Р Е Ф Е РА Т

Эволюция языков программирования.

2001г.

**ПЛАН.**

1. Языки программирования (ЯП).

2. Описание ЯП.

3. Технологии программирования.

4. CASE - системы.

5. Искусственный интеллект, экспертные системы.

6. Список использованной литературы.

1. Языки программирования (ЯП).

 Проведем ассоциации компьютера с человеком. У компьютера есть органы восприятия информации из внешнего мира - это клавиатура, мышь, накопители на магнитных дисках. У компьютера есть органы, "переваривающие" полученную информацию, - это центральный процессор и оперативная память. И наконец, у компьютера есть органы речи, выдающие результаты переработки. Современным компьютерам, конечно далеко до человека. Их можно сравнить с существами, взаимодействующими с внешним миром на уровне ограниченного набора безусловных рефлексов. Этот набор рефлексов образует систему машинных команд. На каком бы высоком уровне мы ни общались с компьютером, в конечном итоге все сводится к скучной и однообразной последовательности машинных команд. Каждая машинная команда является своего рода раздражителем для возбуждения того или иного безусловного рефлекса. Реакция на этот раздражитель всегда однозначная и "зашита" в блоке микрокоманд в виде микропрограммы. Эта микропрограмма и реализует действия по реализации машинной команды, но уже на уровне сигналов, подаваемых на те или иные логические схемы компьютера, тем самым, управляя различными подсистемами компьютера. В этом состоит так называемый принцип микропрограммного управления. Продолжая аналогию с человеком, отметим: для того, чтобы компьютер правильно питался, придумано множество операционных систем, компиляторов сотен языков программирования. Но все они являются по сути лишь блюдом, на котором по определенным правилам доставляется пища (программы) желудку (компьютеру). Только желудок компьютера любит диетическую, однообразную пищу - подавай ему информацию структурированную, в виде строго организованных последовательностей нулей и единиц, комбинации которых составляют машинный язык. Таким образом, внешне являясь полиглотом, компьютер понимает только один язык - язык машинных команд.

Программисту не нужно пытаться постичь значения различных комбинаций двоичных чисел, т.к. еще в 50-е годы программисты стали использовать для программирования символический аналог машинного языка, который назвали языком ассемблера. Этот язык точно отражает все особенности машинного языка. Именно поэтому язык ассемблера для каждого типа компьютера свой.

Язык программирования - это специальный язык, на котором пишут команды для управления компьютером. Языки программирования созданы для того, чтобы людям было проще читать и писать для компьютера, но они затем должны транслироваться (транслятором или интерпретатором) в машинный код, который только и может исполняться компьютером. Языки программирования можно разделить на языки высокого уровня и языки низкого уровня.

 Язык низкого уровня - это язык программирования предназначенный для определенного типа компьютера и отражающий его внутренний машинный код; языки низкого уровня часто называют машинно-ориентированными языками. Их сложно конвертировать для использования на компьютерах с разными центральными процессорами, а также довольно сложно изучать, поскольку для этого требуется хорошо знать принципы внутренней работы компьютера.

 Язык высокого уровня - это язык программирования, предназначенный для удовлетворения требований программиста; он не зависит от внутренних машинных кодов компьютера любого типа. Языки высокого уровня используют для решения проблем и поэтому их часто называют проблемно-ориентированными языками. Каждая команда языка высокого уровня эквивалентна нескольким командам в машинных кодах, поэтому программы, написанные на языках высокого уровня, более компактны, чем аналогичные программы в машинных кодах.

