (файлы). Критерии качества решения задачи: адекватность, корректность, удобочитаемость, понятность диалога, удобство отладки, эффективность, развиваемость, надежность, полнота и другие, становятся ясными при использовании простых ЯП, поддержанных удобными СП учебного назначения [21,37]

19.1. Logo

Язык начального обучения программированию (ЯНОП) Logo разработан в 1972 году С. Пейпертом [19]. В этом языке унаследованы многие идеи языка Lisp [12], ознакомление с которыми выглядит как управление рисующим роботом-черепашкой. Образ «черепаха» возник не случайно. С конца 1940-ых годов выпускалась игрушка – программируемая черепаха, способная ходить через лабиринт. Управлялась черепаха с помощью перфокарты, вставляемой в прорезь на спине.

По мере овладения азами программирования на базе Logo происходит переход к задачам уровня «создай свой язык». Со временем появились конструкторы Log-Lego, приспособленные к созданию конструкция с разными функциями, ещё более образно представляющими возможности конструктивного программирования.

19.2. Karel

Появившийся в начале 1980-ых язык Karel назван в честь Карела Чапека, предложившего слово «робот» [39]. Учебный мир для начального ознакомления с программированием устроен как управление роботом Karel, устанавливающим звуковые сигналки в разных местах сетки улиц и проспектов. Робот вооружён тремя видеокамерами, компасом и корзиной для сигналок. Видеокамеры позволяют видеть установленные сигналки. У робота имеется механическая рука для сбора и установки сигналок.

Учащийся должен, всмотревшись в заданную ситуацию на сетке улиц и проспектов, запрограммировать робота так, чтобы он выполнил задание по установке или удалению сигналок.

Синтаксис языка Karel подобен языку С. Реализация выполнена как библиотека <karel.h>. Программируя функции можно использовать кроме встроенных команд и предикатов рамочные конструкции if, case, iterate, while, do … while. В качестве отладочных средств используются комментарии, вывод сообщений, установка контрольных точек и звуковые сигналы.

19.3. Робик

Идея программируемых роботов в языке начального языка программирования РОБИК, созданного Г.А. Звенигородским, обобщена до уровня коллекции роботов, каждый из которых обладает своей системой команд, но при этом они могут выполнять общую программу. Такой подход позволял определять специализированных роботов на каждый класс задач, наце-

ленный на освоение конкретных понятий. Выбор роботов обеспечивал пошаговое методически обусловленное, возможно, безмашинное изучение основных концепций программирования с последующим переходом к применению стандартных языков программирования. Мостиком такого перехода служили специальные синтаксические диаграммы, позволяющие перелагать программы на разные языки практически без синтаксических ошибок [21].

19.4. Grow

Несколько в стороне от магистральной линии обучения программирования лежит идея учебного конструктора игр Grow, созданного в начале 1980-х годов в Калифорнии. В этом языке конструирование программ допускает использование при организации ветвлений текстовых шаблонов и вероятностей срабатывания ветви, что позволяет снизить стартовый барьер понимания предикатов [41].

19.5. Начала параллелизма

Пришло время изучать информатику и программирование, начиная с мира параллелизма. Современные наблюдательные люди знакомы со многими явлениями этого мира через Интернет-общение, компьютерные игры, многоядерные ноутбуки и графические ускорители и, наконец, через применение бытовых приборов с электронным управлением (мобильные телефоны, фотоаппараты, телевизоры, стиральные машины, микроволновые печи, автомобили и пр.). Они замечали, что выполнение «одинаковых» действий обладает разной длительностью, что длительность реагирования информационной системы может зависеть от текущей ситуации, что собственно выполнимость действий зависит от не всегда заранее известных условий, что системы можно и нужно настраивать, что ряд систем могут работать одновременно и влиять на работу друг друга. Им известно, что существуют кэши, протоколы, верификаторы, резервное копирование, транзакции, «гонки» данных, «смертельное объятие» и многое другое. Они наслышаны об успехах любительской астрономии, перспективах GRID-технологий и «облачных» вычислений. Активизация таких знаний образует основу для быстрого изучения средств и методов параллельного программирования, начало которому дает предлагаемый экспериментальный курс.

Более осознанно и реально с параллельными процессами можно встретиться при работе на уровне операционных систем (ОС), при организации

практики на базе суперкомпьютеров, при сетевой обработке данных, при компиляции учебных программ и т. п. [23, 30]

Проблемы обучения параллельному программированию осложнены дистанцией между уровнем абстракций, в которых описываются решения сложных задач на современных языках программирования (ЯП), и уровнем реализации конкретной аппаратуры и архитектурных принципов управления параллельными вычислениями (потактовая синхронизация, совмещение действий во VLIW-архитектурах, сигналы, семафоры критических участков, рандеву и т. п.). Проблемы подготовки параллельных программ для всех столь разных моделей обладают общностью, но есть и существенная специфика, требующая понимания разницы в критериях оценки программ и информационных систем для различных применений.

Предстоит разработка учебного языка высокого уровня, ориентированного на параллелизм, исследование возможностей языковых средств, обеспечивающих при работе с функциями пространство аргументов, подобное пространству итерации, а также типизацию действий и схем, поддерживающую вычисления, реорганизацию памяти, изменение памяти и, кроме того, учет длительности действий, мемоизацию, конструирование потоков, динамический учет производительности и др.

Следующий шаг – разработка языка высокого уровня параллельного программирования, приспособленного к ознакомлению с более сложными моделями вычислений и с методами верификации программ, без которых надежность параллельного программирования весьма проблематична. Для нужд этого шага потребуется более строгая формализация реализационной семантики в трансформационно-операционном стиле. Представленный в части 4 подход к определению языка параллельного программирования может поддержать эксперименты по разработке новых языков, ориентированных на учебно-исследовательские проекты в области создания распределенных информационных систем.

Рассмотрим экспериментальный курс для раннего ознакомления с понятиями и явлениями параллельного программирования без опоры на опыт практического программирования. Важная часть курса – профилактика жесткого привыкания к принципам традиционного последовательного программирования. Более широкое понимание программирования необходимо для специализации в области разработки распределенных информационных систем, а также приложений для суперкомпьютеров, многопроцессорных конфигураций и графических процессоров.

Сложность перехода от навыков последовательного программирования к организации параллельных процессов в значительной мере обусловлена

системой обучения, требующей все упорядочивать, выстраивать в однозначные, безальтернативные построения без выражения зависимости принятых решений от выбора структур данных и последовательности действий по их обработке.

В учебном языке обучения параллельному программированию желательно разделить последовательность вычислений и последовательность размещения их результатов в элементах памяти. Последовательность вычисления не обязана совпадать с последовательностью размещения вычисленных значений. Основные структуры данных – это очереди, элементы которых доступны последовательно и размещены рассредоточенно, новые добавляются в хвост очереди, и векторы с обычным индексированием элементов, размещенных в соседних регистрах, допускающих произвольный доступ. Скобки символизируют порядок размещения, а знаки – порядок вычисления. Таким образом, «( )» и «;» символизирует последовательность, а « [ ]» и «,» – произвольный порядок или доступ.

(A, B, C,…) – очередь из элементов, порядок вычисления которых не определен.

(A; B; C;…) – очередь из элементов, вычисляемых в порядке записи.

[A; B; C;…] – вектор из элементов, вычисляемых в порядке записи.

[A, B, C,…] – вектор из элементов, порядок вычисления которых не определен.

Параллельное программирование на уровне процессов выглядят как нечто среднее между программированием на макроассемблере и языках высокого уровня. Различие проявляется в понимании данных и команд. В результате разработка программ для организации взаимодействия процессов отличается от подготовки обычных последовательных программ на весьма глубоком уровне, что можно показать на модели интерпретирующего автомата для языка управления процессами. Такой автомат требует реализации дополнительной таблицы событий и структуры данных для очередей, регулирующих доступ к данным.

Такую усложнённую картину может смягчить методика пошагового обучения, основанная на серии специально подобранных задач. В качестве примера можно привести Новосибирский опыт раннего обучения программированию [3, 21].

В середине 1980-х годов вышла в свет сразу ставшая библиографической редкостью книга Г. А. Звенигородского «Первые уроки программирования», вводившая школьников в мир программирования для ЭВМ. В этой уникальной книге была выстроена практичная схема начального ознакомления с основными понятиями программирования, причем последователь-

ность изложения была тщательно методически обусловлена постановками задач, показывающими недостаточность ранее освоенных средств. Каждый шаг усложнения понятий на недоуменный вопрос ***Зачем?*** содержал ответ: ***Чтобы решать вот такие задачи!***

Полученная тематическая лестница введения в программирование имела следующий вид:

Таблица 1

Последовательность начального ознакомления с концепциями программирования

Для начального ознакомления с понятиями параллелизма тематическая лестница приобретает вид

Таблица 2

Последовательность начального ознакомления с феноменами параллельного программирования

Следует отметить, что система понятий языка начального обучения программированию Робик изначально была приспособлена, содержала резерв, для изучения параллельных программ, но этот резерв не был востребован. В наши дни он обретает актуальность.

Чуть позже увидел свет перевод на русский язык превосходной книги блестящего авторитета Т. Хоара «Взаимодействующие последовательные процессы», в которой на простых задачах управления автоматами показано, что параллельные композиции внешне не сложнее последовательных, без акцента на том, что отладка параллельных процессов много сложнее, чем последовательных. Поэтому здесь примеры Хоара дополнены рассмотрением вопросов отладки и методов минимизации ее объема с помощью выделения типовых, многократно используемых фрагментов и схем. В качестве примеров многопоточных программ используются задачи по управлению автоматами, при программировании которых используются потоки, счетчики, барьеры [34].

История ЯВУ и ЯСВУ дала ряд работоспособных идей по эффективной компактизации представления естественного параллелизма при серийной обработке сложных структур данных. Так, APL –просачивание скалярных

операций на произвольные векторы, Setl – использование множеств и кванторных формул, дающих эффект пространства итераций [44], Sisal – выделение участков с однократными присваиваниями, пространства итераций и упаковка их результатов, БАРС и Поляр – сети управления процессами и определение дисциплины доступа к памяти, mpC – представление многопроцессорных конфигураций при выполнении многопоточных программ с синхронизацией потоков в терминах барьеров. Эти идеи образуют основу понятий о параллельной обработке очередей и векторов. Кроме того, они распространены на механизм применения функций и операций – применение тоже операция.

При проектировании курса начального обучения параллельному программированию учтены факторы успеха наиболее известных учебных языков программирования, таких как Basic, Pascal, Logo, Grow, а главное – методически обусловленное введение программистских понятий в языке начального обучения программированию Робик [1, 10, 17, 19, 21, 41].

Ряд моделей параллельных процессов концептуально сродни недетерминизму. Многие из них строятся как коллективное поведение объектов – автоматов с состояниями, изменение которых рассматривается как вычислительный процесс [34].

Термин «процесс» используется для обозначения поведения «объекта».

Описание процесса начинается с определения класса событий, представляющих интерес для участвующих в нем объектов. Множество имен событий, используемых при описании процесса или объекта, называют алфавитом.

