УДК 377.352

Мельник О.С.

Уманський державний педагогічний університет імені Павла Тичини, м.Умань

ВИКОРИСТАННЯ МОВ ПРОГРАМУВАННЯ ПРИ ПІДГОТОВЦІ ФАХІВЦІВ ТЕХНІЧНИХ СПЕЦІАЛЬНОСТЕЙ

Анотація. В статті викладені основні теоретичні та методологічні положення методичної системи навчання мов програмування при підготовці фахівців у вищих навчальних закладах I-II рівнів акредитації. Наведено основні переваги та недоліки використання деяких мов програмування при розв'язанні певних задач.

Ключові слова: мова програмування; підготовка фахівців; алгоритм; компіляція; ЕОМ.

Аннотация. В статье изложены основные теоретические и методологические положения методической системы обучения языкам программирования при подготовке специалистов в высших учебных заведениях I-II уровней аккредитации. Приведены основные преимущества и недостатки использования некоторых языков программирования при решении определенных задач. Показано, что знание компьютера способствуют развитию и реализации творческого потенциала ученика, обеспечивают качественно новый уровень его интеллектуальной и эмоционально-нравственной культуры, создают внутреннюю потребность в саморазвитии и самообразовании.

Ключевые слова: язык программирования; подготовка специалистов; алгоритм; компиляция; ЭВМ.

Annotation. In the article the basic theoretical and methodological provision of methodological training system programming languages for training specialists in higher educational institutions I-II levels of accreditation. The basic advantages and disadvantages of using several programming languages in solving certain problems.

Computer skills contribute to the development and implementation of student creativity, provide a new level of emotional and intellectual and moral culture, creating an internal need for self-development and self-education, promote individual adaptation to rapidly changing social, economic and information technology environments. Insistently raises the question of the introduction of the general college courses (chapters) on the study of the elements of cybernetics and computer programming in the discussion along with Methodists involved are known mathematics. At the same time meaningfully investigated and methodological aspects of inter algorithmic impact on traditional objects, primarily through the mathematics language, algorithmic content focus, increased attention to the applied side of knowledge, etc. The long-term significance of these studies is that they looked exactly those aspects profound impact ideas and methods of programming content and learning, the lack of which became fully apparent in conditions of strong expansion of computerization College thundered decades later. The proliferation of computers and programming the sector of mathematical culture began to acquire independent value, you only need to supplement it by the most universally algorithmic components. Formed in this way a set of specific concepts and skills that defines a new element of general culture of every modern man and claims of this reason, the inclusion of a general school education (as in the category of new concepts of the theory and methods of schooling), called algorithmic culture of students.

Keywords: programming language; training; algorithm; compilation; Computers..

Постановка проблеми. Курс інформатики в системі професійної освіти з кожним днем стає все важливішим за рахунок повсюдної комп'ютеризації та необхідності комп'ютерної грамотності учнів, так як в подальшому це буде комп'ютерна грамотність суспільства. Підходів до викладання інформаційних технологій в коледжах існує декілька і вони різні [4].

Знання комп'ютера сприяють розвитку і реалізації творчого потенціалу учня, забезпечують якісно новий рівень його інтелектуальної та емоційно-моральної культури, створюють внутрішню потребу в саморозвитку і

самоосвіті, сприяють адаптації особистості в швидко змінюються соціально-економічних та інформаційно-технологічних умовах.

За роки становлення і вдосконалення курси інформатики істотно змінювалися. З урахуванням того, яку роль ця дисципліна грала в навчанні, в розвитку методичних систем можна виділити як мінімум шість досить чітко визначених етапів.

Аналіз останніх досліджень і публікацій. Думки про те, яку мову краще викладати в коледжі, відрізняються: від того, що програмування вивчати не потрібно, а слід просто піднімати комп'ютерну грамотність і освоювати офісні програми (як на Заході), до того, що потрібно вивчати операційні системи і кілька мов програмування різних рівнів абстракції та з різними парадигмами. Це крайні випадки, але золоту середину знайти непросто.

В першу чергу, нам потрібно визначити мету; навчити студентів логічно і алгоритмічно мислити; ознайомити з комп'ютерами на побутовому рівні, щоб студенти вміли користуватися інтернетом, електронною поштою і текстовими редакторами; закласти базові знання, необхідні для майбутніх інженерів, математиків, фізиків і фахівців з інформаційних технологій.

