Гришко Л.В.
Черкаський національний університет

Концептуальні підходи до навчання основ програмування у вищій школі

Зміни, що відбуваються сьогодні в різних сферах професійного і суспільного життя, пов'язані перш за все з широким впровадженням засобів інформаційних та комунікаційних технологій. Вони вимагають від фахівця у галузі комп'ютерних наук, зокрема, від програміста, вміння розробляти програмне забезпечення для розв'язування задач науки, техніки, економіки і управління; створювати і використовувати інформаційні моделі складних процесів і явищ; виконувати дослідницьку роботу у сферах, що використовують методи прикладної математики і комп'ютерні технології; використовувати інформаційні технології у проектно-конструкторській, управлінській і фінансовій діяльності.

Однією з фундаментальних дисциплін, що формує теоретичну і практичну базу у професійній підготовці майбутніх програмістів, є програмування. Перелік напрямів і спеціальностей МОН України включає сьогодні більше двадцяти спеціальностей, для яких програмування є фундаментом у підготовці майбутнього фахівця. Крім того, на сучасному етапі програмування є невід'ємною частиною освіти при підготовці математиків, фізиків, хіміків, і тому відповідний курс є складовою навчальних планів цих спеціальностей.

Знання, вміння і навички, отримані студентами в курсі програмування, необхідні їм при засвоєнні учбового матеріалу більшості дисциплін, передбачених навчальними планами різних спеціальностей. Наприклад, для спеціальності "Програмне забезпечення автоматизованих систем" у напрямі підготовки "Комп'ютерні науки" дисципліна "Основи програмування та алгоритмічні мови" є базовою для дисциплін: "Об'єктно-орієнтоване програмування", "Інженерна і комп'ютерна графіка", "Технологія програмування та створення програмних продуктів", "Комп'ютерні мережі", "Чисельні методи в інформатиці", "Системне програмування та операційні системи", "Програмування на Асемблері", "WEB-технології", "Організація системного інтерфейсу", "CASE-технології", "Теорія операційних систем", "Паралельні обчислювальні процеси", "Розподілені системи опрацювання даних", "Проектування комп'ютерних мереж", "Технологія системного програмування", "Технології візуального програмування", "Захист даних". Для спеціальності "Прикладна математика" за напрямом підготовки "Прикладна математика" дисципліна "Програмування" є базовою для дисциплін: "Чисельні методи", "Програмне забезпечення ЕОМ", "Системне програмування", "Бази даних та інформаційні системи", "Чисельні методи математичної фізики", "Комп'ютерна графіка", "Об'єктно-орієнтоване програмування", "Операційні системи", "Комп'ютерні мережі", "Комп'ютерна математика".

Російські навчальні плани спеціальності "Прикладна математика та інформатика" передбачають вивчення дисципліни "Мови програмування і методи трансляції", яка є базовою для наступних дисциплін: "Системне і прикладне програмне забезпечення", "Практикум на ЕОМ", "Чисельні методи", "Теорія ігор і дослідження операцій", "Бази даних і експертні системи" та ін.

Еволюційні і революційні зміни у галузі інформаційних технологій і комп'ютерних наук, а також зміни в педагогіці вищої школи істотно впливають на зміст, засоби, методи і організаційні форми навчання. В роботі [8] викладена наша точка зору з цього питання. Відомий вчений і педагог Д. Гріс стверджує, що "... гарне викладання програмування — важлива частина тих задач, які стоять перед вченими і викладачами інформатики". В [9] відзначено: "Тільки зовсім недавно психологи, що займаються навчанням, і фахівці з машинної математики почали досліджувати специфічні методи навчання програмістів".

Комп'ютерне товариство Інституту інженерів з електротехніки і електроніки (IEEE-CS) і Асоціація з обчислювальної техніки (ACM) спільно розробили "Рекомендації щодо викладання інформатики 2001" (Computing Curricula 2001) [1], в яких наведено набір рекомендацій щодо змісту університетських програм навчальних дисциплін в галузі інформатики. В цьому документі пропонується кілька стратегій складання навчального плану підготовки фахівців в галузі комп'ютерних наук. Зокрема, всі навчальні курси діляться за трьома рівнями: ввідні курси, основні курси і поглиблені курси. Серед них є обов'язкові і факультативні (вибіркові). Курс з програмування тут віднесено до ввідних курсів.