Первая абстракция при моделировании процессов – исключение времени, т.е. отказ от ответов на вопрос происходят ли события строго одно за другим. Это обеспечивается следующими договоренностями:

- элементарные действия исполняются мгновенно;
- протяженное действие: всегда пара событий – начало и конец;
- нет точной привязки действий к моменту времени;
- определены отношения «раньше–позже», «одновременно», «независимо»;
- совместность событий понимается как отношение «синхронизация»;
- одно событие из независимых возникает в любом порядке, без причинно-следственной связи.

Если известно, что процесс P может произойти вслед за событием x, влекущим этот процесс, то событие x называют префиксом процесса P.

x → P -- P за x

Обычно для решения конкретной задачи вводится специальный объект – автомат, способный реагировать на определенное множество событий в зависимости от состояния автомата. Поведение объекта, который способен действовать, можно описать в виде системы правил, внешне похожих на уравнения. Решением такой системы уравнений являются процессы, которые может выполнить автомат.

Пример 1. Рассмотрим автомат, установленный в здании НГУ на 4-ом этаже, продающий кофе.

Торговый автомат: {деньги, кофе}
(деньги => (кофе => автомат))

- нет кофе без денег
- нет двойных порций

сломанный автомат - нет кофе

(деньги => СТОП)

- нет действий = нет событий

Пример 2 [32]. Рассмотрим простые задачи типа перемещения фишки на клетчатой доске по свободным позициям.

X		
O		X

На данной доске фишка может двигаться лишь вправо или вверх. При множестве событий – { вправо, вверх } правильный процесс можно специфицировать как:

(вправо вверх вправо => СТОП)

Пример 3. Простейшая модель часов – типовой пример рекурсивного описания процессов.

Часы: { тик-так }

(тик-так - часы) - рекурсия

Пример.4 [32]. Небольшое изменение разрешенных позиций на клетчатой доске показывает возможность выбора вариантов хода событий при выполнении процессов. Появляется понятие «меню».

	X	
O		

(вверх => СТОП
| вправо вправо вверх СТОП)

| - представление вариантов в меню процесса

Обычно выделяют начальное меню процесса. Несложно выразить выбор из произвольного числа альтернатив и взаимную рекурсию. При этом достаточно естественно используется понятие "переменная", что существенно расширяет возможности записи систем уравнений.

Рассматривая примеры работы с одним автоматом, не видно причин для выхода за пределы обычных последовательных процессов. Просто лишь выделен ряд понятий, которые позволят потом гладко перейти к описанию процессов, выполняемых взаимодействующими автоматами.

Формулировка ряда закономерностей позволяет разделить описание процесса на уровень спецификации и реализации. Например: «Если для всякой альтернативы меню дальнейшего поведения совпадают, то процессы тождественны».

Сравнивая запись меню процесса с представлением текстов языка, можно обратить внимание на их подобие. Отличие заключается в отношении к одновременности существования элементов. Во многих моделях процессов используются множества текстов или языки как характеристика, показывающая разнообразие вариантов хода событий.

Реализация процессов

Следуя Хоару [32], представим, что события можно реализовать как атомы языка Лисп, а процессы определять как функции, задающие реакции на события. Структуры данных стандартных языков программирования, таких как Паскаль-Си, даже при ООП-расширении менее удобны для расширяемой реализации, полезной при обсуждении понятий.

В таком случае реализация процесса – это функция, определяющая ход процесса по начальному событию. Таким событием может быть, в частности, готовность входных данных. Функциональная модель представления процессов позволяет легко описывать и взаимодействие процессов в виде функционалов, т.е. функций над процессами, представленными как функциональные переменные.

При определении взаимодействий используется понятие "протокол". Протокол – это последовательность символов, обозначающих произошедшие события.

Особый интерес представляют монотонные функции над протоколами, допускающие аналитику прогнозирования поведения и доказательные построения. Важное направление анализа – проверка соответствия объектов спецификации процесса.

Взаимодействие параллельных процессов

Разнообразие схем взаимодействия процессов требует особой заботы об экономии концепций, понятий и их представлений. Функциональный подход к программированию ценен именно возможностью унификации понятий, различие которых малосущественно с точки зрения их реализации. В этом плане можно не придавать значения разнице между процессами, объектами, системами и их окружениями. Все это определенные процессы с заданной схемой взаимодействия и разными предписанными ролями в более общем процессе информационной обработки.

Взаимодействие – это события, в которых требуется участие ряда процессов. При необходимости взаимодействие процессов может быть пошагово синхронизовано. Представляя процессы функциями, взаимодействия процессов естественным образом могут быть заданы как взаимосвязанные рекурсивные функции.

Именование и пометка процессов дают основания для удобной связи символики представления процессов с текстами программ, порождающих процессы.

Дальнейшее развитие базовых понятий, используемых при организации параллельных процессов связано с привлечением недетерминизма и общих разделяемых ресурсов.

В качестве примеров, показывающих типовые схемы взаимодействия, принятые в основных областях задач параллельного программирования предполагается рассмотреть однородные вычисления, такие как задачи на шахматной доске или обработка векторов или матриц, асинхронные процессы, такие как задачи про философов и читателей-писателей, а также приемы синхронизации и оптимизации, описанные в книге Хоара [34]. Некоторые примеры рассмотрены в четвёртой препринта, посвященном языкам параллельного программирования и высокопроизводительных вычислений.

Учебная демонстрация проблем программирования взаимодействующих процессов показана на примере исполнителя “Машинист” ЯНОП Робик [21].

Роль параллелизма в современном программировании распределенных систем подробна рассмотрена [9].

Покажем простейший пример автоматизированного перехода от последовательных процессов к параллельным.

Условный оператор	*Эквивалентное выражение, приспособленное к распараллеливанию*
if X then Y else Z	(x * y) + (~x * z)

Пример 5. Распараллеливание. Параллельная реализация ветвления.

Проблемы разработки, отладки, тестирования и верификации параллельных программ по сложности превосходят обычное программирование [36]. Именно здесь сосредоточена исследовательская активность современного языкотворчества и системного программирования, поиск средств и методов интеграции удачного опыта параллельного программирования и успешного обучения методам параллельных вычислений.

20. УЧЕБНОЕ ПРОГРАММИРОВАНИЕ

Рассмотрим наиболее очевидные особенности языков программирования, рекомендуемых в качестве типовых при изучении основ информатики, оценивая удобство применения в учебном процессе систем программирования для этих языков на современной технике.

Исторически в разных странах сложились разные принципы преподавания информатики, обусловленные общим укладом жизни большинства населения. В частности, в США характерна большая пропасть между программистами и рядовыми пользователями. В программном производстве собственные студенты-информатики не котируются, популярно привлечение иностранной молодёжи. Тем не менее лозунг «Дети не могут ждать!» сделал своё дело, повлёк широкое внедрение компьютерных игр и разработку компьютеров для домохозяек. Особо следует отметить традицию избыточного доверия инструкциям, в то время как не все ошибки диагностируются компилятором, т.к. существуют «правила», которых программист обязан придерживаться, а он это не всегда делает. В России такие ошибки обнаруживаются сразу.

Совсем иная картина в Японии, остерегающейся нарушать традиции и экспериментировать с любимыми детьми. Детские компьютеры обычно содержат эффективную аппаратную реализацию ЯВУ без упора на внедрения компьютеров в домашнем хозяйстве и обучении. Предпочтение отдается компьютерной графике. Имеет место государственная поддержка совершенствования передовых технологий.

Для Англии и Франции характерна тенденция к снижению возраста, с которого в школах начинают обучение информатике. Обычно это преподавание языков Logo и Lisp, удобных для постановки учебных задач на алгоритмику. Практикуется дополнительная оплата труда педагогам за работу по освоению компьютеров и методики их использования в учебном процессе, включая парное преподавание.

В Скандинавии предпочитают информатику преподавать не как самостоятельную учебную дисциплину, а только как инструмент в рисовании, музыке, текстовой обработке и т.п.

Для Болгарии был успешно выполнен эксперимент по ранней модернизации образования, вплоть до полной перестройки системы образования. Учебные дисциплины поделены на три цикла: гуманитарный, точные науки и художественные, что позволило отменить избыточную номенклатуру отдельных предметов и выстроить комплексное обучение. Следует отметить, что болгарские программисты на мировом рынке котируются очень высоко.

В начале 1990-х консультанты UNESCO обратили внимание, что «Информатика свободна от культурных напластований», таким образом, она объединяет мир, а значит, с помощью информатики можно повысить жизненный уровень слаборазвитых стран. Выработаны рекомендации по постановке обучения программированию в терминах Logo-миров.

В нашей стране существовал централизованный план информатизации учебных заведений с приоритетом введения информатики и всеобщего оснащения техникой. Преподавание нацелено на понимание алгоритмов и умение оценивать оптимальность программируемого решения.

20.1. Сравнение языков программирования Бейсик, Паскаль, Рапира и Лисп

При организации процесса представления, хранения и обработки информации язык программирования определяет грань между ручным трудом программиста и возможностью автоматизации этого процесса Естественность языковых средств и удобство их применения на практике по существу определяет трудоемкость работы, возникающей при изучении используемых средств и решении с их помощью разных задач. Поскольку допустимые трудозатраты всегда ограничены вполне реальными силами, ресурсами, временем, постольку ограничена принципиально сложность решения задач с помощью определенных языков. Поэтому, выбирая производственный язык программирования, мы фактически выбираем и класс задач, в пределах которого сможем работать, и пространство решений, которые сумеем представить, точнее построить из конструкций выбранного языка программирования.

Учитывая, что при знакомстве с новыми явлениями обычно происходит сильное запечатление первых впечатлений, от которых трудно перейти к идеям, не соответствующим или противоречащим этим впечатлениям («Первое слово дороже второго», «эффект второго языка» – замечено, что особенно трудно освоить второй естественный язык и второй язык программирования и т.п.), можно сделать вывод, что выбирая учебный язык программирования, мы выбираем объем работы, которую придется проделать нам и нашим ученикам, чтобы решить намеченные задачи [45,47].

Каждый язык программирования формируется вокруг определенных целей, в конкретной обстановке, на некотором уровне понимания возможностей компьютера и содержания работы программиста, на фоне системы вкусов и предпочтений, сложившихся ко времени создания языка [31,38]. Поэтому по мере расширения сферы применения компьютера и уточнения спектра ее возможностей возникает вполне закономерная потребность в построении новых языковых средств. Это не всегда означает, что прежние средства категорически не состоятельны. Чаще это симптом несоответствия новым задачам традиционной реализации языковых средств и методики их применения. Но практически язык программирования воспринимается неразрывно с системой программирования и привычной методикой ра-

боты. Следовательно при выборе языка программирования для любой области применения необходимо проанализировать не только собственно язык - определение и объем подготовленных на нем программ, но свойства доступной системы программирования и методику основной работы с системой и программами.

Язык Паскаль создан для обучения основам программирования студентов высших учебных заведений [8, 17, 18, 22]. Вузовский цикл преподавания (лекция, семинар, практика) существенно повлиял на структуру языка и методику его преподавания. Основы программирования сводятся к задаче представления программ, техника представления программ оттачивается на стереотипном классе вынуждено решаемых при программировании на Паскале задач по обработке векторов и текстов, выработаны удобные для преподавателя приемы оформления программ. Для самостоятельной работы требуется предварительное накопление некоторой «критической массы» знаний о языке и его программном окружении.

