3. Самостійно пізнавати одночасно з конструкцією електричного апарата про матеріали, з яких зроблені окремі елементи та деталі апаратів.

4. У процесі навчання здійснювати перехід від реального фізичного уявлення тих або інших елементів електричного апарата до його креслярських аналогів (тобто умовних графічних позначень).

Це дозволяє в одному модульному елементі дати учням більш повну картину і здійснити диференціацію навчання про пристрої, конструкцію, матеріали та інші технічні дані електричного апарата.

БІБЛІОГРАФІЯ

1. Анисимов Н. В. Теоретические основы построения моделей электрорадиотехнических профессий в системе ПТО: [монография] / Н. В. Анисимов. – Кировоград: Издательство ГЛАУ, 2005. – 448 с.: ил.

2. Анисимов Н.В. Системный подход при построении профессиональных моделей. Освітянські обрііï. Збірник наукових праць КДТУ. Сер. пед. науки. – 2007. – Вип. 72. – С. 15-20.

3. Анисимов Н. В. Демонстрационное объемно-модульное устройство // Профессионально-техническое образование. – 1988. – № 11. – С. 49…51.

4. Педагогические требования к лабораторным занятиям в профтехучилищах: [монография] / Н. В. Анисимов. – Кировоград: Издательский центр АНПР. – 1999. – 128 с.

5. Модульная система обучения профессии. 4.1. Общие вопросы / сост. Г. П. Матвеев и др. Донецк, 1992. – С. 4.

ВІДОМОСТІ ПРО АВТОРА

Анісімов Микола Вікторович – кандидат педагогічних наук, доцент кафедри ЗТД та методики трудового навчання КДПУ ім. В.Винниченка.

Наукові інтереси: прогнозування змісту професійної освіти та моделювання електронних підручників.

ПОШУК ШЛЯХІВ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ВИВЧЕННЯ МОВИ АСЕМБЛЕРА

Олександр БАРАНЮК

У статті дається детальний аналіз проблеми вивчення мови асемблера і здійснюється пошук шляхів підвищення ефективності засвоєння цієї мови студентами вищих навчальних закладів.

The article provides a detailed analysis of the assembler language learning problem and the search for ways to improve the language learning by students in higher education.

За час навчання студенти напрямів «Прикладна матемптика» та «Комп'ютерні науки» вивчають кілька мов програмування, серед яких переважну більшість складають мови високого рівня, що надають наступні переваги [6]: Програми на мовах високого рівня коротші, швидше розробляються і відлагоджуються; ці програми на мовах високого рівня легші для розуміння і підтримки; вони портативні (легко переносяться на різні платформи).

Крім цього, студенти комп'ютерних напрямів підготовки вивчають і мови низького рівня, до яких належить асемблер. Програмування мовою асемблера значно складніше, вимагає знання апаратних засобів комп'ютера, більш глибокої підготовки, терпіння, наполегливості, здатності до абстрактного мислення. Хоча сфера застосування асемблера сьогодні звужується, ця мова буде існувати до тих пір, поки будуть існувати процесори. Є принаймні дві причини, завдяки яким будуть продовжувати створювати програми мовою асемблера.

По-перше, мовою асемблера можна створити ефективніший код. Під

ефективністю мають на увазі здатність програми відповідати поставленій меті. Тут виділяють два основних аспекти: просторову і часову ефективність. Під просторовою ефективністю розуміють здатність програми займати якомога менше пам'яті, а під часовою – здатність виконуватися за найменший час.

По-друге, мова асемблера дає прямий контроль над апаратними засобами комп'ютера.

Інколи враховують і третій аспект – швидкість розробки програми. За цим критерієм, асемблерні програми, звичайно, програють програмам на мовах високого рівня, хоча і в цьому напрямі зроблено значні кроки. Мається на увазі, що останнім часом з'явилося багато різноманітних інтегрованих середовищ розробки, у тому числі й візуальних.