В Росії всі дисципліни ділять на загальні гуманітарні і економічні дисципліни, загальноматематичні і природничонаукові дисципліни, загально професійні дисципліни, дисципліни спеціальності, дисципліни спеціалізацій і факультативи. У перших трьох розділах наявний федеральний компонент, тобто обов'язкові дисципліни, національно-регіональний компонент (або компонент ВНЗ) і курси за вибором студента, які визначаються ВНЗ. Курс з програмування відносять до федерального компоненту загально математичних і природничонаукових дисциплін.

Російські програми з інформатики в основному відповідають рекомендаціям Computing Curricula 2001 [7].

В тимчасових стандартах, що сьогодні діють в Україні, всі навчальні курси ділять на:

- нормативно-навчальні дисципліни, серед яких виділено цикл гуманітарних і соціально-економічних дисциплін, цикл природничонаукової підготовки і цикл професійної і практичної підготовки;
- вибіркові навчальні дисципліни, серед яких виділено цикл дисциплін самостійного вибору навчального закладу і цикл дисциплін самостійного вибору студента;
- дисципліни підготовки спеціаліста (магістра), які включають цикл гуманітарних і соціально-економічних дисциплін та цикл професійно-орієнтованих дисциплін.

Курс програмування включено до циклу дисциплін природничонаукової підготовки розділу нормативних навчальних дисциплін .

Як показав аналіз навчально-методичної літератури, в Україні немає чітко визначених стратегій навчання програмування.

За кордоном, зокрема у США, для ввідного курсу з програмування існує шість стратегій навчання: імперативний підхід, об'єктний підхід, функціональний підхід, з максимальним охопленням матеріалу, алгоритмічний підхід і апаратний підхід. Зупинимося на більш детальному розгляді цих стратегій.

Підхід *з орієнтацією на імперативне програмування* – це найтрадиційніший з усіх перерахованих підходів, який передбачає вивчення однієї з процедурних мов програмування (Pascal, C). Розробники документа Computing Curricula 2001 відзначають, що при цьому підході можна використовувати і одну з об'єктно-орієнтованих мов для навчання основ програмування. Відмінність цього підходу від об'єктно-орієнтованої моделі, полягає в акцентуванні і порядку проходження тем: якщо викладання ведеться з використанням об'єктно-орієнтованої мови, то на першому етапі увага повинна концентруватися на імперативних аспектах цієї мови, а саме виразах, структурах управління, процедурах і функціях, а також інших ключових елементах традиційної процедурної моделі. Технології об'єктно-орієнтованого проектування вивчаються на наступному етапі.

До недоліків імперативного підходу відносять те, що студенти одержать менше досвіду з використання технології об'єктно-орієнтованого програмування, ніж це можливо при використанні моделі "з орієнтацією на ООП". Водночас студентам необхідне розуміння традиційного імперативного (процедурного) стилю програмування, який є невід'ємною частиною об'єктно-орієнтованого програмування. Відзначимо, що для спеціальностей "Програмне забезпечення автоматизованих систем" і "Прикладна математика" навчальними планами передбачений окремий курс з об'єктно-орієнтованого програмування.

Підхід "*з орієнтацією на об'єктно-орієнтоване програмування"* також концентрується на програмуванні, але при цьому із самого початку робиться акцент на принципах об'єктно-орієнтованого проектування і програмування.

При "об'єктному підході" до навчання програмування здійснюється раннє ознайомлення з об'єктно-орієнтованим програмуванням, яке останніми роками стало надзвичайне важливим і для академічного середовища, і для промисловості. Але й об'єктно-орієнтована модель має свої недоліки, зокрема багато мов, які використовуються для об'єктно-орієнтованого програмування (C++, Java, C#), що набагато складніші за класичні процедурні мови. При виборі цієї моделі викладачеві слід ретельно продумати методику вивчення матеріалу, інакше деталі вибраної мови можуть затінити суть об'єктно-орієнтованого програмування.