Безумовно, в науці про програмуванні є фундаментальна складова, але визначити її не просто. Деякі вважають, що не так важливо, яку мову програмування вивчати: на уроках інформатики потрібно вчити не мови програмування, а методам програмування і системного підходу вирішення завдань. Потрібно розвивати алгоритмічне мислення і на прикладах знайомитися з принципами побудови сучасних комп'ютерних систем [2-3].

Мета статті – розкрити основний зміст використання мов програмування при підготовці фахівців технічних спеціальностей.

Виклад основного матеріалу дослідження. Виявляється, що у кожного викладача є свій список вимог до навчальної мови програмування. Наприклад: простий, інтуїтивний синтаксис, наявність високорівневих інструментів для виявлення і недопущення помилок і для налагодження програм, наявність якісної документації з прикладами, наявність доброзичливого середовища розробки, міжплатформену.

Колись найбільш популярними мовами програмування в школах світу були Бейсік і Паскаль. Бейсік завжди вважалася самою простою мовою програмування, а Паскаль - найкращою мовою для навчання програмуванню. Бейсік створювався за часів, коли людство не мало ніякого досвіду створення комп'ютерних систем, і заснований на застарілих і не виправданих себе принципах. Власне, ніякої фундаментальної цілісної ідеї в основі Бейсіка не лежить. Сьогодні є прості і при цьому більш наочні і ідейно замкнуті мови програмування, ніж Бейсік.

Паскаль зручний в навчальних цілях; адже саме для них він і створювався. Студенти швидко навчаються вирішувати з його допомогою алгоритмічні завдання. Але так виходить, що вивчати Паскаль корисно тільки для того, щоб писати програми на Паскалі. А якщо потрібно створити справжній програмний продукт, Паскаль виявляється незручний. І студентам, які знають тільки Паскаль, доводиться перенавчатися, що часто складніше, ніж вивчити правильні мови і технології з нуля.

Часто чуєш від викладачів шкіл і вузів, що ж краще Паскаль, ніж Бейсік. І краще Java, а не Паскаль: в Java є прибирання сміття, а це дуже зручно для вивчення програмування. А ще краще яка-небудь сценарна слабо нетипізована мова [3].

Але є й інші думки, перша мова програмування повинна бути вимогливою до учня. Необхідно, щоб учень мав чітке уявлення про те, що його програма робить на кожному кроці, і вміти записувати алгоритми на суворій формальній мові, без зайвих «поблажок», які є, наприклад, в мові Перл, де можна писати круглі дужки навколо аргументів функцій, а можна не писати, і робити інші подібні речі.

Перша мова повинна бути типізованою, бо змішання цілих чисел, дійсних чисел і текстових змінних призводить у початківців-програмістів до неправильного уявлення про методи зберігання даних в пам'яті комп'ютера. Чим більше повідомлень про помилки учні побачать від компілятора, і чим більше з цих повідомлень вони зрозуміють, тим більше фундаментальних знань про програмування вони отримають. Паскаль - непогана мова в цьому сенсі.

Особливо приємно, що в ньому є перевірка на приналежність індексу масиву допустимому безлічі значень. Це студентам дуже корисно. Але в ньому немає покажчиків, і учні не розуміють простої речі - того, що у змінних є місце (адресу), де вона зберігається, і значення (те, що там зберігається). З мовою Сі інша проблема: в ній багато складних конструкцій. З іншого боку, ніхто не змушує вчителів показувати всі глибини Сі. З ним можна працювати на тому ж рівні, що і з Паскалем, не займаючись складними махінаціями з покажчиками і не використовуючи складних конструкцій.

Альтернатив багато. Нині є цілий ряд мов програмування, які постійно еволюціонують, розщеплюються і зливаються. Це вже згадані вище мови Форт, Ruby і Python [4].

Перерахуємо ключові фактори, що керують відбором. Надання мовою високорівневих засобів контролю за цілісністю і безпомилково кода на першому етапі складання проектів. Це стосується в першу чергу до мов Java, Haskell, і Python. Мови намагаються робити такими, щоб програміст просто не міг допускати помилок. А якщо помилки все-таки робляться, то на етапі компіляції (трансляції) вони повинні знаходитися. Зокрема, помилка в одному символі не повинна призводити до того, що програма Компільо і запускається (а таке буває, наприклад, в мовах Бейсік і Perl, якщо не вказано явно спеціальний режим strict). Мова Java створювалася в контексті аналізу типових помилок і проблем, що виникають в проектах на мові Сі ++. Творці Java постаралися внести в синтаксис і базову парадигму такі обмеження, щоб типові помилки програмістів на Сі ++ просто не могли з'явитися в проектах на Java. Це дуже важлива ідея: якщо вміло укласти себе в рамки, можна отримати вигоду.