Один із визнаних фахівців у галузі викладання асемблера, автор високорівневої мови програмування асемблера HLA Рендалл Хайд, говорить, що навіть якщо студент не збирається програмувати мовою асемблера в майбутньому, її вивчення дає гарне розуміння комп'ютерної системи в цілому і можливість створювати більш ефективні програми мовами високого рівня [7].

Проблемам навчання мовам програмування високого рівня присвячено досить багато досліджень вітчизняних та зарубіжних вчених, асемблера – значно менше.

Проблеми вивчення асемблера. Програмування це складний предмет, який потребує значних зусиль і від студента, і від викладача.

Відомо, що багато студентів відчувають труднощі при вивченні мов програмування. Перш ніж визначити шляхи полегшення і поліпшення вивчення мов програмування, потрібно зрозуміти, з якими саме труднощами стикаються студенти при вивченні мов програмування. У своєму дослідженні з цієї проблеми А. Гомес і А. Мендес [8] виділяють такі складові труднощів навчання мов програмування.

Методи навчання. Навчання не персоналізоване. Викладацькі стратегії не підтримують усі стилі навчання студентів. Навчання динамічним концепціям відбувається через статичні матеріали. Вчителі більше зосереджені на викладанні мови програмування і її синтаксичних деталях замість того, щоб сприяти розв'язанню задач з використанням мови програмування.

Студенти використовують неправильні методології вивчення, не працюють достатньо наполегливо аби досягнути компетенцій програмування.

Студентські здібності і ставлення. Студенти не знають як розв'язувати задачі, не мають достатніх математичних і логічних знань, їм бракує спеціальних знань з програмування.

Суть програмування. Програмування вимагає високого рівня абстракції. Мови програмування мають дуже складний синтаксис.

Психологічні чинники. Студенти не мають мотивації. Студентам доводиться вивчати програмування в досить складний період їхнього життя.

Початківці відчувають брак спеціальних знань та фахових навичок, і ця точка зору пронизує значну частину літератури з проблеми. Серед основних проблем початківців у програмуванні Леон Вінслоу [9] виділяє: відсутність адекватної ментальної моделі предметної області, обмеженість поверхневими знаннями предмету, неміцність знань і нехтування стратегією, використання загальних стратегій розв'язку задач, а не стратегій розв'язання конкретного завдання, схильність до підходу «програмування через керуючі структури», і використання підходу

знизу вгору (рядок за рядком) до розв'язання задач.

На відміну від експертів, новачки витрачають дуже мало часу на аналіз задачі та планування, проектування, тестування коду, і тяжіють до спроб невеликих локальних виправлень, а не глибинного перегляду програми. Це підхід під умовною назвою «кодуємо і виправляємо» (Code-and-Fix), який насправді можна охарактеризувати як відсутність будь-якого підходу. Новачки зазвичай досить слабкі у відстеженні коду, мають погане розуміння основного потоку виконання програми, часто забувають, що кожний поточний стан програми створений її попереднім станом.

Звичайний підхід до навчання програмуванню передбачає спочатку навчити основам мови програмування [5], а потім провести студентів через ефективні стратегії процесу програмування в цілому. Тому повинно переважати вивчення основних концепцій, що створює основу для розбудови більш просунутих навичок.

Рон Портер [10, с. 125] відзначає, що більшість проблем з програмуванням пов'язані скоріше із загальними аспектами розв'язування задач, ніж із деталями мови програмування. Це означає, що перш, ніж писати програму, яка розв'язує задачу, виражену звичайною мовою, потрібно розв'язати її в загальних термінах. Повинен існувати етап, який передбачає розуміння задачі та її розв'язку в концептуальному плані, що передує її розв'язуванню в термінах програмування. Етап програмування, в самому вузькому сенсі, це просто процес перекладу концептуального рішення в послідовність конструкцій мови програмування і на цьому рівні мало або взагалі немає творчості. Специфікація проблеми, для якої потрібне рішення у вигляді комп'ютерної програми передбачає звичайний мовний опис. Тому найбільшою і найскладнішою частиною завдання програмування є отримання необхідного концептуального розуміння із специфікації природної мови для одержання рішення на концептуальному рівні, а не на рівні коду.