Підхід "*з орієнтацією на функціональне програмування*" був вперше використаний в Массачусетському технологічному інституті в 80-х роках XX століття. Цей підхід характеризується використанням функціональних мов програмування (Lisp, Scheme). Порівняно з іншими, цей підхід має такі переваги:

- використання мови, що не вивчається в школі, зменшує негативний ефект від різниці в початковій підготовці студентів з інформатики;
- нескладний синтаксис функціональних мов дозволяє викладачеві акцентувати увагу на фундаментальних питаннях програмування;
- використання рекурсії, функцій як даних і зв'язаних структур даних природним чином вписуються в такий підхід, що дає можливість вивчати ці питання на самому початку навчання основ програмування.

Проте і в цьому підході є певні недоліки, зокрема цей підхід, як правило, вимагає від студентів достатньо високого рівня абстрактного мислення на більш ранній стадії навчання у порівнянні з використанням традиційних мов програмування.

Підхід "*з максимальним охопленням"* матеріалу дає студентам більш цілісний погляд на дисципліну, що вивчається, уявлення про цілий ряд важливих тем, замість того, щоб відразу занурювати їх в деталі однієї конкретної проблеми. Студенти, які навчаються за таким підходом, одержують більш широке уявлення про програмування, що дозволяє їм рухатися далі, впевнено і цілеспрямовано, опановуючи інші комп'ютерні курси.

До недоліків такого підходу можна віднести те, що розгляд широкого спектру теоретичних питань з програмування на ранній стадії навчання, який передбачає такий підхід, ускладнює засвоєння навчального матеріалу студентами молодших курсів.

Підхід з "*орієнтацією на алгоритми"* при викладанні основних концепцій програмування передбачає використання псевдокоду замість реальної мови програмування. Цей підхід мінімізує зусилля, що йдуть на вивчення специфічних синтаксичних конструкцій конкретної мови програмування. Натомість від студентів вимагається обґрунтування і роз'яснення алгоритмів, які вони створюють. Це дозволяє студентам працювати з широким діапазоном типів даних і структур управляючої логіки. Після того, як студенти опанують основні типи алгоритмів і типи даних, вони можуть починати використовувати одну з мов програмування. Завдяки виключенню з програми навчання часу, відведеного на синтаксис і деталі певного середовища програмування, **ввідний** курс, який побудований на основі підходу "з орієнтацією на алгоритми", може включати додаткові теоретичні теми. В результаті, студенти починають знайомитися з відповідними аспектами теорії з найперших днів навчання.

Водночас підхід "з орієнтацією на алгоритми" має ряд недоліків: курси, що зводяться до конструювання алгоритмів у псевдокоді, викликають у студентів розчарування і знижують мотивацію навчання; орієнтація на псевдокод не припускає необхідності демонструвати роботу відлагоджених текстів програм, хоча оволодіння навичками налагоджування програм необхідне для подальшої успішної роботи студентів, і тому бажано, щоб вони практикувалися в цьому при навчанні якомога раніше.

Підхід "*з орієнтацією на апаратну частину"* передбачає вивчення основ інформатики і, зокрема, основ програмування, починаючи з машинного рівня. Зазначається, що таким чином студенти можуть вивчити інформатику більш глибоко і послідовно. Тільки після створення у студентів розуміння структурних особливостей апаратної частини, машинної логіки і математики, курс переходить до

розгляду програмування мовами високого рівня. З погляду розробників такий підхід більш придатний для студентів, що вважають за краще розуміти процес обчислення на ЕОМ у всіх його деталях.

Як вважають розробники документа, цей підхід мало ефективний для того, щоб заохочувати студентів бачити цілісні концепції, що стоять за механізмом реалізації. Підхід "з орієнтацією на апаратну частину" також не дуже добре узгоджується з сучасною тенденцією все більшого вдосконалення віртуальних машин, що відділяють процес програмування від апаратних засобів. Цей підхід може бути використаний при підготовці фахівців з проектування комп'ютерної техники, коли необхідно досягнути раннього ознайомлення з апаратною частиною обчислювальної системи.

Незалежно від обраної стратегії навчання існує набір вимог у вигляді списку розділів і тем, які повинен охопити курс з програмування. В документі Computing Curricula 2001 запропоновані варіанти розділів і тем для двохсеместрового і трьохсеместрового курсу з програмування. Вибір того або іншого варіанту залежить від навчальних планів спеціальностей і конкретних вищих навчальних закладів. В обох варіантах передбачаються наступні форми занять – курс лекцій, практичні (лабораторні роботи) з програмування, самостійна робота, що цілком збігається з навчальними планами, передбаченими тимчасовими стандартами в нашій країні.