Язык Бейсик возник как практическое средство для быстрого привлечения практикантов и лаборантов к работе на компьютере [10]. Он ориентирован на доступ к основным возможностям оборудования при выполнении разовой работы, не рассчитанной на многократное повторение, на долгую жизнь программ, и не требующей поэтому особой квалификации от исполнителя или качества результата. По этой причине типовая реализация Бейсика – это небольшая операционная система, в которой Бейсик является и языком заданий, и языком программирования, часто единственным. Начальное знакомство с такой системой просто и приятно, но переход к систематической работе усложнен необходимостью наработки профессиональных приемов по организации программ. В случае Паскаля такие приемы предложены в языке: процедуры, функции, области действия имен, передача значений с помощью параметров, конструирование типов данных. Бейсик является самым легкодоступным языком на микропроцессорной технике. Часто он встроен в персональный компьютер, поэтому начальное знакомство возможно при минимальной конфигурации оборудования.

При разработке языка Рапира предпринята попытка объединить достоинства ранее сложившихся учебных языков программирования, избежав их недостатков по отношению к задаче преподавания основ информатики учащимся средней школы. Язык Рапира сформировался в процессе факультативной работы со школьниками разных возрастов в условиях регулярного доступа к компьютеру. Типовая реализация, как и в случае Бейсика, обеспечивает простоту начального знакомства. Языковые средства органи-

зации программ, как и в случае Паскаля, обеспечивают переход к систематической экспериментальной работе, но с помощью средств, более полно соответствующих возрастным особенностям обучения школьников [4,29][2].

Универсальный язык программирования Pure Lisp возник как язык обучения специалистов, ведущих исследования проблем искусственного интеллекта, методам символьной обработки [42]. И в наше время его построения составляют ядро многих ЯП и воспроизводятся в новых СП. Кроме того, появляются новые реализации языка Lisp, предоставляющие СП с современными возможностями. Яркий пример – разработка Б. Л. Файфеля, предоставляющая типичные возможности Widows-интерфейса, Интернет-оперирования и компьютерной графики в системе Home Lisp [48].

«Стартовый барьер» перехода к практике на компьютере

Для первой лабораторной работы по Бейсику достаточно познакомиться с простейшим вариантом оператора вывода и записью констант и операций. Это позволяет изучить особенности обработки значений и их границы. При этом возможно оперативное получение результатов каждого шага работы, что исключает случайное накопление ошибочного понимания.

```
print 123.5 + 3.7 - 2.9 * 11.16 / 3
```

Аналогично в Рапире:

```
вывод: 123.5 + 3.7 - 2.9 * 11.16 / 3
```

или

```
? 123.5 + 3.7 - 2.9 * 11.16 / 3
```

Аналогично Lisp

```
(print (/ (* (+ 123.5 3.7 - 2.9) 11.16) 3)
```

Или

[2] Учебно-производственный язык программирования Рапира был реализован в составе программной системы Школьница на ПЭВМ Агат, переданной для распространения Таллинскому СФАП. Разработаны системы программирования для языка Рапира на ПЭВМ Ямаха (перенесена на Корвет) и на СМ-4 (перенесена на Электронику УКНЦ).

```
(print (/ (* (- (+ 123.5 3.7) 2.9) 11.16) 3)
```

Такая работа на Паскале требует знакомства с оформлением программы и понимания, что Паскаль-программа сначала транслируется, затем исполняется, для чего следует набирать соответствующие команды. Даже если предусмотрена возможность задать одной командой трансляцию и исполнение, в дальнейшем при переходе к работе с разными данными придется отделить трансляцию от счета из соображений эффективности, чтобы не тратить свое время на ожидание конца трансляции. Не забудьте, что в конце текста программы на Паскале обязательно должна стоять точка!

```
program PROG ;
 begin
  writeln ( 123.5 + 3.7 - 2.9 * 11.16 / 3 )
 end.
```

Обнаружение ошибок

Несоответствие программы определению изучаемого языка или невозможность исполнения в случае Бейсика показывается диагностическим сообщением с указанием номера строки, содержащей ошибку. Сообщение обычно содержит указание типа ошибки, естественно, на английском языке.

```
10 print 2+3+
run
Missing operand in 10
```

Это значит, что в 10-й строке пропущен операнд. В случае Паскаля появится примерно такое же сообщение, только без номера строки. По нажатию клавиши "конец набора" (Enter) курсор установится на строке, содержащей ошибку, в позиции, где она обнаружена.

```
Missing operand

program PROG ;
 begin
  writeln ( 2+3+ )
                ~
 end.
```

(Только сообщение на экране не видно одновременно с ошибочным текстом. Его придется запомнить.)

В случае Рапиры появится сообщение на русском языке. На экране Ямахи оно высвечивается в особом окошке. Если хорошо подобрать размеры окон для текста программы и процедур, то можно сообщение и контекст ошибки увидеть на экране одновременно и следовательно - не запоминать.

```
 _____________________
| Требуется операнд |
 -------------------
  вывод: 2+3+
                ~
```

При желании и для Паскаля нетрудно обеспечить русскую диагностику. Достаточно всего лишь сменить файл с текстами сообщений, что кое-кто сделал для себя, но не имеет возможности распространять свои результаты.

Лисп

«+» мультиоперация, может иметь любое число аргументов, в том числе ни одного. Поэтому для ошибки такого рода просто нет места. При отсутствии операндов значение – «0».

При необходимости диагностика выдается на том языке, на который настроена система.

Коррекция текста программы

Чтобы исправить ошибку в тексте на Бейсике надо с помощью специальной команды «list» вывести на экран строки, содержащие корректируемый текст.

```
list 10-20
10 print 2+3+
OK
```

Появился текст – надо высмотреть в нем ошибку и после ее исправления (стереть лишний плюс или добавить еще одно число и, главное, не позабыть про «ритуальную» клавишу «конец строки», иначе вопреки изображению на экране исполнится прежний ошибочный текст) можно сразу повторно исполнить программу и заняться следующей ошибкой или увидеть результат работы программы.

```
10 print 2+3+4
run
9
OK
```

В тексте на Паскале или на Рапире курсор автоматически устанавливается в позиции предполагаемой ошибки. После исправления текста на Рапире можно сразу исполнить программу повторно, а на Паскале – оттранслировать и затем исполнить.

Лисп

При использовании системы GNU Clisp можно использовать UNIX-технику коррекции строк, если их немного, или редактировать программу обычным текстовым редактором, который при желании можно встроить в систему [40, 43].

Организация программы

В языке Бейсик программа – это набор пронумерованных операторов, взаимодействующих с помощью переходов по этим номерам. Незанумерованные операторы исполняются немедленно в диалоге. Набор номера является сигналом, что предстоит сохранить программы в памяти КОМПЬЮТЕР. Чтобы исполнить программу, достаточно набрать незанумерованную обычную команду языка RUN (при работе в режиме AUTO придется стереть автоматически появляющийся номер, что новичку еще предстоит сообразить).

В Паскале программа предоставляет собой целостную систему вложенных блоков с выделенной головной программой. Программа транслируется и исполняется с помощью особых команд системы или интерфейса.

Язык Рапира определяет программу как одноуровневый набор процедур и функций, каждая из которых пишется как бы на отдельной странице. Страницу можно найти по имени записанной на ней процедуры, если надо посмотреть, как она выглядит, поправить текст или исполнить процедуру. Исполнение осуществляется непосредственно в диалоге или из других процедур предписанием вызова. Аналогичные действия выполняются и для функций, только их вызовы выглядят как формулы в выражениях.

В Лиспе программа представлена последовательностью выражений.

Элементарные данные и операции

Элементарные типы данных и набор операций над ними задают в языке программирования основную область приложения языковых средств. Формально Бейсик, Паскаль и Рапира обладают общей областью приложения – это обработка целых и вещественных чисел и текстов с помощью общего набора операций и встроенных функций. Но типовые системы программирования для этих языков обладают заметными различиями, существенными для учебного процесса:

В Лиспе основной тип данных – список, строящийся из атомов, но поддерживается работа с числами произвольной длины и строками на правах атомов.

В Паскале принято целые и вещественные числа рассматривать как объекты принципиально различной природы, что требует специальных пояснений, трудно воспринимаемых учащимися. Кроме того, в учебных экспериментах приходится сразу знакомиться с особенностями работы на границах диапазона целых, т.к. он невелик, что методически не обусловлено для учащихся средней школы, является преждевременным усложнением.

Бейсик допускает числа в более широком диапазоне, причем строгого разделения на целые и вещественные не предусмотрено, поэтому длинные целые автоматически переводятся в форму с плавающей запятой, что расширяет допустимый диапазон целых чисел в программах. Значения параметров цикла for особо ограничены.

Рапира также обеспечивает обработку чисел без строго разделения целых и вещественных. В некоторых реализациях снято фиксированное ограничение на длину представления чисел. Естественным ограничением является общий объем памяти, необходимой для работы программы. Поэтому допускается свободный учебный эксперимент, выполнимый без знания технических деталей организации процесса обработки данных на компьютере. При вводе данных в диалоге с программой возможны некоторые ограничения длины набора строкой экрана (на СМ-4 и Электронике УКНЦ) или форматом экрана (Школьница на АГАТе) или общим наличием памяти (на Ямахе).

Работа с текстовыми строками в этих языках различается незначительно (особенностями в выборе подстрок).

Лисп традиционно поддерживает работу с целыми без ограничения длины, обработка вещественных чисел может быть связана с указанием требуемой точности вычислений, кроме того, имеются рациональные чис-

ла, выглядящие как пара из числителя и знаменателя (1/2). Работа со строками допускает переход к использованию других структур данных.

Идентификация

Обычно язык программирования позволяет обозначать значения и функции с помощью имен, выбираемых программистом. Свобода в выборе таких имен, возможность систематизировать обозначения, связывать их с мнемоникой естественного языка существенно повышает удобство работы преподавателя с программой и упрощает долговременную экспериментальную работу. Специалисты по эргономике утверждают, что русскоязычные программисты проигрывают 30% производительности труда из-за вынужденной привязки к английскому языку.

В Бейсике имена различаются по малому числу литер (часто 2) и для текстовых данных содержат специальный символ-флаг, отличающий их от имен чисел. Это упрощает Бейсик-интерпретатор, но требует специальных пояснений при изучении, а также специальных навыков по обозначению данных небольшим набором различимых имен (обычно допустимы только латинские буквы).

```
N1  ST  ST$              '--- последнее - имя строки

LETTER  LENGTH  LE   '--- неразличимые имена
```

Паскаль предлагает активное использование длинных имен из латинских букв, почти фраз без пробелов между словами (Turbo-Pascal допускает подчерки вместо пробелов), что удобно преподавателю при чтении ученических программ. (Но во многих реализациях Паскаля это удобство ослаблено тем, что имена фактически различаются по первым 6-8 символам.)

```
NAMEOFVERYLONGSTRING        (*--- имя очень длинной строки *)

NAME_OF_VERY_LONG_STRING   (*--- можно в некоторых реализациях*)
```

В Рапире также активно используются длинные имена, но разумеется из русских букв, наряду с латинскими, если надо. (Определение языка требует различения имен по полному набору образующих их символов.)

```
ИМЯ_ОЧЕНЬ_ДЛИННОЙ_СТРОКИ  //--- можно русские фразы
```

```
hardware_book                    //--- можно и английские
```

В Лиспе нет ограничений на длину идентификатора, а используемый алфавит может быть расширен почти любыми символами, не мешающими распознаванию структуры списка, т.е. пробелы, точка и круглые скобки.