Слід зазначити, що в великих корпораціях часто програмістам видається список правил оформлення програм і набір конструкцій, які не можна використовувати в коді, не дивлячись на те, що сама мова їх допускає. Зайва гнучкість мови іноді шкідлива, так як дозволяє програмістам писати каламутні і заплутані програми. Нові мови програмування роблять так, щоб не спокушати програмістів і не давати їм можливості писати плутано і з помилками.

Чистота і ясність коду, читаність коду. Далі всіх тут просунувся, мабуть, Рубі. Сьогодні на всіх офіційних сайтах програмних засобів серед перших переваг вказується "природність синтаксису" або "близькість до природної мови" (зазвичай англійської). Звичайно, це важливий фактор. Давно минув час, коли люди підлаштовувалися під комп'ютери і переводили свої ідеї та алгоритми в машинну мову нулів і одиниць. Сьогодні комп'ютери все більше і більше підлаштовуються під людську мову. Це зручно. Збільшується швидкість написання програм, хоча зазвичай це йде на шкоду швидкості виконання і взагалі раціональності отриманої програми.

Чистота і цілісність парадигми, закладеної в основу мови. Наприклад, мови Smalltalk і Ruby базуються на чистій об'єктно-орієнтованої парадигми, а Haskell - на чистій функціональній парадигмі. Ця чистота корисна, щоб програміст чітко уявляв модель, якої він обмежений, і в термінах якої йому потрібно мислити при проектуванні програми.

Простота синтаксису, прозорість інтерпретації мовних конструкцій. Наприклад, синтаксис мови Python настільки простий, що його опис поміщається на одну сторінку. Це дозволяє програмісту завжди розуміти те, що він написав. Простота синтаксису, яка з одного боку є обмеженням, може бути дуже корисною, оскільки дозволяє писати прості програми і не думати про те, як же саме компілятор (інтерпретатор) відтранслює ту чи іншу програму.

Багатогранність і гнучкість, можливість писати складні програми коротко і красиво. Такою властивістю володіють зараз мови Perl, Ruby, Python. Але слід зазначити, що така універсальність мов може мати і недоліки, так як часто призводить до зайвого ускладнення синтаксису. Наприклад, дуже багатогранний Perl, він багатий різними конструкціями і хитрими штучками, які дозволяють записувати складну логіку дуже коротко. В результаті дуже легко написати програми, які потім неможливо читати. Втім, те ж саме стосується і мов C і C++. Мовам Ruby, Python багатогранність дається з меншими втратами, ніж Perl і C++.

Наявність стандартних бібліотек і наявність можливостей інтеграції проектів один з одним і з іншими системами і технологіями. Мови надають

можливості для роботи з базами даних, для створення графічних інтерфейсів, для роботи з мережевими протоколами і створення додатків з архітектурою клієнт-сервер. Сьогодні йде безперервне змагання між скриптовими мовами програмування типу PHP, Ruby, Python, Perl та ін. В тому, наскільки добре розвинені в них кошти інтеграції з різними технологіями. Хтось уміє працювати з OpenGL, а хтось ні [1].

Можливість розробляти адаптивні системи. Мова має бути такою, щоб програми, написані на ній, чи не були відсталими і неповороткими. Мова повинна допускати можливість внесення малих змін в код, щоб підлаштуватися під динамічно змінюючим і ускладнюючим завданням. Наріжними каменями адаптивності мовних програмних засобів є гнучка багаторівнева модульність (як у мов Java, Ruby, Python) простота засобів експорту та імпорту функціональності (маються на увазі засоби, спрямовані на те, щоб проекти могли ділиться один з одним класами, об'єктами і функціями) і засоби підтримки рефакторинга - глобальних революційних змін коду, що проходять крізь модулі і спрямованих на поліпшення читання коду і позбавлення від нагромадженої в процесі еволюції вантажу непотрібної функціональності.