Програмування – це перша серед професійно-орієнтованих дисциплін, які вивчаються майбутніми програмістами. Від змісту курсу, від методики його навчання, від знань, умінь і навичок, які одержать студенти в цьому курсі, залежить їх подальше навчання, якість засвоєння дисциплін, заснованих на знаннях з програмування, успішність в майбутній професійній діяльності.

Аналізуючи вітчизняні навчальні плани спеціальностей, що готують програмістів, можна зробити висновок, що ці плани, на відміну від Computing Curricula 2001, не передбачають вивчення таких важливих для майбутніх програмістів дисциплін, як:

– "Алгоритми і складність", де вивчаються основи аналізу алгоритмів, методи побудови алгоритмів, класи складності, розподілені алгоритми, основи теорії обчислень;

– "Програмна інженерія", де розглядаються питання проектування програмного забезпечення, процеси розробки програмного забезпечення, специфікації і вимоги до програмного забезпечення.

Крім того, як правило, курси з функціонального і логічного програмування віднесено до дисциплін за вибором, тобто не є обов'язковими. Ці дисципліни входять до циклу дисциплін самостійного вибору навчального закладу. Разом з тим, професійний програміст повинен мати досить повне уявлення про наявні парадигми програмування.

Виходячи з цих міркувань, можна зробити висновок, що при вивченні курсу з основ програмування у студентів повинні формуватися:

1) уявлення про наявні парадигми програмування;
2) уміння аналізувати алгоритми і уміння будувати ефективні алгоритми;
3) навички практичного програмування однією або двома мовами високого рівня;
4) уміння і навички колективного проектування програмного забезпечення;
5) навички самостійної роботи з навчальною літературою при розв'язуванні завдань з програмування.

При складанні робочої програми з програмування для майбутніх програмістів ми, по-перше, орієнтувалися на рекомендації документа Computing Curricula 2001 і, по-друге, виходили з наступних позицій:

- необхідно визначити роль і місце програмування в підготовці фахівця;
- робоча програма повинна бути узгоджена з суміжними дисциплінами;
- робоча програма повинна містити базисні уміння, знання і навички, якими повинні володіти всі студенти;
- не обмежуватися тільки описом розділів курсу, а також містити коментар щодо концептуальної побудови курсу;
- при розробці робочої програми враховувати зміни, які відбуваються в інформаційних технологіях, інноваційні підходи і технології в педагогіці вищої школи, а також все зростаючу важливість навчання протягом всього життя.

Слід відзначити, що швидка еволюція комп'ютерних наук вимагає щорічного перегляду робочих програм.

При побудові методичної системи курсу ми обрали для різних форм занять (лекцій і лабораторних робіт) різні стратегії подання навчального матеріалу.

Розділи і теми лекційного курсу визначалися нами на основі підходу "з максимальним охопленням матеріалу", який починається із загального огляду дисципліни, її місця, ролі у підготовці майбутніх програмістів, і далі зосереджується на алгоритмічних концепціях, а не на нюансах синтаксису конкретної мови програмування.

Ми вважаємо, що лекційний курс повинен включати такі розділи і теми:

Вступ.

Предмет, об’єкт, завдання та зміст курсу “Програмування". Основні етапи розв'язування задач на ЕОМ. Галузі застосування програмування: наукові дослідження, комерційна діяльність, штучний інтелект, системне програмування, мови підготовки сценаріїв, спеціалізовані мови програмування, інформаційні та телекомунікаційні технології. Огляд літератури.

Розділ І. Основи програмування та мови програмування.

Історія розвитку мов програмування. Програмування на апаратному рівні: псевдокоди. Мова Fortran. Функціональне програмування: мова Lisp. Мова ALGOL 60. Комп'ютеризація комерційних записів: мова Cobol. Мова Basic. Мова PL/1. Дві перши динамічні мови: APL и Snobol. Виникнення абстракції даних: мова Simula 67. Ортогональна структура: мова ALGOL 68. Спадкоємці сімейства мов ALGOL. Навмисна простота: мова Pascal. Машинно-незалежна мова: C. Логічне програмування: мова

Prolog. Мова Ada. Об'єктно-орієнтоване програмування: мова Smalltalk. Об'єднання імперативних і об'єктно-орієнтованих властивостей: мова C++. Програмування в World Wide Web: мова Java.