1. Описание ЯП

|  |  |  |
| --- | --- | --- |
| **Язык** | **Основное использование** | **Описание** |
| Ада | В обороне | Высокого уровня |
| Ассемблер | Работы, требующие детального контроля за аппаратным обеспечением, быстрого исполнения и программ малого размера | Быстрый и эффективный, но требующий определенных усилий и навыков |
| Бейсик | В образовании, бизнесе, дома | Прост в изучении |
| С | Системное программирование, универсальное программирование | Быстрый и эффективный, широко используется как универсальный язык |
| С++ | В объектно-ориентированном программировании | Основан на языке С |
| Кобол | Программирование в бизнесе | Жестко ориентирован на коммерческие задачи, легко научиться, но очень много операторов |
| Форт | Управление приложениями | Использует инверсную польскую запись |
| Фортран | Научная работа и вычисления | Основан на математических формулах |
| Лисп | Искусственный интеллект | Язык символов с репутацией трудно изучаемого |
| Модула-2 | Системное программирование и программирование в режиме реального времени, универсальное программирование | Высоко структурирован, предназначен заменить Паскаль для приложений "реального мира" |
| Оберон | Универсальное программирование | Небольшой, компактный язык, соединяющий многие черты Паскаля и Модула-2 |
| Паскаль | Универсальный язык | Высоко структурирован |
| Пролог | Искусственный интеллект | Символьно-логическая система программирования, в начале предназначенная для решения теорем, но сейчас использующаяся чаще для решения задач, связанных с искусственным интеллектом |

3. Технологии программирования.

 В основе того или иного языка программирования лежит некоторая руководящая идея, оказывающая существенное влияние на стиль соответствующих программ.

Структурное программирование.

Структурное программирование - методология программирования, базирующаяся на системном подходе к анализу, проектированию и реализации программного обеспечения. Эта методология родилась в начале 70-х годов и оказалась настолько жизнеспособной, что и до сих пор является основной в большом количестве проектов. Основу этой технологии составляют следующие положения:

* Сложная задача разбивается на более мелкие, функционально лучше управляемые задачи. Каждая задача имеет один вход и один выход. В этом случае управляющий поток программы состоит из совокупности элементарных подзадач с ясным функциональным назначением.
* Простота управляющих структур, используемых в задаче. Это положение означает, что логически задача должна состоять из минимальной, функционально полной совокупности достаточно простых управляющих структур. В качестве примера такой системы можно привести алгебру логики, в которой каждая функция может быть выражена через функционально полную систему: дизъюнкцию, конъюнкцию и отрицание.
* Разработка программы должна вестись поэтапно. На каждом этапе должно решаться ограниченное число четко поставленных задач с ясным пониманием их значения и роли в контексте всей задачи. Если такое понимание не достигается, это говорит о том, что данный этап слишком велик и его нужно разделить на более элементарные шаги.

Концепция модульного программирования.

Так же как и для структурной технологии программирования, концепцию модульного программирования можно сформулировать в виде нескольких понятий и положений:

* Функциональная декомпозиция задачи - разбиение большой задачи на ряд более мелких, функционально самостоятельных подзадач - модулей. Модули связаны между собой только по входным и выходным данным.
* Модуль - основа концепции модульного программирования. Каждый модуль в функциональной декомпозиции представляет собой "черный ящик" с одним входом и одним выходам. Модульный подход позволяет безболезненно производить модернизацию программы в процессе ее эксплуатации и облегчает ее сопровождение. Дополнительно модульный подход позволяет разрабатывать части программ одного проекта на разных языках программирования, после чего с помощью компоновочных средств объединять их в единый загрузочный модуль.
* Реализуемые решения должны быть простыми и ясными. Если назначение модуля непонятно, то это говорит о том, что декомпозиция начальной или промежуточной задачи была проведена недостаточно качественно. В этом случае необходимо еще раз проанализировать задачу и, возможно, провести дополнительное разбиение на подзадачи. При наличии сложных мест в проекте их нужно подробнее документировать с помощью продуманной системы комментариев. Этот процесс нужно продолжать до тех пор, пока вы действительно не добьетесь ясного понимания назначения всех модулей задачи и их оптимального сочетания.
* Назначение всех переменных модуля должно быть описано с помощью комментариев по мере их определения.

Объектно-ориентированное программирование (ООП).

Идея ООП заключается в стремлении связать данные с обрабатывающими эти данные процедурами в единое целое - объект. ООП основано на трех важнейших принципах, придающих объектам новые свойства. Этими принципами являются инкапсуляция, наследование и полиморфизм.

* Инкапсуляция - объединение в единое целое данных и алгоритмов обработки этих данных. В рамках ООП данные называются полями объекта, а алгоритмы - объектными методами.
* Наследование - свойство объектов порождать своих потомков. Объект - потомок автоматически наследует от родителей все поля и методы, может дополнять объекты новыми полями и заменять (перекрывать) методы родителя или дополнять их.
* Полиморфизм - свойство родственных объектов (т.е. объектов, имеющих одного общего родителя) решать схожие по смыслу проблемы разными способами.