Мова Python сьогодні переможець в номінації "простота синтаксису", а Perl більш, ніж будь-яка інша мова зручна для обробки текстів і CGI-скриптинга. Мова Python в принципі створювалася як мова інтегратор. З його допомогою можна інтегрувати різні додатки і створювати свої власні пакети і нові макромови.

В основі програмування для ЕОМ лежить поняття алгоритмізації, що розглядається в широкому сенсі як процес розробки і опису алгоритму засобами заданої мови. Однак алгоритмізація як метод, на який спирається спілкування людини з формалізованим виконавцем (автоматом), пов'язана не тільки зі складанням програм для ЕОМ. Так само як і моделювання, алгоритмізація - це загальний метод інформатики. Процеси управління в різних системах зводяться до реалізації певних алгоритмів. З побудовою алгоритмів пов'язано і створення найпростіших автоматичних пристроїв, і розробка автоматизованих систем управління складними виробничими процесами.

Фундаментальні основи алгоритмізації лежать в суто теоретичної області сучасної математики - теорії алгоритмів, проте, алгоритмізація в широкому практичному сенсі розуміється як набір певних практичних прийомів, заснованих на особливих специфічних навичок раціонального мислення про алгоритми. Добре відомо, що уявлення про алгоритмічних процесах і способи їх опису формувалися (хоча і неявно) в свідомості учнів при вивченні шкільних дисциплін ще до появи інформатики та обчислювальної техніки. Основна роль серед шкільних дисциплін при цьому випала математиці, в якій операційні та алгоритмічні дії спочатку становили один з істотних елементів навчальної діяльності. Дійсно, вміння формулювати, записувати, перевіряти математичні алгоритми, а також точно виконувати їх завжди складали найважливіший компонент математичної культури студента, хоча сам термін "алгоритм" міг при цьому в шкільних навчальних програмах і не вживатися.

З поширенням ЕОМ і програмування цей сектор математичної культури став набувати самостійного значення, потрібно лише доповнити його за рахунок найбільш загальнозначущих компонентів алгоритмізації. Утворена таким чином сукупність специфічних понять, умінь і навичок, яка визначає новий елемент загальної культури кожної сучасної людини і претендує з цієї причини на включення в загальну шкільну освіту (як і в розряд нових понять теорії і методики шкільного навчання), отримала назву алгоритмічної культури учнів.

Висновки. Все наполегливіше ставиться питання про введення в коледжі загальноосвітніх курсів (розділів), присвячених вивченню елементів кібернетики, ЕОМ і програмування, в його обговоренні поряд з методистами беруть участь відомі математики. У той же час досліджуються змістовно-методичні аспекти міжпредметних впливу алгоритмізації на традиційні предмети і, перш за все, математику через мову, алгоритмічну спрямованість змісту, посилення уваги до прикладної стороні знань і т.п. Перспективна значимість цих робіт в тому, що вони розглядали саме ті аспекти глибокого впливу ідей і методів програмування на утримання і процес навчання, недолік

яких в повній мірі став проявлятися в умовах рішучої експансії комп'ютеризації коледжу, гримнула десятиліття потому.

Література

1. Байденко, В.І. Компетенції у професійній освіті: (до освоєння компетентнісного підходу): Метод. посібник. - М.: Дослідницький центр проблем якості підготовки фахівців, 2005. - 114 с.

2. Бакланова М.Л. Сучасний стан реалізації положень Болонської декларації у вищих навчальних закладах I-II рівнів акредитації // Тези науково-практичної конференції «Організація навчально-виховного процесу у вищій школі в світлі входження України в Європейський освітній простір». – Бердянськ: БДПУ, 2007. – С. 53-55.

3. Богданович В. Ефективність творчої діяльності на заняттях інформатики в коледжах. // Наука, освіта, молодь: матеріали Дев’ятої Всеукраїнської студентської наукової конференції. – Умань: ФОП Жовтий О.О., 2016 – с.32-34.

4. Лапчик, М.П. Методика преподавания информатики [Текст] / М.П. Лапчик, И.Г. Семакин, Е.К. Хенер.- М.: Академия, 2007.- 622 с.

5. Олейникова, О.Н. Розробка модульних програм, заснованих на компетенціях: навчальний посібник / О.М. Олейникова, А.А. Муравйова, Ю.В. Коновалова, Є.В. Сартакова. - М.: Альфа-М, 2005. - 288с.