Віртуальні машини. Критерії оцінювання мов програмування: читабельність, легкість створення програм, надійність, вартість. Фактори, що впливають на розробку мови: архітектура комп'ютера, методології програмування. Поняття віртуальної машини. Ієрархія віртуальних машин. Проміжні мови. Проблеми безпеки, пов'язані з виконанням коду на сторонній машині.

Введення в трансляцію. Методи реалізації мов програмування: компіляція, чиста інтерпретація, змішані системи реалізації. Стадії трансляції (лексичний аналіз, синтаксичний аналіз, генерація коду, оптимізація). Машинно-залежні і машинно-незалежні аспекти трансляції. Середовища програмування.

Алгоритми і розв'язування задач. Різни підходи до поняття "алгоритм". Базові структури алгоритмів. Форми подання алгоритмів (текстова, блок-схеми, діаграми Несі-Шнейдермана, таблиці). Роль алгоритмів у процесі розв'язування задач. Поняття про виконавця алгоритмів. Властивості алгоритмів: дискретність, скінченність, визначеність, результативність, масовість. Типи алгоритмів.

Основні конструкції програмування. Вирази та оператор надання значення. Арифметичні вирази: порядок обчислення операторів, порядок обчислення операндів. Перевантажені оператори. Перетворення типів: приведення типів у виразах, явне перетворення типів, помилки у виразах. Вирази-відношення і логічні вирази. Скорочене обчислення. Оператори надання значення: прості надання значення, множинні цільові об'єкти, умовні цільові об'єкти, складені оператори надання значення, унарні оператори надання значення, надання значення як вираз. Змішані надання значення. Складені оператори. Оператори розгалуження: двовариантні оператори розгалуження, конструкції багатоваріантного вибору. Оператори циклу: цикли з лічильником; логічно керовані цикли; цикли з механізмами управління, розміщеними користувачем. Безумовний перехід: проблеми безумовного переходу, види міток, обмеження переходів. Захищені команди.

Змінні і типи даних. Концепція типу даних як множини значень і операцій над ними. Імена, зв'язування, перевірка типів і області видимості. Імена: питання структури, види імен, спеціальні імена. Змінні: ім'я, адреса, тип, значення. Концепція зв'язування: зв'язування атрибутів зі змінними, зв'язування типів, зв'язування змінних з комірками пам'яті і час їхнього життя. Перевірка типів. Строга типізація. Сумісність типів. Область видимості: статична область видимості, блоки, динамічні області видимості. Область видимості змінних і час їхнього життя. Іменовані константи. Ініціалізація змінних. Збирання сміття.

Фундаментальні типи даних. Елементарні типи даних. Числові типи. Логічні типи. Символьні типи. Символьні рядки: питання опрацювання, рядки і дії над ними, варіанти довжини рядків. Реалізація символьних рядків. Порядкові типи, визначені користувачем: перелічувальні типи; інтервальні типи; реалізація порядкових типів, визначених користувачем. Масиви: питання створення, масиви і індекси, сполучення індексів і категорії масивів, кількість індексів масиву, операції над масивами, переріз, реалізація типів масивів. Асоціативні масиви: структура і операції, реалізація асоціативних масивів. Записи: опис записів, посилання на поля запису, операції над записами, реалізація записів. Об'єднання: питання розробки, вільні об'єднання, реалізація об'єднань. Множини: множини в мові Pascal, реалізація множинних типів даних. Файли: файли послідовного доступу, файли прямого доступу, індексно-послідовні файли. Покажчики: питання розробки; операції над покажчиками, проблеми, що виникають при використанні покажчиків; покажчики в мові Pascal, покажчики в мові C і C++, посилання, реалізація посилань і покажчиків. Дерева: основні поняття і визначення. Графи: основні поняття і представлення графів.

Механізми абстракції. Підпрограми: загальні властивості підпрограм, основні визначення, параметри, процедури і функції. Питання створення підпрограм. Середовища локальних посилань. Методи передачі параметрів: семантичні моделі передачі параметрів, моделі реалізації передавання параметрів в основних мовах програмування, перевірка типів параметрів, методи реалізації передавання параметрів, питання розробки. Параметри, що є іменами підпрограм. Перевантажені підпрограми. Роздільна і незалежна компіляція. Питання розробки функцій: побічні ефекти функцій, типи значень, що повертаються. Перевантажені оператори, обумовлені користувачем. Співпрограми.