Области действия имен

Имена в программе на Бейсике глобальны. Поэтому в случае сколько-нибудь больших программ приходится согласовывать их использование, что требует специальных навыков.

В Паскале имена локализуются по статически вложенным блокам, благодаря чему возможна согласованная разработка независимых частей программы. Согласование необходимо для совместно используемых имен в статически объемлющих блоках.

В Рапире вложенных блоков нет, но автоматически локализуются имена параметров при вызове функций (процедур). Локализацией других имен можно управлять с помощью специальных предписаний, задающих способ поиска значения имени в зависимости от цепочки ранее вызванных незавершенных функций и процедур.

Лисп предпочитает локализацию имён переменных и функций, но поддерживает определение глобальных функций с возможностью управлять методами локализации имён.

Описание типов данных

Переменной в Паскале присваиваются значения одного заранее определенного типа, описываемого при объявлении переменной в спецификации блока. Это позволяет на этапе трансляции осуществлять контроль типов данных при присваиваниях до исполнения программы, что в случае больших программ и/или данных экономит машинное время и упрощает отладку.

```
program PR ( output );
 var N, L, M : integer;   (* имена целых *)
    C, H, S : char ;      (* имена литер *)
  begin
- - - - - -
   N := 1;
   S := '&';
```

```
- - - - - -
 end.
```

Переменные, обозначающие элементарные данные, в Бейсике можно не описывать. Но к именам переменных, обозначающих строки, приписывается специальная литера "$", чтобы отличать их от переменных, обозначающих числа. Обязательно объявляются заранее переменные, обозначающие массивы. Контроль типов данных при присваивании выполняется при исполнении программы.

```
10 string$ = "string"
20 number$ = "1234567890"   '--- присваивания строк

30 string = 123.45
40 number = 12345               '--- присваивание чисел

50 dim A (5)              '--- объявление массива и
60 A(1) = 120             '--- присваивание его компоненту
```

Имена в Рапире и Лиспе могут обозначать значения любого типа без предварительного объявления, т.е. вводятся по мере необходимости.

```
СИМ := "&"
НОМЕР := 124
ПИ := 3.14
ГЛАСНЫЕ := "аеиоуызюя"

ВЕКТОР := < 2 , 4 , 6 , 9 >   \\--- присваивание кортежа
ВЕКТОР [ 1 ] := 8             \\--- изменение его компоненты
A := ВЕКТОР                   \\--- пересылка кортежа
A [ 3 ] := 120
```

В Лиспе нет оператора присваивания – значения переменных задаются при вызове функций.

```
( (lambda (СИМ НОМЕР ПИ ГЛАСНЫЕ) () ) "&" 124 3.14 "аеиоуызюя" )
```

Распределение памяти

В Паскале распределение памяти выполняется на этапе трансляции. При недостатке ее перераспределение можно организовать в программе с помощью встроенных процедур отказа от определенных ячеек памяти (dispose) и повторного их использования (new).

В Бейсике память под данные отводится при обработке объявления массивов (dim) и по мере появления имен переменных. Перераспределение памяти выполняется как отказ от ненужных данных (erase).

В Рапире и Лиспе память под данные отводится по мере формирования значений, в том числе и составных – кортежей или списков. Освобождающаяся при этом память автоматически используется повторно, что не требует специальных усилий при программировании.

Структуры данных

Языком Паскаль предусмотрена статическая, т.е. заранее продуманная, прогнозируемая организация структур данных типа векторов, множеств и записей из определенного числа однотипных элементов (для записей – определенного типа). Динамические, т.е. зависящие от текущей ситуации, структуры данных или «накрываются» статическими структурами, или моделируются в программе с помощью встроенных процедур, выполняющих выделение и освобождение памяти (new, dispose).

Язык Бейсик чуть-чуть проще благодаря интерпретационной, диалоговой схеме обработке программ и ориентации на один тип структур данных – массивы из однотипных элементов.

Язык Рапира допускает конструирование и представление любых иерархически организованных структур данных – кортежей – без ограничений на типы элементов, число компонент и глубину их вложенности. Необходимое освобождение и повторное распределение памяти выполняется автоматически. Например, можно построить кортеж вида:

```
< "цифры",  < 1, "один" > ,
        <  2, "два" > ,
        - - - - - - -
        <  0, "ноль" > , 10 >
```

или

```
 < 1,  2,  3,  4, "дно"  >
```

Аналоги в Лиспе

```
( "цифры"  ( 1 "один" )
        ( 2 "два" )
        - - - - - - -
        (  0 "ноль" )  10 )
```

или

```
( 1 2 3 4 "дно"  )
```

Исходные данные

В Бейсике чтобы задать исходные данные в программе, необходимо перечислить входящие в них элементарные данные в специальном операторе data и организовать цикл, выполняющий запись этих данных в нужные компоненты структур данных. Например:

```
10 data 11,22,33,44,55,66,77,88,99
20 dim A(10)
30 for i=1 to 10
40 read A(i)   '--- запись очередного элемента данных в массив
50 next i
```

В стандартном Паскале для этого придется трудолюбиво выписать 10 присваиваний.

```
program PR2 ( input, output ) ;
 var A : array integer [1..10] ;
  begin
   A[1] := 11;
   A[2] := 22;
   A[3] := 33;
   A[4] := 44;
   A[5] := 55;
   A[6] := 66;
   A[7] := 77;
   A[8] := 88;
   A[9] := 99;
   A[10]:= 0 ;
 - - - - - - - -
  end.
```

TURBO Pascal предоставляет более щадящее решение:

```
const A : array [ 1..10 ] of integer = ( 11,22,33,44,55,66,77,88,99,0 ) ;
```

но выполнить это можно только при описании переменных.

В Рапире достаточно по мере необходимости выполнять присваивание кортежа переменной:

```
A :=  < 11,22,33,44,55,66,77,88,99,0 >
```

Лисп допускает:

```
( (lambda (A) …. ) '(11 22 33 44 55 66 77 88 99 0 ))
```

Вывод структур данных

Чтобы посмотреть результаты, накопленные в векторе или массиве, на Бейсике требуется написать цикл, организующий покомпонентный вывод полученных данных.

```
  10 dim A(10)
- - - - - - - - -
  200 for i=1 to 10
  210 print A(i)
  220 next i
```

Для массива потребуется двойной цикл и т.д.

Аналогичная работа типична и для Паскале при обычном выводе на экран:

```
program PR3 ;
 var A : array [1..10] of integer ;
  begin
 - - - - -
  for i := 1 to 10
      write ( A[i] );

 - - - - -
  end.
```

На Рапире независимо от структуры данного для этого достаточно:

```
вывод: A
```

или

```
? A
```

В случае ввода данных и обмена с внешней памятью картина такая же.

Аналогично Lisp:

```
(print A)
```

Слияние структур данных

При необходимости совместной обработки раздельно полученных данных на Бейсике (и на Паскале) необходимо заранее предусмотреть массив (вектор) достаточной для этого размерности и написать программу переписи этих данных в общий массив.

```
 10 dim A(10), B(20), C(30)
- - - - - -
 200 for i=1 to 10
 210 C(i) = A(i)
 220 next i

 230  for i=1 to 20
 240  C(i+10) = B(i)
 250  next i
```

На Рапире это выполняется операцией склеивания кортежей, результат которой присваивается переменной.

```
C := A + B
```

Для текстовых строк и в Бейсике, и в Паскале обеспечен такой же способ склеивания данных.

Pure Lisp такое преобразование выполняет копированием первого списка, что защищает данные от случайных искажений.

Передача параметров

Параметризация формул является мощным средством унификации записи программ, а главное, методом принципиальной экономии трудозатрат на отладку с обеспечением надежности функционирования программ и, естественно, с разделением труда. Средства выделения подпрограмм в Бейсике не поддерживают никакого механизма параметризации (кроме функций). Поэтому передачу параметров приходится организовывать с помощью присваиваний переменным и циклов в случае пересылки массивов.

В Паскале процедуры и функции обеспечены передачей параметров при совпадении типов аргумента и параметра.

В Рапире выполнится передача параметром любого элемента данных, включая процедуры и функции. То обстоятельство, что возможное несоответствие типа значения типу операции обнаружится лишь при исполнении, мало сказывается на удобстве учебного диалога, в котором разница между моментом перед передачей параметров и моментом его применения малосущественна (в отличие от производственного применения программ, когда необходимо не допустить до исполнения сколько-нибудь сомнительный процесс). При начальном знакомстве это освобождает от принудительного планирования структуры данных и ее представления.

Лисп отличается от этих языков возможностью в качестве параметров передавать представления функций, т.к. функции представляются списками, как и данные.

Модификация программы

На начальных этапах отладки программ на Бейсике можно просто наслаждаться удобством вставки новых команд. Стоит лишь набрать строку с новым номером, как она вставится в нужное место без особых усилий по подводу курсора в нужное место. Если изменить номер строчки, то она сразу раскопируется, и т.п. Но вскоре резерв свободных номеров строк исчерпывается, и возникает необходимость повторной нумерации. При программировании сколько-нибудь большой программы происходит распределение и учет диапазонов занятых номеров строк текстов подпрограмм и имен переменных и функций. При модификации программы следует внимательно отслеживать взаимосвязи подпрограмм по переходам. После перенумерации программы это приводит к трудноуловимым дефектам, потому что номера подсознательно связываются с отдельными осмысленными частями программы. При абсолютной уверенности в правильности программист обращается к таким частям по устаревшим номерам. Искать та-

кие ошибки очень трудно. Это провоцирует воздержание от перенумерации построением длинных строк, содержащих много команд, что быстро портит обозримость текста и усложняет работу по его коррекции и модификации вплоть до несостоятельности автора в поддержании работоспособности программы (можно рекомендовать специальную методику работы с Бейсиком на основе систематической перенумерации текста и наглядных комментариев, обеспечивающих профилактику подсознательного связывания смысла с номерами строк. Но кто может утверждать, что умеет прививать школьникам склонность к добровольной дисциплине, связанной с непонятно когда окупающимися накладными расходами? Одни из них не верят в ограниченность своих способностей, другие тяготятся набирать лишние символы, «ненужные» машине. Неприятность сейчас, а мифический выигрыш потом).

В Паскале при модификации программы достаточно отслеживать взаимодействие с именами в объемлющих блоках. Они всегда расположены выше по тексту программы вне блоков равного ранга. Поэтому естественная схема развития Паскаль-программ – «сверху-вниз», что подходит для хорошо отработанных, стабильных постановок задач.

В Рапире при модификации процедуры необходимо понимание обстановки, в которой предстоит исполнять эту процедуру, что требует учета смысла имен процедур и внешних имен значений, используемых динамическими предшественниками. В не слишком больших программах такая работа сводится к просмотру страниц, содержащих упоминание модифицируемой процедуры. Достаточно естественно выполняется как нисходящее, так и восходящее развитие программы, что пригодно и для экспериментального решения слабо изученных задач.

Lisp допускает повторное определение функций.