4. CASE - системы.

Представление о CASE - комплексах связано в нашем сознании с чем - то, не имеющим отношения к обычному программированию.

В Америке из - за сильнейшнй конкуренции CASE - средства используются подавляющим большинством фирм - разработчиков программного обеспечения. Мощный толчок CASE - средства получили в пору внедрения объекто - ориентированной технологии разработки ПО, когда старого, проверенного временем метода проектирования "сверху вниз" стало явно недостаточно. К тому же появились технологии объектного моделирования Booch, OMT, UML, сами по себе весьма сложные для привязки к языкам программирования, чтобы оперировать ими вручную.

 Сегодня лидирующей в мире CASE-системой считается Rational Rose корпорации Rational Software. Система Rational Rose нацелена на создание модулей с использованием языка Unified Modeling Language (UML). Кстати, UML стал стандартным языком объектно-ориентированно разработки не без подачи Rational Software, которая не только выпускает программные продукты, где используются UML, но и активно принимает участие в организации Object Management Group (OMG), занятой созданием и обновлением спецификаций языка UML, технологии распределенных вычислений CORBA и т.д. в компании Rational работают три создателя и евангелиста объектно-ориентированной разработки и языка UML. Это Гради Буч, Айвар Джекобсон и Джим Рамбаух.

 Последняя версия CASE-системы компании Rational Software Rational Rose 98 уже вовсю применяется для создания коммерческого ПО и поддерживает популярные языки программирования Java, Cu++, Смолток, Ада, Visual Basic, Power Builder и Forte. Кроме того, пакет Rose 98 способен генерировать описания на языках Interface Definition Language (IDL) для приложений CORBA и Data Definition Language (DDL) для приложений доступа к базам данных, в том числе и Oracle 8. Разумеется, поддержка того или иного языка программирования зависит от того, о какой редакции пакета Rational Rose 98 идет речь.

К примеру, нельзя требовать многого от самого простого варианта пакета - Rose 98 Modeler Edition. Зато Rose 98 Enterprise Edition оснащен от души.

Нельзя не отметить, что система Rose - признанный лидер среди средств визуального моделирования, и, мспользуя ее, можно интерактивно разрабатывать архитектуру создаваемого приложения, генерировать его исходные тексты и параллельно работать над документированием разрабатываемой системы. С помощью Rational Rose можно создавать новые модели на базе обратного разбора двоичных com модулей или исходные тексты прикладных программ и библиотек классов.

 Преимущества от применения Rational Rose 98 значительны:

1. Сокращение цикла разработки приложения.
2. Увеличение продуктивности работы программистов.
3. Улучшение потребительских качеств создаваемых программ за счет ориентации на пользователей и бизнес.
4. Способность вести большие проекты и группы проектов.
5. Возможность повторного использования уже созданного ПО за счет упора на разбор их архитектуры и компонентов.
6. Язык UML служит универсальным "мостиком" между разработчиками из разных отделов.

5. Технологическая схема решения задач.

Технологическая схема, в которой пользователь, желающий решить свою задачу на ЭВМ, обращается за консультацией к специалисту по алгоритмизации (формализации), а тот, в свою очередь, к программисту, создающему программу на основе формальной модели решаемой задачи, сейчас уходит из жизни.

Она оказывается неэффективной по ряду причин. Во-первых, пользователь не всегда точно знает, чего он хочет, и алгоритмист, когда формализует задачу, поневоле упрощает ее, теряет или отбрасывает многое из того, что пользователь знает, но либо не сообщил алгоритмисту, либо опрометчиво согласился на предлагаемые упрощения. Полученная после этого модель программируется и реализуется на ЭВМ. А пользователь явно не доволен. Только теперь он понял, что ему нужно, и видит, что ему дали не то, что ему нужно. После этого начинается второй раунд взаимодействия, за ним, возможно третий, четвертый и т.д.