Рекурсія. Поняття рекурсії і основні означення. Форми рекурсивних процедур: виконання дії на рекурсивному спуску; виконання дії на рекурсивному поверненні; виконання дії як на рекурсивному спуску, так і на поверненні.

Подійно-кероване програмування. Методи опрацювання подій. Поширення подій. Опрацювання подій.

Розділ II. Алгоритми і теорія складності.

Основи аналізу алгоритмів. Асимптотичне означення: Θ-означення; O-, Ω-означення; o- і ω-означення. Стандартні функції і означення: монотонність, цілі наближення знизу і зверху. Оцінка ефективності алгоритму: аналіз часу виконання в середньому, Марківська схема, метод Кірхгофа. Доведення правильності програм. Скінченність алгоритмів. Економія пам'яті.

Алгоритмічні стратегії. Алгоритми повного перебору. Метод "розподіляй та володарюй": загальна характеристика, приклад застосування – множення довгих цілочисельних значень. Метод динамічного програмування: доцільність використання динамічного програмування, приклади застосування – найбільша спільна підпослідовність, задача триангуляції. Метод "жадібних" алгоритмів: теоретичні основи "жадібних" алгоритмів, доцільність використання "жадібних" алгоритмів, приклади застосування – задача про вибір замовлень, коди Хафмена. Метод пошуку з поверненням. Метод гілок та меж: приклад застосування – функція виграшу. Метод локального пошуку: загальна характеристика, локальні і глобальні оптимальні розв'язки, приклад застосування - задача комівояжера. Евристики. Зіставлення зі зразком і алгоритми опрацювання тексту: пошук підрядка в тексті. Алгоритми чисельної апроксимації: розв'язування математичних задач, наприклад – знаходження наближених коренів многочлена.

Фундаментальні обчислювальні алгоритми. Алгоритми послідовного і бінарного пошуку. Квадратичні методи сортування: метод "бульбашки", сортування методом вибору, сортування вставлянням. Алгоритми сортування за час *O(NlogN)*: швидке сортування, пірамідальне сортування, сортування злиттям, сортування Шелла. Сортування вичерпуванням. Сортування підрахунками. Цифрове сортування. Хеш-таблиці: розв'язування колізій за допомогою ланцюгів. Бінарні дерева пошуку: пошук у двійковому дереві, додавання і вилучення елемента. Графи: подання графів – списки суміжності, матриця суміжності; обходи в глибину і ширину; алгоритми пошуку найкоротшого шляху – алгоритми Дейкстри і Флойда; транзитивне замикання – алгоритм Флойда; мінімальне остовне дерево – алгоритми Прима і Крускала. Топологічне сортування.

Розподілені алгоритми. Алгоритми консенсусу і голосування. Розпізнавання завершення. Стійкість до відмовлень. Стабілізація**.**

Класи складності P і NP. Визначення класів P і NP. NP-повнота. Стандартні NP-повні задачі. Методи зведення.

Розділ III. Парадигми програмування.

Мови програмування низького рівня. Мови асемблера. Структура програми. Система команд мікропроцесора. Команди обміну даними. Арифметичні команди.

Структурне програмування. Загальні відомості. Модуль. Загальні характеристики програмного модуля. Розмір модуля. Надійність модуля. Зв'язок з іншими модулями. Методи розробки структури програми. Метод "знизу догори". Метод "згори донизу". Конструювання модуля. Конструктивній підхід. Архітектурний підхід. Розробка структури програм.

Вступ до об'єктно-орієнтованого програмування. Мова C++. Загальні відомості. Основи об'єктного програмування мовою C++. Об'єкти. Успадкування. Екземпляри об'єктних типів. Поля об'єктів. Методи. Код і дані, що суміщені. Визначення методів. Область дії методу. Поля даних об'єкта і формальні параметри методу. Секція private. Інкапсуляція. Сумісність типів об'єктів. Поліморфні об'єкти. Віртуальні методи. Перевірка діапазонів при виклику віртуальних методів. Динамічні об'єкти. Розміщення та ініціалізація за допомогою процедури New. Вилучення динамічних об'єктів. Деструктори.