Трудоемкость программирования

Трудоемкость программирования качественно различна, если сложность работы характеризуется определенными границами в выборе средств и решений. Сравнительно очевидна разница в следующих уровнях сложности программирования:

1. Выбор обозначений в программе не требует никаких искусственных приемов, и при работе программы не возникает никаких проблем ни с объемом памяти, ни со скоростью исполнения. При таких условиях программирование не вызывает особых затруднений, но выход из таких границ зависит от особенностей реализации языковых конструкций.

2. Затруднения в выборе обозначений могут быть разрешены систематизацией и учетом, недостаток памяти нейтрализуется своевременным отказом от ненужных данных, а скорость становится приемлемой в результате рациональной организации циклов. Работа на этом уровне сложности требует некоторых профессиональных приемов, до которых надо догадаться или у кого-то научиться дополнительно. Во всяком случае реорганизация циклов, систематизация, учет и отказ от значений – неизбежные накладные расходы.

3. Трудности с обеспечением приемлемого решения задачи столь существенны, что приходится тщательно организовывать переобозначение данных и частей программы и временное хранение их во внешней памяти, реализовывать специальные способы представления и обработки информации, позволяющие исполнить программу на грани возможностей имеющихся технических средств. Независимо от языковых средств работа на этом уровне сложности принципиально очень трудоемка даже при высокой профессиональной квалификации. Но приближение к границам возможностей и трудоемкость достижения предельной производительности зависят от приспособленности используемых средств к выражению эффективных решений, что требует от программиста и знаний, и навыков, и опыта.

На Бейсике первый уровень сложности исчерпывается очень быстро на сравнительно малых программах и данных из-за малой мощности набора различимых идентификаторов и применения номеров строк в качестве обозначения части программы. Второй уровень сложности обладает высокой трудоемкостью из-за неудобства модификации программ, превышающих экран по объему. При необходимости работать на третьем уровне сложности средствами Бейсика до работоспособной программы дело доходит не часто, хотя и не так уж редко. Проще изучить еще один язык, более приспособленный для сложной работы.

На Паскале первый уровень сложности связан с умением использовать латинские (или английские и т.п.) обозначения, но благодаря трансляционной схеме обработки программ довольно свободен в отношении памяти для программы и данных. Программист, владеющий иностранным языком[3] или привыкший пользоваться транслитерацией, на первом уровне сложности может решать довольно широкий класс задач. Второй уровень сложности возникает при больших объемах данных и при организации динамических структур. Необходимость работы на третьем уровне сложности воз-

[3] Специалисты в эргономике утверждают, что русскоязычные программисты проигрывают примерно 30% производительности труда на использовании английского вместо родного.

никает не скоро, т.к. определение языка и его типовая реализация приспособлены к достаточно эффективной работе.

На Рапире первый уровень сложности более свободен по системе обозначений – русские и латинские литеры, но более ограничен по объему памяти для программы и данных. Теоретически возможна реализация, поддерживающая такую работу на более широком классе задач, но она пока не выполнена. Из-за недостатка памяти на имеющихся школьных компьютерах вскоре возникает необходимость экономии, и работа приобретает сложность второго уровня. Переход к работе на третьем уровне сложности в Рапире не поддержан, причем, возможно, в этом нет необходимости, т.к. для чисто производственного применения язык не предназначен. Но навыки, приобретенные при работе с Рапирой, на втором и третьем уровне сложности пригодятся в работе на производственных языках.

Лисп приспособлен к предельному повышению сложности решаемых задач благодаря приспособленности к отладке и программному эксперименту.

20.2. Elan

В конце 1980-х языков появился ряд учебных языков программирования, нацеленных на нейтрализацию доминирования Бейсика. Яркой пример – разработка языка Elan, созданный одним из авторов языка Algol-68 Костером. Язык конструктивно близок языку Рапира, напоминает бестиповый Паскаль [46].

20.3. Языки XXI века: A++ и Oz

С начала XXI века новые учебные языки программирования обрели мультипарадигматический характер. Например, язык A++ поддерживает изучение функционального, объектно-ориентированного, императивного и логического программирований, включая рекурсию и функции высших порядков [37].

Более амбициозный мультипарадигматический язык программирования Oz, реализованный в учебной системе программирования Mozart, создан для ознакомления студентов с наиболее важными парадигмами программирования, обеспечивающими рациональное применение современных ИТ и перспективы их усовершенствования на базе полного спектра доступных механизмов, включая параллелизм, сети, реальное время и т.д., без претензий на предварительное изучение теории программирования [49].

Операционная семантика Oz представляет собой объединение абстрактных машин императивного и функционального программирований, погруженное в схему параллельного программирования в виде многопоточного комплекса с монитором. Поддержка логического и объектно-ориентированного программирований выполнена как синтаксическое расширение, что достаточно для ознакомления с техникой представления знаний и развития иерархии классов в некоторых приложениях.

Лексика и синтаксис Oz наследуют многие черты языка Pascal.

Элементарные значения – целые без ограничения длины, литеры, атомы и имена, включая идентификаторы анонимных конструкций. Выделен тип Bool, к которому сводятся значения предикатов.

Структуры данных строятся из записей, допускающих расширение числа полей и сокрытие их части. При необходимости манипулировать изменением данных используются ячейки – контейнеры для хранения заменяемых значений. Кроме обычных структур данных поддержаны очереди (stream) и порты. Арность записей – это список имён полей в лексикографическом порядке.

Списки могут быть открытыми или замкнутыми. Строки рассматриваются как списки литер. Представление списков допускает использование переменных. Список – это Nil или запись из двух полей.

Язык поддерживает динамическую типизацию без предварительного описания объявляемых переменных. Неявное приведение типов значений отсутствует. Отсутствуют средства доступа к внутреннему представлению данных.

Как правило, структуры данных монотонны, т.е. информацию можно дополнять, а не изменять или уничтожать.

Переменные пишутся с заглавной буквы и подчинены дисциплине однократного присваивания. Значения переменных могут быть неизвестны или объявлены вычисляемыми в будущем, что приводит к откладыванию зависящих от них вычислений подобно ленивым вычислениям.

Конструирование программы основано на введении процедур и функций, включении модулей (пакетов) и функторов, специфицирующих поведение модулей, возможно доступных дистанционно с указанного сайта. Функции реализованы как процедуры без побочных эффектов, сохраняющие результат в специальной переменной. Анонимные процедуры получают внутренние идентификаторы. Процедуры высших порядков используются как средство представления схем функционирования программ и их интерфейсов. Существуют глобальные и локальные области видимости и иерархия процедур.

Предусмотрено использование шаблонов в управляющих конструкциях типа CASE. Нелогическое значение предиката в IF рассматривается как ошибка. Имеются средства реагирования на события, обработки ошибок и исключений, учёта реального времени.

Параллельные вычисления поддержаны в стиле разделения времени между потоками программы, способными взаимодействовать в терминах сообщений или общей памяти. Дисциплина разделения времени учитывает приоритеты, задающих пропорции выделения времени с профилактикой полного оттеснения какого-либо потока. Для решения задач переборного типа имеется механизм программирования стратегии поиска в рамках ограниченной области.

Потоки, из которых строится многопоточная программа, обладают своей памятью и могут иметь доступ к внешним данным, включая доступ к сторонним сайтам. Потоки устроены как традиционная императивно-процедурная программа. Новые потоки запускаются в стиле развилка как в UNIX [43]. Авторы утверждают, что это происходит в 60 раз быстрее, чем в Java JDK 1.2. Систем программирования Mozart доступна через Emacs.

20.4. Практикум по программированию

Центральным моментом обучения программированию бесспорно является компьютерный практикум. Именно компьютерный эксперимент объективно подтверждает правильность понимания изучаемых явлений. Успех такого обучающего эксперимента зависит от дозирования изучаемых конструкций и очерёдности их освоения. Для большинства ЯП это средства обработки данных, их идентификации и объединения в комплексы и механизмы управления обработкой данных.

Первый шаг обычно представляет собой линейный участок с константными выражениями над элементарными значениями: целыми числами и строками. Для интерпретируемых ЯП обычно СП предоставляет режим автоматического вывода результатов на экран. Программы на компилируемых ЯП должны явно включать такой вывод.

Типовая задача второго шага – навыки идентификации значений с акцентом на её роль при многократном использовании промежуточных результатов и вспомогательных определений. Третий шаг – декомпозиция программ на функции и процедуры с отработкой техники конструирования программ с использованием своих и библиотечных процедур. Часто это библиотеки компьютерной графики. Рекурсия и сопутствующая ей про-

блема ограничения вычислений дает убедительное методическое обоснование роли предикатов и сравнений в выборе нужной ветви вычислений.

Далее появляются структуры данных и соответствующие им механизмы управления обработкой структур данных. Сначала ветвления: If, Case, затем циклы: While; Repeat … Until; For. Обработка сложных структур данных методично приводит к изучению средств ввода данных и использования файлов для их подготовки и хранения.

Значительные трудности в постановке практикума по программированию влечёт использование производственных СП, неприспособленных для образовательных целей, в качестве полигона для учебного эксперимента [11, 15].

21. ОЛИМПИАДНОЕ ПРОГРАММИРОВАНИЕ

Конкурсы дают возможность всеобщее нормированное обучение сопроводить для наиболее заинтересованных интенсивным обучением.

21.1. Logo-олимпиады

С начала 1990-ых годов ЮНЕСКО рекомендовал идею Logo-миров как основную линию внедрения информатики в образовательный процесс слабо развитых стран. Эта идея выглядела особо привлекательной благодаря свободе от разных национальных традиций и одновременно возможности настройки на местный колорит. Появились конкурсы для младших школьников на базе систем программирования для языка Logo. Оказалось, что изначально проявляется существенный разброс в способности школьников найти и запрограммировать алгоритм решения задачи на прорисовку.

21.2. Школьные конкурсы

Система школьных конкурсов позволяет создать условия для разработки методов факультативного обучения по самобытным авторским методикам (чужие методики не соответствуют). Олимпиадное движение по информатике фактически выполняет работу творческих мастерских, объединяющих программистов и методистов.

Школьные научно-практические конференции, рассматривающие разнообразные проекты, отдельно (новички, выпускники, программисты, прикладники, команды, игры) поддерживают мягкий, поощряющий регламент, в котором достижима компетентная оценка с разных сторон (подача мате-

риала, качество результата, техника исполнения) при участии преподавателей в работе жюри. Формируется регулярный обзор средств, смотр-парад школьных разработок, конференции школьных достижений на всех видах техники, не только на компьютерах.

21.3. Студенческие чемпионаты

Значимость таких вопросов заметна в практике олимпиадного программирования (часто рассматриваемого как интенсивное обучение производственному программированию), в котором различаются роли тренеров, непосредственных участников, технической поддержки и членов жюри. Если не учитывать стадию тренировок, то собственно олимпиадный процесс разделяется на четыре этапа:

Отбор комплекта задач:
- идея программисткой задачи;
- выбор красивой легенды постановки задачи;
- методы проверки возможных решений;
- критические тесты/нетесты на «узкие места» в решении;
- оценка сложности решения.

Техническая подготовка олимпиады:
- создание эталонного решения задачи;
- определение форматов ввода-вывода данных для программы решения;
- задание ограничений на решение задачи;
- комплектация информационной обстановки для проведения олимпиады;
- техника подсчета рейтинга решений (баллы);
- настройка тестирующего автомата для прогона решений на тестах жюри.