Почему так происходит? Скорее всего, потому, что пользователь, работающий в областях, где формализация еще не проявила себя в полную силу - сейчас их принято называть плохо структурированными проблемными областями, - просто не ведает о том, какие же знания необходимо сообщить алгоритмисту о своей задаче, чтобы полностью удовлетворить и его и себя.

Возникает идея - убрать из технологической схемы алгоритмиста, сократить пользователя к ЭВМ: пусть он теперь со своей задачей обращается прямо к программисту. Правда, для этого нужно, чтобы программист повысил свой профессиональный уровень, овладел бы "смежной" профессией алгоритмиста. Но тогда программисты станут более дефицитными, чем сейчас, ибо требования к ним резко возрастут. А ведь армия программистов и так не успевает обслужить всех желающих, и если темпы роста пользователей не уменьшатся, то все население земного шара будет состоять из пользователей и программистов.

 Ясно, что этот путь тупиковый. К тому же он не решает основной проблемы - прямого доступа пользователей к ЭВМ и не устраняет непонимания между пользователем и программистом, возникающего из-за отсутствия у программиста знаний о проблемной области пользователя, а у пользователя - о способах решения задач на ЭВМ.

 А что если и программиста удалить из технологической цепи пользователь - ЭВМ? Это можно сделать, если пользователи научатся программировать, станут профессионалами в двух областях - в своей собственной и в программировании. Насколько это возможно? И сейчас существуют специалисты, овладевшие искусством программирования настолько, что сами свободно работают с вычислительной машиной. Но таких специалистов не много, так как овладеть двумя совершенно разными профессиями - дело нелегкое. И, как правило, в одно из них человек остается все-таки полупрофессионалом.

Есть ещё один путь приобщения специалиста к современной вычислительной технике - это повышение возможностей самих ЭВМ, повышение уровня их "интеллекта". Программиста можно убрать из технологической цепи решения задачи лишь тогда, когда в самой ЭВМ появиться "автоматический программист", который будет взаимодействовать с пользователем, и помогать ему составлять программы. Так возникает идея ЭВМ нового - пятого поколения. В отличие от ЭВМ предшествующих поколений новые машины должны иметь средства для интеллектуального взаимодействия с пользователем на его профессиональном естественном языке. Другими словами не пользователь приближается к ЭВМ, а сама ЭВМ становится интеллектуальным собеседником и помощником пользователя.

Индустрия искусственного интеллекта.

Бум, возникший в конце семидесятых годов в искусственном интеллекте и приведший к созданию новой отрасли промышленности, не случаен. Три причины вызвали его.

Первая - угроза всеобщей мобилизации населения земного шара в программисты привела к идее пятого поколения ЭВМ. Но создание таких ЭВМ требует разработки средств автоматического выполнения функций алгоритмиста и программиста, то есть интеллектуальных функций по формализации задач и составлению программ для их решения. А это уже сфера искусственного интеллекта, ибо одно из толкований целей этой науки состоит как раз в утверждении, что она должна создавать методы автоматического решения задач, считающихся в человеческом понимании интеллектуальными. Это означает, что создание ЭВМ пятого поколения невозможно без использования достижений, накопленных в искусственном интеллекте.

Вторая - развитие робототехнических малолюдных или безлюдных производств. На современных промышленных предприятиях происходит активное внедрение автоматических систем, в которых широко используются интеллектуальные роботы. Прогресс в этой области во многом зависит от того, насколько роботы могут хранить в своей памяти необходимую сумму знаний о профессии, которой они овладевают.

Третья - необходимость передавать на ЭВМ задачи из плохо структурированных проблемных областей. Именно для них нужно автоматизировать труд алгоритмиста, его способность формализовать то, что с трудом поддается формализации. Путь решения этой проблемы - формализация знаний, которые есть у профессионалов в данной проблемной области, но хранятся в их памяти в виде неформализованных соображений, умений и навыков. Такие профессионалы являются экспертами своего дела, а получаемые от них знания обычно называют экспертными. Если в базу знаний системы заложить знания подобного типа, то система будет называться экспертной.

 ЭВМ пятого поколения, и интеллектуальные роботы, и экспертные системы, и многие другие интеллектуальные системы обладают одним общим свойством: их работа основывается на знаниях, хранимых в базе знаний системы. Их часто так и называют - системами, основанными на знаниях.