Вступ до функціонального програмування. Застосування функціональних мов програмування. Математичні функції. Функціональні форми. Перша мова функціонального програмування - Lisp: типи і структури даних, перший інтерпретатор мови Lisp.

Вступ до логічного програмування. Застосування логічного програмування. Елементи числення висловлень і предикатів: висловлення, диз'юнктивні форми, предикати, числення і доведення теорем. Походження мови Prolog. Основні елементи мови Prolog: терми, факти, правила, мета, процес логічного виведення в мові Prolog, проста арифметика, списки.

Вступ до програмування паралельних процесів. Загальні відомості. Переходи за показниками. CRCW- та EREW-алгоритми. Теорема Бронта та ефективність за витратами. Ефективна паралельне опрацювання префіксів. Порушення симетрії.

Вступ до програмування під Internet. Загальні відомості. Характеристика мов: HTML, CGI, JavaScript, Perl, Java, PHP.

Розділ IV. Технологія побудови програмного комплексу.

Технології розробки програмного забезпечення. Життєвий цикл програмного забезпечення. Основні вимоги до програмного забезпечення. Методи проектування. Постановка задачі. Технічне завдання.

Побудова програмної системи, впровадження і супровід. Концептуальна цілісність системи. Створення системи. Інсталяція. Впровадження**.** Супровід. Програмне забезпечення як товар. Право власності та відповідальність за створене ПЗ**.**

Кількість годин, відведених на кожну тему, залежить від загальної кількості лекційних годин, передбачених для даної дисципліни і від спеціальності, на якій вивчається програмування.

Робочі програми з програмування для різних спеціальностей повинні розрізнятися. Наприклад, для спеціальності "Прикладна математика" при розробці робочої програми слід більше часу відвести на вивчення математичного апарату аналізу алгоритмів, методів розробки ефективних алгоритмів. На інженерній спеціальності "Програмне забезпечення автоматизованих систем" більше уваги слід приділити питанням подання даних в пам'яті віртуальної машини і на апаратному рівні. Можливо, окремі теми для деяких спеціальностей можуть бути просто опущені. Наприклад, для спеціальності "Комп'ютерне моделювання процесів і технологій" можна не включати в робочу програму розділ "Технологія побудови програмного комплексу", а також не включати теми: "Вступ до логічного програмування", "Вступ до функціонального програмування", "Вступ до програмування під Internet". Але повинне існувати "ядро" змісту курсу програмування, яке дозволить студентам оцінити об'єм і можливості програмування, забезпечить наявність необхідного фундаменту для вивчення суміжних дисциплін і для професійної діяльності.

Лабораторний цикл занять ми пропонуємо будувати на одному з "програмістських" підходів, а саме, на підході "з орієнтацією на імперативне програмування", в рамках якого студенти вивчають мову програмування Pascal або C, або дві мови програмування: Pascal і C.

Вибір одного з трьох варіантів залежить, по-перше, від спеціальності і, по-друге, від загального рівня підготовки студентів, а точніше, абітурієнтів, тому що програмування вивчається на всіх інженерних спеціальностях на першому курсі. Студенти спеціальностей "Програмне забезпечення автоматизованих систем" і "Прикладна математика" повинні уміти програмувати мовою C і тому для них прийнятний варіант, в якому вивчається мова C або варіант, що передбачає вивчення двох мов: Pascal і C. Знання мови C послужить базою для вивчення дисциплін "Об'єктно-орієнтоване програмування", "Системне програмування", "Операційні системи", з одного боку, і професійні програмісти повинні знати мову системного програмування, якою є C [6], з іншого боку. Для студентів

інших спеціальностей, що вивчають програмування, можливо досить знання однієї мови програмування, якою може бути мова Pascal.

Відзначимо, що побудова циклу лабораторних робіт з вивченням двох мов програмування припустима у двох варіантах. Як показує досвід, залежно від рівня підготовки студентів можливе:

- послідовне вивчення мов Pascal і C;
- паралельне вивчення двох мов програмування Pascal і C з використанням порівняльного підходу [4]. При такому підході використовуються знання мови Pascal, які студенти одержали в шкільному курсі інформатики.