Проведение олимпиады:
- вопросы к Жюри;
- проект хода решения комплекта задач;
- изобретение алгоритмов решения для незнакомых задач;
- кодирование программ для известных решений знакомых задач;
- подбор тестов для отладки программы;
- отладка программы;

– сдача отлаженной программы на проверку автоматом жюри;
– просмотр реакции автомата на сданную программу;
– повтор решения отвергнутых программ.

Подведение итогов:
– уточнение инструкции, условий задач и тестов;
– мониторинг хода олимпиады;
– поддержка работы автомата в нестандартных ситуациях и перегрузке;
– апелляция;
– утверждение итогов олимпиады;
– награждение;
– публичный разбор решений задач;
– анализ качества работы автомата.

При весьма высокой значимости способностей в программировании опыт показал существенный вклад тренировок в олимпиадные победы. Многие тренеры олимпиадных команд начинают обучение программированию с освоения ассемблера и работы на уровне командной строки [30, 43].

22.3. Досуговое программирование

В круг интересов академика А.П. Ершова естественным образом входили проблемы обучения программированию. Образовательный потенциал программирования А.П. Ершов отмечал еще в отчете о свей первой поездке в Англию. Знаменитый, до сих пор встречающийся в ссылках, Альфа-проект сопровождался впечатляющими краткосрочными курсами по программированию, заметно пополнившими ряды хороших программистов не только в Новосибирске. Уже в начале 60-х годов сотрудники отдела Ершова учили программированию как трудовой профессии школьников, многие из которых сохранили верность программированию до сих пор.

ШЮП, ЛШ, ЗШ, ШИ – этими аббревиатурами Андрей Петрович обозначал части механизма, продвигавшего Информатику и Компьютеры в наши школы:
– с 1976 года – ЛШ или ЛШЮП (Летняя Школа Юных Программистов)
– с 1977 года – ШЮП (Школа Юных Программистов)
– с 1978 года – ЗШ (Заочная ШЮП)
– с 1982 года – ШИ (Школьная Информатика)

Летом 1976 года Ю.А. Первин и Н.А. Садовская пригласили в Новосибирск из Харькова кружок юных кибернетиков Г.А. Звенигородского. Ре-

зультативность харьковской методики вовлечения школьников в компьютерный мир настолько впечатлила А.П. Ершова, что в 1977 году Г.А. Звенигородский был приглашен в Новосибирск и сделал первый набор (более 70 школьников, возраст от 10 до 15 лет) в Новосибирскую Школу юных программистов (ШЮП). Эту методику сейчас можно бы называть как объектно-ориентированное программирование учебных обстановок на понятном младшим школьникам языке Робик. В 1978 году на страницах журнала «Квант» Н.А. Юнерман энергично развернула Заочную школу юных программистов (ЗШ), лучшие ученики которой приглашаются в Новосибирск на летние школы (ЛШ) поработать на компьютере. Преподают в этих школах преимущественно настоящие программисты-практики.

ЛШ – самое яркое и эффективное звено этого, запущенного А.П. Ершовым механизма распространения и популяризации Информатики. ЛШ предельно привлекательны для всех участников – школьников, педагогов, программистов, конструкторов и ученых.

На ЛШ съезжались школьники, чтобы впервые увидеть компьютер и «вкусно» поработать, педагоги, чтобы нащупать методику учебного применения компьютера, программисты, чтобы вместе со школьниками опробовать свои замыслы, конструкторы, чтобы испытать новую технику, ученые, чтобы популярно изложить свои идеи благодарной аудитории.

Ехали, чтобы за 10–20 дней узнать 5–10 языков и систем программирования, чтобы испытать себя в программистских олимпиадах, поработать в проектах рядом с мастерами программирования и подискутировать с мировыми светилами и классиками программирования. Решения задач на олимпиаде представлялись любыми способами. Никому и в голову не приходило ограничивать используемые языки программирования. Делом чести компетентного Жюри было обеспечить проверку решения на любом известном школьнику языке.

Ехали, чтобы вернуться знатоками программирования на ассемблере, Паскале, Форте, Алголе-68, Лиспе, Сетле, Модуле, Фортране, Поплане и т.д. и специалистами по наладке функционирования самого капризного в мире школьного оборудования. БЭСМ-6, HP, ЕС ЭВМ, СМ, Apple, АГАТ, Yamaha, БК0010, IBM PC, выставочные экземпляры зарубежной техники, предоставляемой ИПИ АН (А.В. Гиглавый). Машин остро не хватало – распределяли время круглосуточно.

Были случаи, что после ЛШ восьмиклассник в родном городе, где нет программистов, самостоятельно справлялся с установкой и запуском в эксплуатацию новых, никому не известных школьных компьютеров, превращая свою школу в очередной центр ШИ. Школьный учитель, прошедший

ЛШ на правах ученика, получал импульс для самостоятельной организации аналогичной школы у себя дома.

Ехали, чтобы завести друзей, петь ночью у костра, ставить спектакли про Аду, Алгол и Фортран, «начинать» КВН, читать стихи, выпускать стенгазету и сочинять свои песни к конкурсу городов, слушать программистские байки от умных людей. А.А. Берс., Н.Н. Бровин, Д.Ш. Матрос, А.Н. Терехов, О.Ф. Титов, Л.Б. Штернберг и многие другие были доступны круглосуточно и в отличии от компьютера бесперебойно. Ежевечерние «Программистские сказки» А.А. Берса у костра впечатляли школьникам не меньше, чем борения с компьютером. Иные лекции, например, по технологии программирования в исполнении А. Н. Терехова, завершались криком души юных дарований «ХОЧУ!... ХОЧУ, ЧТОБЫ У МЕНЯ БЫЛИ ТАКИЕ УЧИТЕЛЯ!!».

Сложнейшие приемы программотехники от кодирования до оптимизации программ, включая методы тестирования, разработки компиляторов и создания своих языков программирования и конструирования приложений постигались школьниками ради продвижения любимых и интересных тем. Сообщение Л.Б. Штернберга, что он в любой программе найдет хотя бы одну неоптимальность, юные «асы программирования» воспринимали как интеллектуальный вызов и рьяно доводили свои программы до совершенства.

Мир ЛШ отнюдь не замыкается на программировании. Вечерами ученые Академгородка, Москвы, Ленинграда и др. городов многое рассказывали о физике, биологии, генетике, лингвистике, астрономии, истории и другое. Рассказывали не только школьникам, но и педагогам, и программистам.

Ученики могли показать себя не только успехами в учебе и программировании, но и докладом на конференции, и показом «домашних» проектов, и изданием газеты, и подготовкой концерта. Можно было поздравлять именинников на утренней линейке (Н. Н. Бровин ее проводил так, что это было весело), поболеть или победить на шахматном чемпионате, теннисном турнире, поиграть в баскетбол. На берегу Обского моря – катамараны, серфинг, водный парашют, прогулка на теплоходе.

Труднее всего жилось наградной бригаде (рук. Н.А. Юнерман), экспертной комиссии (рук. О.Ф. Титов) и Жюри (рук. Н.Н. Бровин). Экспертная комиссия добивается от юных программистов понимания, что и зачем они делают на машине. Жюри выявляет, в чем именно школьник добился прогресса, чтобы при награждении каждый чувствовал, что его наградили заслужено. Было не принято бросаться словом «Лучший». Все, кто приехал на школу, имели право и возможность в чем-либо считать себя лучшими. Типичные формулировки награждения: «За достигнутые успехи ...», «За

интересную разработку ...», «За красивое решение...», «За самую сложную ошибку, найденную в ...», «За удачную форму доклада ...», «За любознательность на конференции и лекциях ...» или «За доставленное Жюри удовольствие ...» (автор формулы Игорь Мадьяров из Иркутска). Поощрять старались индивидуально, отмечая все, что поможет юному программисту найти самостоятельный путь. Наградная бригада заботилась, чтобы призы соответствовали вкусам и способствовали профессиональному становлению растущих программистов.

Каждой ЛШ ее научный руководитель, при всей его необычайной занятости в связи с введением ОИВТ в школы, А.П. Ершов посвящал хотя бы один полный день, в 1987 году в работе ЛШ принял участие Дж. МакКарти, бывали и другие знаменитости. Все это создавало атмосферу энтузиазма, творческого накала, жажды знаний и доброжелательной инициативы. ЛШ начинали ждать сразу с сентября, как только приходили в себя.

Наиболее естественное формирование ЛШ получалось, когда приглашение учащихся определялось по результатам их учебы в ЗШ. Практиковался и конкурсный отбор, а также привлечение по рекомендации. Следует отметить, что рекомендации программистов-практиков намного превышают конкурсный отбор по уровню подтверждения в деле. По-видимому, критерии профессиональной пригодности к программированию еще не вполне осознаны, но на практике достаточно очевидны.

Если ЗШ занималась поиском наиболее способных и заинтересованных учеников для ЛШ (почти без возрастных ограничений, включая отдельных школьных педагогов), то Новосибирская ШЮП в течение учебного года формировала команду консультантов-подмастерьев, берущих на себя значительную нагрузку по организации и проведению ЛШ. Эта команда начинала работу с марта в форме заседаний оргкомитета ЛШ. Обсуждались, проговаривались заранее, согласовывались разные детали механизма ЛШ – нечто вроде безмашинной отладки программ. Школьникам и студентам, включившимся в эту команду, многое доверяли, но и многое спрашивали. Ни возраст, ни неопытность, ни неумение не могли быть оправданием в случае провала поручения. Такая, заранее сплоченная, команда всегда была в курсе происходящего на ЛШ и обеспечивала ее здоровый микроклимат – дух ЛШ.

География участников ЛШ быстро расширилась: Харьков, Красноярск, Ленинград, Горький, Москва, Иркутск, Абакан, Братск, Томск, Петропавловск, Ульяновск, Махачкала, Владивосток, Фрунзе, Ташкент, Алма-Ата и другие города СССР, Болгария, Чехословакия, Венгрия, Германия, Польша,

Голландия. По сути ЛШ превращалась в международный клуб любителей детского программирования.

Методика Г.А. Звенигородского раннего обучения программированию учащихся ШЮП кроме языка Робик, (идеи исполнителей которого отчасти воплощены в Роботландии Ю.А. Первина), активно использовала прорисовку синтаксических диаграмм, четкое выделение методически обусловленных учебных концентров и профилактику прирастания к первому средству. Разнообразие компьютеров, языков и систем программирования, операционных систем, стилей и технологий представало перед учащимися в первый же год обучения. Для более старших школьников методика Г.А. Звенигородского в Новосибирске получила развитие в виде учебно-производственного языка Рапира – он похож на бестиповый Паскаль с более универсальными структурами данных и управления – и проекта интегрированной учебной среды «Школьница». Большую роль на этом уровне сыграла компьютерная графика – система Шпага для программирования прорисовки любимых картинок в понятных детям терминах.