Экспертные системы.

Экспертные системы могут не только найти решение той или иной задачи, но и объяснить пользователю, как и почему оно получено. Это означает, что в экспертных системах реализована возможность "самоанализа", в них появилась возможность рассуждать о знаниях и манипулировать ими. А значит, появилась и возможность иметь знания о знаниях, т.е. метазнания. С их помощью в экспертных системах стала возможной оценка знаний с точки зрения их полноты и корректности, а также реализуется "функция любопытства", связанная с активным поиском связей между хранящимися в памяти знаниями, их классификацией и пополнением за счет разнообразных логических процедур.

 В экспертных системах сделан важный шаг - знания, хранящиеся в системе, стали объектом ее собственных исследований.

Потенциально человек способен к овладению любым видом интеллектуальной деятельности. Он может научиться играть и в шахматы, и в морской бой, и в любые другие игры, ибо он обладает универсальными метапроцедурами, позволяющими ему создать процедуры решения конкретных интеллектуальных задач.

 Развитие теории искусственного интеллекта в конце шестидесятых годов началось с осознания именно этого факта. У новой науки появился свой специфический объект исследований и моделирования - универсальные метапроцедуры программирования интеллектуальной деятельности. В их числе имеются метапроцедуры общения, обучения, анализа воспринимаемой системой информации и многие другие. Но, несомненно, центральное место здесь занимают те метапроцедуры, которые связаны с накоплением знаний и использовании их при решении интеллектуальных задач. Именно эти метапроцедуры находят свое воплощение в экспертных системах.

 Существующие сейчас экспертные системы принято делить на два класса: консультационные и исследовательские. Первые призваны давать советы, когда у пользователя возникает необходимость в них, а вторые - помогать исследователю решать интересующие его научные задачи.

Внешняя память

Процессор

Решатель

База знаний

Система поддержки базы знаний

Система объяснения

Система общения

Блок обучения

 Интеллектуальный интерфейс

 Пользователь

 Рисунок демонстрирует общую структуру консультационной экспертной системы.

Система общения позволяет вводить в экспертную систему информацию на, ограниченном рамками профессиональной области, естественном языке и организует ведение диалога с пользователем. Эта система сообщает пользователю о непонятных для нее словах, о допущенных им ошибках, предлагает наборы действий, которые пользователь при желании может выполнить. Если пользователь еще не освоил "этику приема", то в дело включается блок обучения; в диалоговом режиме он постепенно обучает пользователя с общению с ЭВМ, учит его, используя примеры, решению задач. Пользователь может обращаться к этому учителю, когда захочет, - система всегда найдет время для пояснения непонятных пользователю моментов.

Решатель осуществляет поиск вывода решения, нужного пользователю на основе тех знаний, которые хранятся в базе знаний системы. Он играет роль мозгового центра системы. Чтобы функции решателя в консультационной экспертной системе стали более понятными, рассмотрим конкретный пример. Предположим, что в полевых условиях археолог столкнулся с находками, которые поставили его перед задачей датировки раскапываемого объекта. Известно, что точная датировка во многих случаях вещь весьма сложная. Она требует тщательного изучения находок, привлечения огромного по объему сравнительного материала из находок других археологов, требует от археолога умения делать правильные логические выводы, выдвигать гипотезы и отвергать их на основании найденного. При работах на раскопках рядом может не быть тех специалистов, которые могли бы оказать квалифицированную помощь. Именно для такой ситуации предназначена консультационная экспертная система. В ее базе знаний могут храниться огромное количество накопленных ранее фактов и установленных связей между этими фактами, а также мнения (не всегда совпадающие между собой) ведущих специалистов в данной области.