Крім того в циклі лабораторних робіт з курсу програмування можливе ознайомлення студентів з мовами програмування, які належать до інших парадигм, відмінних від імперативної парадигми.

В циклі лабораторних робіт доцільно застосовувати задачі за рівнями складності в межах кожної теми. При цьому доцільно усі задачі поділити на три рівні складності. Задачі першого рівня повинні передбачати засвоєння студентом навчального матеріалу з програмування на базовому задовільному рівні. Задачі другого рівня повинні бути більш трудомісткі, вимагати вдумливості, уміння аналізувати. Нарешті, задачі третього рівня - найскладніші. Наприклад, задачі рівня шкільних олімпіад. Для розв'язування цієї категорії задач від студента потрібна творчість, розвинене алгоритмічне мислення, зацікавленість процесом програмування. Такий підхід забезпечує індивідуальний шлях для кожного студента при навчанні програмування, дозволяє точніше визначити рівень знань кожного студента [3].

При навчанні основ програмування необхідно здійснити перехід від пасивних методів навчання, для яких характерною є ситуація коли студент „знає, але не вміє", до активних методів навчання, які забезпечують досягнення кінцевої мети навчання – розуміння. Тому, окрім вище перелічених форм занять доцільно включити до робочої програми з програмування такий вид навчальної діяльності, як колективний проект [5], що:

1) сприяє осмисленню студентами призначення дисципліни програмування, створює уявлення про майбутню професійну діяльність, сприяє розвитку у студентів таких умінь і навичок:
   - формулювання думок і ідей в зрозумілій формі;
   - здійснення усних презентацій як у формальній, так і в неформальній обстановці;
   - робота зі спеціальною літературою;
   - розуміння відповідальності за виконання своєї частини проекту;
   - розуміння і конструктивне обговорення результатів роботи інших учасників проекту;
   - спілкування з колегами в межах професійної діяльності;
   - складання технічної документації (опис структури проекту, алгоритмів) і посібника для користувача тощо;

2) є складовою частиною підсумкового контролю з дисципліни.

Підводячи підсумки, відзначимо, що при складанні робочої програми з програмування доцільно:

- спиратися на рекомендації документа Computing Curricula 2001;
- визначити інваріантну і варіативну частини курсу, зміст яких залежить від спеціальності;
- запланувати, крім існуючих видів навчальної діяльності, виконання студентами колективного проекту.

Такий підхід дозволить забезпечити високий рівень підготовки майбутніх програмістів для вивчення спеціальних дисциплін в галузі комп'ютерних наук і майбутньої професійної діяльності в цілому.

ЛІТЕРАТУРА:

1. Computing Curricula 2001 http://www.computer.org/education/cc2001
2. Грис Д. Наука программирования. – М.: Мир, 1984. – С.180.
3. Гришко Л.В., Чернявский Н.В. Пути индивидуализации процесса обучения основам программирования // Комп'ютерне моделювання та інформаційні технології в науці, економіці та освіті. V Всеукраїнська науково-практична конференція. Черкаси, 2003. – С.27-29.
4. Гришко Л.В. Порівняльний підхід до навчання основам програмування студентів технічних спеціальностей ВНЗ // Теорія та методика навчання математики, фізики, інформатики. Збірник наукових праць. - Випуск 3. Т. 3. – Кривий Ріг. - 2003. - С.101-106.
5. Гришко Л.В. Коллективный проект как практический прием обучения будущих программистов // Информационные технологии в учебном процессе. Четвертый международный научно-методический семинар. –Одесса, 2003. – С.189-191.
6. Керниган Б., Ритчи Д. Язык программирования С.\Пер. с англ., 3 изд., испр. – СПБ.: "Невский диалект", 2001. - 352с.
7. Одинцов И.О. Профессиональное программирование. Системный подход. – СПб.: БХВ·Петербург, 2002. - 512с.
8. Триус Ю.В., Богатирьов О.О., Гришко Л.В. Особливості створення методичної системи навчання основам програмування для підготовки майбутніх інженерів-програмістів // Вісник Черкаського університету, серія "Педагогічні науки". Випуск 35. – Черкаси, 2002. – С.135-141.
9. Шнейдерман Б. Психология программирования: Человеческие факторы в вычислительных и информационных системах. Пер. с англ.– М.: Радио и связь, 1984.– 304 с.