Важнейшей особенностью подхода Г.А. Звенигородского к обучению школьников следует признать целенаправленное формирование разноуровневых по квалификации и возрасту групп/мастерских, выполняющих коллективный проект по внешнему заказу. В группу включаются добровольно – кому интересно. При группе имеются консультанты/подмастерья – знатоки используемых средств и методов из числа более опытных учеников. Группой руководит мастер – специалист, интересующийся задачами, близкими к внешнему заказу. Внешне это очень похоже на часто упоминаемый метод проектов: система вводных и базовых проектов ВКИ НГУ, проектные разработки по заказу учителей в своей школе, проекты по моделированию физических процессов (движение ракеты, «стрелялки» и т.п.). Принципиальная разница заключается в наличии внешнего заказа, критерии выполнения которого не подвластны руководителю и консультантам. Интересно, что прошедшие такую выучку программисты оказались в дальнейшем способны к бесконфликтному оптимальному распределению обязанностей в коллективных проектах.

Под руководством Г А. Звенигородского группой студентов и старшеклассников на первом советском ПК «АГАТ» была создана система «Школьница», в которую вошли Робик и Рапира, использовавшиеся при разработке первых демонстрационных образцов программных средств обучения по школьной тематике. Эти образцы были продемонстрированы Комиссии очень высокого ранга, что и позволило ШИ стать новой школьной дисциплины «Основы информатики и вычислительной техники».

В 1985 году участники и выпускники ЛШ Л.С. Бараз, Н.Г. Глаголева, П.А. Земцов, Е.В. Налимов, Е.В. Боровиков, А.В. Грабарь, С.А. Терехов, Н.Ш. Погосян, М.А. Зайцев, и др. под руководством Л.В. Городней приступили к профессиональной разработке школьного программного обеспечения на базе КУВТ Ямаха и Электроника УКНЦ. Кроме удобной системы программирования на языке Рапира с дружелюбным интерфейсом, окнами и достаточно быстрой машинной графикой, был разработан тренажер Микрорапира для начального знакомства с программированием, непревзойденный по устойчивости и удобству ряд текстовых редакторов Tor, система подготовки текстовых документов Dcm, автоматизированные средства восстановления файловой системы Vfy, текстовый макрогенератор Gpm, низкоуровневые средства ручной работы с файлами Fix, и Nalim Commander для ПК Yamaha-MSX-2 – прекрасный аналог Norton Commander. В реализации на Электронике УКНЦ язык Рапира был обогащен возможностью работать с вычисляемыми именами и ассоциативными таблицами. Кроме того, были разработаны эффективные сервисные средства системного администрирования и поддержки профессионального программирования на СМ-4. Многие из этих средств эксплуатируются в школах до сих пор.

Таким образом Новосибирская система ЛШ+ЗШ+ШЮП+ШИ показала способность готовить программистские кадры мирового уровня. Механизм ее организации и функционирования вполне соответствовал критерию Дж. Вейнберга, автора основополагающего труда «Психология программирования». На вопрос «Откуда берутся хорошие программы и хорошие программисты» Дж. Вейнберг дает ответ: «"Хорошее программирование, как хорошая дружба – крепнет от поощрения индивидуальности и признания достоинств». Программные средства для новых компьютеров и очередные версии информационных систем многих известных фирм теперь выпускаются при участии выпускников ЛШ, ныне работающих в фирмах Intel, Microsoft, HP, IBM, Oracle и т.д.

Летние школы юных программистов проводились в Новосибирске с 1976 года. На них собирались интересующиеся программированием школьники, которые в очень интенсивном режиме за две-три недели приобретали или развивали навыки программирования, выполняя часть проекта под руководством опытного программиста и при содействии сверстников-консультантов, ранее получивших навыки работы. Заочные школы юных программистов функционировали на страницах журнала «Квант», публиковавшего научно-популярные статьи по программированию и задачи, по результатам решения которых учащиеся могли попасть на Летнюю

школу. Районная школа юных программистов при ВЦ СО АН СССР работала с 1977 года. Она выполнила ряд экспериментов по раннему обучению программированию, по разработке программных средств, в том числе и обучающих, силами школьников и студентов, а также готовила консультантов для Летних школ. Многие выпускники этой системы ШЮП стали высококвалифицированными специалистами в области системного программирования.

Наиболее значительным звеном этой системы была Летняя школа, которая имела статус Всесоюзной с международным участием. На этой школе, обладавшей предельной привлекательностью для увлеченных программированием школьников, учились и их учителя – преподаванию информатики, учились и студенты – руководству разработкой программных средств, учились и программисты – разработке программного обеспечения учебного назначения. Школа выполняла роль выставки новых образцов школьных компьютеров и обучающих программных средств. В работе школы принимали активное участие специалисты из разных городов и стран, причем не только информатики, но и физики, математики, химики, биологи, лингвисты. Опыт Новосибирских летних школ был воспринят и творчески переработан при организации аналогичных школ во многих городах. Наиболее гармоничное сочетание учебы, труда и отдыха достигалось при проведении Школы на базе туристского центра «Сибиряк», но в настоящее время из-за соображений безопасного размещения техники Школа проводится на базе ВКИ НГУ, который переключил на себя также работу по заочной и районной школе с целью отбора и подготовки абитуриентов.

Более чем 60 лет существует программирование как профессия. Но редкое учебное заведение может похвастаться успехами в области полноценной подготовки программистов. Многие полагают, что программистом надо родиться. Музыкантом тоже надо родиться, но это не отменяет необходимость в обучении музыкальному искусству и музыкальной культуре. Многие талантливые программисты могли бы ярче проявить себя при систематическом обучении своей профессии. Наиболее убедительным в этом плане был опыт Г.А. Звенигородского (ШЮП).

Не слишком многочисленные предложения по систематизации обучения программированию на базе системы народного образования оказываются на практике достаточно трудны и не очень результативны. Основные проблемы теперь осложнены резким сужением круга квалифицированных руководителей учебных проектов. Предлагается вариант организации профессиональной подготовки программистов на основе доступных в настоя-

щее время новых технических и телекоммуникационных средств с учетом появления домашних компьютеров у молодежи.

Типичные «проколы» при организации обучения программированию сводятся к слишком прямолинейным попыткам следующего типа:

- преподавать «чистое» программирование;
- формировать одновозрастные группы;
- отбирать исключительно способных к учебе;
- выстраивать «сквозной» учебный план на много лет;
- снижать трудоемкость и сложность учебных работ;
- форсировать приобретение практических навыков.

Все это нужно, но следует учесть, что

- практика программирования должна быть погружена в общедоступный контекст, чтобы ученики познакомились со всеми фазами жизненного цикла программ;
- в разновозрастных группах легче происходит смена ролей, многие из которых желательно испытать на этапе учебы;
- рядом должны быть будущие пользователи, чтобы будущий программист учился их понимать;
- темп «созревания» психологических структур может требовать приостановок в учебе;
- многие явления производственного программирования не понятны без повышенной сложности, трудоемкости и работы в ограниченных условиях;
- навыки, приобретенные на ранних этапах обучения, потребуется уметь заменять на новые.

Кроме того, профессия программиста бывает сопряжена с определенными требованиями к этико-психологическим особенностям, складу нервной системы и характера, которые могут не проявляться в жесткой учебной обстановке, но обычно проявляются в неформальном и досуговом общении. Неудивительно, что чисто конкурсный отбор для специализации по программированию существенно менее эффективен, чем рекомендация профессионала.

Таким образом, схема учебного процесса включает в себя досуг, учебу и практику. Часть практики посвящена обеспечению досуга и учебы. Структура практических задач выстроена как развиваемая силами учащихся системная поддержка учебной обстановки. Каждый этап учебы сопряжен с передачей опыта на более ранние этапы в форме консультаций, по-

становки задач, руководства учебными проектами или усовершенствования системной поддержки.

При таком подходе выпускник полного цикла обучения имеет шанс обладать навыками:

- консультирования пользователей;
- сборки программ из готовых компонентов;
- индивидуального программирования небольших программ;
- конструирования несложных игровых миров и языков;
- исполнения части работ в программистской команде;
- выполнения основной работы с программистской командой;
- консультирования начинающих программистов;
- совмещения ряда фоновых работ;
- руководства коллективным проектом;
- скоростного выполнения срочных работ;
- выполнения квалификационных работ;
- руководства квалификационными работами;
- оценки своих и чужих результатов;
- ведения исследовательских проектов;
- постановки своих тем;
- демонстрации результатов;
- приемки результатов;
- проверки корректности поведения информационных систем;
- оценки трудоемкости программирования;
- определения и выполнения графика работ.

При желании можно выбирать комплект необходимых навыков, путь их приобретения и тематику учебного материала.

Общий подход к практическим заданиям – улучшаем то, с чем имели дело сами, или создаем нечто впервые.

Переход на очередной уровень обучения осуществляется как демонстрация достигнутых умений в виде полученных результатов. При необходимости привлекаются на помощь тренажеры и консультанты.

ЗАКЛЮЧЕНИЕ

Выбор парадигмы программирования – это выбор концептуальной схемы постановки проблем и методов их решения, инструмент «грамотного» описания фактов, событий, явлений и процессов, выделения частных и общих понятий. Альтернативные подходы к обработке информации, накопленные и

сложившиеся в форме языков и систем программирования, принято называть парадигмами программирования. Изучение и чёткая классификация уже сложившихся и новых компьютерных парадигм призваны помочь обоснованному выбору компьютерных языков при формировании новых программных проектов и создании новых информационных технологий.

Естественная классификация по ёмкости абстрагирования конструкций, выделяющая ЯП низкого, высокого и сверх высокого уровня, дополняется определением парадигм, поддерживаемых ЯСП. В настоящее время число по существу различимо более двух десятков парадигм. Многие языки программирования относят к пяти-восьми парадигмам. Часть упоминаемых в разных источниках ПП можно характеризовать как стили или методики, отражающие поиск путей снижения трудоёмкости программирования и повышения надёжности программ на базе доступных СП. Например, аспектно-ориентированное программирование поддерживается как макрорасширение ООП. Структурное программирование фактически сводится к ряду рекомендаций по стилю представления императивно-процедурных программ. Мета-программирование представляет собой технику компиляции программ в комплекте с типовыми элементами данных. Недетерминированное программирование не более чем частный случай параллелизма.

При определении ПП обычно поляризуются следующие характеристики ЯП:

- программируемые решения представляются в императивной или в декларативной форме;
- обрабатываемые элементы данных позиционируются как адресуемые блоки памяти или независимо размещаемые значения;
- программа может быть защищена от изменений в процессе её выполнения или допускать программируемые модификации по ходу получения результатов вычисления;
- программа может быть целостной или собираться из типовых компонент и шаблонов;
- представленные в программе функции могут быть частичными, типизированными, обрабатывающими значения заданного домена или универсальными, дающими разумную реакцию на любой элемент данных;
- управления вычислениями выполняется последовательно или параллельно;
- порядок действий может быть определённым или недетерминированным;
- вычисления могут быть энергичными или ленивыми;

– области видимости имён могут быть глобальными или локализованными по иерархии конструкций с возможностью восстановления контекста;
– распределение и повторное использование памяти может быть действием в программе или выполняться автоматически СП;
– инициирование памяти первоначально размещаемыми значениями может требовать программируемых действий или выполняться в СП по умолчанию;
– домены значений могут быть независимыми или допускать пересечения;
– результат выполнения программы может быть рассредоточен по ряду переменных или сконцентрирован в специальном регистре;
– контроль правильности может выполняться статически – при анализе текста программы или динамически – при выполнении кода программы.