Когда археолог через систему общения обращается к системе за консультацией, то она может начать с того, что потребует ввести описание всех тех находок (на языке, понятном системе), которыми этот археолог располагает. Получив в свое распоряжение эти описания, экспертная система начинает формировать логический вывод. От исходных фактов, введенных в нее, и с помощью тех взаимосвязей, которые должны существовать между фактами, она выводит гипотезы, которые не противоречат наблюдаемым фактам. Если эта гипотеза однозначна, то она сообщается пользователю. Если имеет альтернативные возможности, то экспертная система может задать археологу дополнительные уточняющие вопросы, например, о характере рисунков на остатках найденной керамики, которые еще не были сообщены системе. Если археолог не может сообщить системе никаких новых дополнительных сведений, то ему будет сообщено несколько гипотез о датировке. При этом каждая гипотеза может оцениваться некоторым весом достоверности. Например, ответ может иметь вид: "Данный объект относится к периоду А с достоверностью 15% и к периоду В с достоверностью 85%". Если при дальнейших раскопках будет обнаружен другой предмет, то он датируется периодом В как наиболее вероятным. Для каждого вновь найденного предмета могут быть получены вероятности датировки, а затем все результаты могут быть проанализированы совместно.

 Информация в базе знаний не хранится, как зерно в элеваторе, просто сваленное в бункер. В этом случае база знаний не смогла бы обеспечить эффективную работу решателя. В экспертной системе существует специальный комплекс средств, с помощью которых в базе знаний наводится необходимый порядок. Информация здесь классифицируется, обобщается, оценивается ее непротиворечивость, отдельные информационные единицы объединяются связями различного типа. Другими словами, в базе знаний возникает структурированная модель проблемной области, в которой отражены все ее особенности, закономерности и способы решения задач. Всеми этими процедурами заведует система поддержки базы знаний.

 Система объяснения - важнейшая отличительная компонента экспертных систем. К ней пользователь может обращаться с вопросами типа "Что есть Х?", "Как получен У?", "Почему получен У, а не Z?" и "Зачем нужен Х?". За каждым таким вопросом скрывается свой комплекс процедур, выполнение которых позволяет дать пользователю интересующий его ответ. Вопрос "Что есть Х?" требует выдачи пользователю всей информации о Х, которой система располагает, что может потребовать весьма непростых поисковых процедур в базе знаний. Эти процедуры реализуются в решателе, так как во многих случаях для ответа на вопрос пользователя надо из исходных фактов, хранящихся в базе, получить логическим путем новые производные факты.

 Вопрос "Как получен У?" означает, что пользователь хочет ознакомиться с тем, как рассуждала система, шаг за шагом выводя из сообщенных ей пользователем фактов свое заключение. Для ответа на такой вопрос система объяснения должна обратиться в решатель, в памяти которого, как на экране электронно-лучевой трубки с послесвечением, должен некоторое время сохраняться "трек" того пути, который прошел решатель.

 Вопрос "Почему получен У, а не Z?" требует от экспертной системы умения обосновывать отказ от гипотез. В том же решателе хранится информация об альтернативном выборе между У и Z, который один или не один раз возникал на пути поиска решения. В этих "точках разветвления" система выбирала путь, ведущий к У, а не тот, который вел к Z. Использованные в этот момент соображения, определявшие выбор, выдаются пользователю.

 Наконец ответ на вопрос "Зачем нужен Х?", возникающий в ситуации, когда экспертная система просит пользователя ввести в нее информацию о Х, требует выполнения процедур обоснования необходимости сведений о Х для получения решения. Эти обоснования извлекаются из модели проблемной области, хранящейся в базе знаний.

 Возможны, по-видимому, и другие типы вопросов пользователя к системе объяснения, но и приведенных достаточно, чтобы понять, сколь важна ее роль: только она делает выдаваемые решения понятными и обоснованными для пользователя.

 Почти так же, как и консультативные, устроены исследовательские экспертные системы, но в них имеются еще и блоки, в которых выполняются все необходимые для специалиста расчеты. Можно сказать, что экспертные системы такого типа - это симбиоз ЭВМ пятого поколения и консультационных экспертных систем.

6. Список использованной литературы.

1. Журнал "Наука и жизнь" № 6 1987г.
2. Юров В., Хорошенко С. Assembler:учебный курс. СПб:Издательство "Питер", 1999г.
3. Фаронов В.В TurboPascal 7.0:начальный курс. М:Издательство "Нолидж", 1998г.
4. Карманный словарь "Computing & Multimedia". М:Издательство "Внешсигма", 1996г.
5. Журнал "Мир ПК" №4 1999г.