Выбор между так противопоставленными характеристиками в представлении программы выражается синтаксически или с помощью спецификаторов. В практике программирования признаны основными парадигмы императивно-процедурного, функционального, логического и объектно-ориентированного программирований, поддерживающими механизмы снижения трудоёмкости полного жизненного цикла программ, с тенденцией продвижений к параллельному программированию. При оценке образовательного значения ПП выделяют в качестве базовых функциональное, параллельное и императивно-процедурное программирования, а логическое и ООП рассматривают как дополнение, изучаемое в виде расширения базовых парадигм. Мультипарадигматичность долго живущих ЯП и тенденция создания новых мульти-парадигматических ЯП говорит о созревании единой ПП, объединяющей выверенные в практике механизмы результативного программирования.

Разработка учебного языка программирования, приспособленного к ознакомлению студентов со сложными моделями вычислений, с параллельным программированием и с методами верификации программ, без которых надежность и производительность программирования весьма проблематична, должна быть нацелена на выбор механизмов для поддержки экспериментов по разработке новых программ, языков и парадигм, ориентированных на учебно-исследовательские проекты в области создания распределенных информационных систем по той простой причине, что решения хорошо изученных задач непрерывно перетекают в библиотеки типо-

вых компонент или в комплекты инструментальных средств, готовых для непосредственного применения. Наличие таких движений как GNU, Imagine Cup, школы Технопарка, сайты ведущих программистских компаний, привлекающие энтузиастов к участию в новых проектах, можно рассматривать как благоприятную среду для формирования профессионального корпуса программистов.

Данный материал завершает серию из пяти препринтов, посвященных парадигмам программирования. в нём оценивается образовательное значение базовых парадигм программирования и дана характеристика ряда языков и систем учебного назначения. В первом препринте описаны особенности проявления и поддержки парадигм программирования, во втором, третьем и четвертом представлены результаты сравнения языков программирования разного уровня и приведены характерные черты поддержки основных парадигм программирования.

СПИСОК ЛИТЕРАТУРЫ

1. Абрамов В.Г., Трифонов Н.П., Трифонова Г.Н. Введение в язык Паскаль. – М.: Наука, 1988.
2. Андреева Т.А., Ануреев И.С., Бодин Е.В., Городняя Л.В., Марчук А.Г., Мурзин Ф.А., Шилов Н.В. Компьютерные языки как форма и средство представления, порождения и анализа научных и профессиональных знаний. // Тр. XV Всерос. научно-методической конф. «Телематика-2008». – С-Пб, 2008. – С. 77–78.
3. Андрей Петрович Ершов - ученый и человек / Отв. ред. А.Г. Марчук. – Новосибирск: Изд-во СО РАН, 2006. – 504 с. (Наука Сибири в лицах)
4. Бараз Л.С., Боровиков Е.В., Глаголева Н.Г., Земцов П.А., Налимов Е.В., Цикоза В.А. Язык программирования Рапира. – Новосибирск, 1987. – 34 с. – (Препр. / ВЦ СО АН СССР; № 767).
5. Бауэр Ф.Л., Гооз Г. *Информатика. Вводный курс*. – М.: Мир, 1990. Ч. 1, 2.
6. Бежанова М.М., Поттосин И.В. Современные понятия и методы программирования. – М.: Научный мир, 2000. – 191 с.
7. Брой М. Информатика. Основополагающее введение. – М.: Диалог-МИФИ, 1996. – 299 с.
8. Вирт Н. От Модулы к Оберону // Системная информатика. Вып 1: Проблемы современного программирования. – Новосибирск: Наука. Сиб. отд-ние, 1991. – С. 63–75
9. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления.– СПб.: БХВ-Петербург, 2002. – 608 с.
10. Гиглавый А.В., Гнездилова Г.Г., Гуткин М.Л. Программирование на языке Бейсик для школьной ЭВМ. – М.: ВЦ АН СССР, 1986

11. Городняя Л.В. Макетирование программ с помощью тестов и описаний // Языки и системы программирования. – Новосибирск, 1981 – С. 115–123.
12. Городняя Л.В. Некоторые диалекты Лиспа и сферы их приложения // Трансляция и преобразования программ. – Новосибирск, 1984. – С. 60–71.
13. Городняя Л.В. О конструировании учебных систем программирования. // Методы трансляции и конструирования программ. – Новосибирск, 1987.
14. Городняя Л.В. Основы функционального программирования. – М.: Интернет-Университет Информационных технологий. – http://www.intuit.ru, 2004. – 272 с.
15. Городняя Л.В. Схема изучения языка программирования с практикой на ЭВМ.// Теория и практика системной информатики и программирования. – Новосибирск, 1988.
16. Городняя Л.В. Функциональный подход к системному представлению прикладных программ учебного назначения. – Новосибирск, 1993. – 25 с. – (Препр. / РАН; Сиб. отд-ние. ИСИ; № 16)
17. Грогоно П. Программирование на языке Паскаль. – М.: Мир, 1982. – 382 с.
18. Дейкстра Э. Дисциплина программирования. – М.: Мир, 1978. – 275 с.
19. Дьяконов В.П. Язык программирования Лого. – М.: Радио и связь, 1991. – 145 с.
20. Евстигнеев В.А., Касьянов В.Н. Графы в программировании: обработка, визуализация и применение. – С-Пб.: ЕХП-Петербург, 2003. – 1104 с.
21. Звенигородский Г.А. Первые уроки программирования. – М: Наука, 1985. Серия «Библиотека “Кванта”»
22. Зуев Е.А. Программирование на языке Turbo-Pascal 6.0 -7.0. – М.: Веста, Радио и связь, 1993.
23. Йодан Э. Структурное проектирование и конструирование программ. – М. Мир , 1979. – 409 с.
24. Кнут Д. Искусство программирования. Основные алгоритмы. – М.: Мир, 1976. – 735 с.
25. Лавров С. С. Основные понятия и конструкции языков программирования. – М. Финансы и статистика, 1982. – 80 с.
26. Лавров С.С., Городняя Л.В. Функциональное программирование. Интерпретатор языка Лисп // Компьютерные инструменты в образовании. – СПб., 2002. – № 5.
27. Малиновский Б.Н. История вычислительной техники в лицах. – Киев: Фирма «КИТ», ПТОО «А.С.К.», 1995. – 383 с.
28. Марчук А.Г., Городняя Л.В., Мурзин Ф.А., Шилов Н.В. Классификация компьютерных языков: состояние, проблемы, перспективы // Тр. междунар. конф. «Космос, астрономия и программирование» (Лавровские чтения). Санкт-Петербургский государственный университет. – СПб., 2008. – С. 15–22.
29. Новосибирская школа программирования. (Перекличка времен) /Под ред. проф. И.В.Поттосина и к.ф.-м.н. Л.В. Городней. – Новосибирск, 2004. – 243 с.

30. Основы информатики и вычислительной техники: Проб. учеб. пособие для средних учебных заведений. Под редакцией А.П. Ершова, В.М. Монахова. – М.: Просвещение, 1986. – Ч. 1, 2.
31. .Парамзин А. В. Введение в программирование на языке Ассемблера. – Новосибирск, 1993. – 180 с.
32. Пратт Т., Зелковиц М. Языки программирования. Разработка и реализация./Под общей редакцией А.Матросова. – СПб.: Питер, 2002. – 688 с.
33. Фет Я.И. История информатики в России: ученые и их школы.
34. Хендерсон П. Функциональное программирование. – М.: Мир, 1983. – 349 с.
35. Хоар Ч. Взаимодействующие последовательные процессы. – М.: Мир, 1989 – 264 с.
36. Хьювенен Э., Сеппанен Й. Мир Лиспа. – М.: Наука, 1994. – Т.1, 2.
37. Черноножкин С.Л. Методы тестирования программ.
38. http://www.aplusplus.net/ A++ -- Введение в программирование на языке A++.
39. Backus J. Can programming be liberated from the von Neumann style? A functional stile and its algebra of programs. – Comm. ACM 21, 8, 1978, – P. 613-641.
40. Bergin, Joseph (1997) Karel++: A Gentle Introduction to the Art of Object-Oriented Programming. John Wiley & Sons, Inc. ISBN 0-471-13809-6
41. Graham P. ANSI Common Lisp. //Prentice Hall, 1996. – 432 p.
42. Jeff Leonard Levinsky The GROW Book. San Diego, California, Computer Systems Design Group. 1980, p.60
43. McCarthy J. LISP 1.5 Programming Mannual. – The MIT Press., Cambridge, 1963. – 106 p.
44. Ritchie D.M., Tompson K. The UNIX Time-Sharing System – Bell System Technical Journal. V. 57, N 6, 1978. – P. 1905-1929.
45. Schwartz J.T. Set Theory as a Language for Program Specification and Programming. – Courant Institute of Mathematical Sciences, New York University, 1970.
46. Weinberg G.M. The Psychology of Computer Programming. – New York: Van Norstand Reinhold Comp., 1971.
47. http://www.cs.ru.nl/elan/ Educational programming language ELAN
48. http://www.literateprogramming.com Сайт с материалами по грамотному программированию. Кнут Д. Literate programming.
49. http://homelisp.ru – Сайт с материалами по системе Home Lisp, созданной Б.Л. Файфелем.
50. http://sourceforge.net/projects/mozart-oz/ – Сайт с материалами по системе Mozart, поддерживающей учебный мульти-парадигматический язык программирования Oz.

Приложение

Аббревиатуры

Обозначение	*Расшифровка*
АМ	Абстрактная машина
АС	Абстрактный синтаксис
ЖЦП	Жизненный цикл программ
ЗШ	Заочная школа юных программистов
ЗШЮП	Заочная школа юных программистов
ИП	Императивное программирование
ЛП	Логическое программирование
ЛШ	Летняя школа юных программистов
ЛШЮП	Летняя школа юных программистов
ОИВТ	Основы информатики и вычислительной техники
ООП	Объектно-ориентированное программирование
ОС	Операционная система
ПП	Парадигма программирования
СД	Структуры данных
СП	Система программирования
ТД	Типы данных
УЯ	Учебный язык программирования
ФП	Функциональное программирование
ШИ	Школьная информатика
ШЮП	Школа юных программистов
ЯВУ	Язык высокого уровня
ЯНОП	Язык начального обучения программированию
ЯП	Язык программирования
ЯСВУ	Язык сверхвысокого уровня
ЯСП	Язык и система программирования
ЯФП	Язык функционального программирования

СОДЕРЖАНИЕ

[4] Части 1–4 – отдельные препринты.

Л.В. Городняя
ПАРАДИГМЫ ПРОГРАММИРОВАНИЯ

Часть 5
Учебные языки и системы программирования

Препринт
176

Рукопись поступила в редакцию 18.02.2015
Редактор Т. М. Бульонкова
Рецензент Ф.А. Мурзин

Подписано в печать 19.03.2015
Формат бумаги 60 × 84 1/16 Объем 3.4 уч.-изд.л., 3.75 п.л.
Тираж 60 экз.

Типография Оригинал-2, г. Бердск, ул. Олега Кошевого, 6, оф. 2
тел./факс: 8 (383) 328-32-38, (38341) 2-12-42, сот.: 8 913 987 77 67