Проведений аналіз показує, що існує ряд досліджень проблеми навчання програмування. Але більшість із них присвячено розгляду питання навчання мов програмування високого рівня, переважно в межах вступного курсу (т. зв. Introductory Course або CS1). Дуже мало досліджень присвячено навчанню мови програмування асемблера. Вважається, що асемблер – мова програмування для професіоналів і вони здатні успішно її опанувати.

Завдання ускладнюється тим, що в сучасних навчальних планах на вивчення асемблера як мови програмування відводиться все менше годин, часто асемблер переводиться в розряд дисциплін за вибором або й зовсім вилучається з навчальних планів як окрема дисципліна. У багатьох навчальних закладах вивчення мови асемблера поєднується в одному курсі разом з архітектурою ЕОМ [4]. І цьому, звичайно, є пояснення. Асемблер як мова програмування вивчається вже приблизно чотири десятиліття. За цей час з'явилося чимало нових цікавих і змістовних курсів, які можна запропонувати студентам. Це пов'язано із широким розповсюдженням комп'ютерних мереж, Інтернет-технологій, мультимедіа-технологій, баз даних та ін. Проте повсюдне використання мікропроцесорних засобів керування і обчислювальних систем вимагає кваліфікованих фахівців, здатних програмувати мовою асемблера.

Метою даної статі є пошук основних напрямів і визначення

методичних підходів, здатних підвищити ефективність вивчення мови асемблера студентами вищих навчальних закладів.

Попередній аналіз проблем, з якими стикаються студенти при вивченні мов програмування [5, 8, 9] дає підстави зосередити свої пошуки в таких напрямах: організація навчальних курсів в цілому та окремих занять курсу, розвиток навичок розв'язування задач, використання структурного підходу до програмування та шаблонів, використання ефективних навчальних середовищ розробки програм.

Організація занять. Більшість дисциплін, пов'язаних із вивченням мов програмування, традиційно включають лекційні заняття, лабораторні заняття і самостійну роботу студентів. Зрозуміло, що всі ці види навчальної діяльності існують не відокремлено, а складають єдиний комплекс, підпорядкованій загальній меті викладання дисципліни.

При розробці інформаційних систем часто використовують так званий уніфікований процес, в основу якого покладено кілька важливих ідей, зокрема ітеративна розробка. Згідно з цим підходом вся розробка складається з кількох окремих ітерацій, кожна з яких включає свої етапи аналізу вимог, проектування, реалізації, тестування і закінчується створенням працездатної системи з певним набором функціональності [1]. Ітеративний цикл базується на постійному розширенні і доповненні системи з додаванням нових модулів до існуючого ядра. Система поступово розростається і крок за кроком наближається до заданої замовником системи. Такий підхід називають ітеративною інкрементною розробкою. У процесі аналізу вимог до кожної нової інформаційної системи розробники поступово проходять шлях від першого знайомства з предметною областю до повного знання про систему, яка створюється.

Аналогічний підхід можна запровадити і при вивченні мов програмування. Весь процес вивчення курсу слід представити не у вигляді однієї або кількох прямих ліній (за видами робіт), а у вигляді послідовних витків спіралі зі зворотними зв'язками. Кожний виток повинен включати теоретичне вивчення теми на лекції, виконання лабораторної роботи, самостійну роботу з теми і завершуватися створенням невеличкої, але обов'язково працюючої програми, в якій реалізовано одне із ключових понять даної теми. З кожним новим витком знання студента поповнюються, розширюються, у нього формуються навички створення все більш складних програм. Дослідження показують, що весь курс при інкрементному вивченні розбивається на 8–15 тем (підмножин мови) [6].

Для забезпечення інкрементного підходу при вивченні програмування мовою асемблера основними кроками на кожному витку спіралі повинні бути такі:

теоретичне вивчення одного невеликого питання в кожній темі;

здобуття навичок використання нового матеріалу для розширення функціональності, підвищення ефективності, оптимізації коду програм, створених при вивченні попередніх питань;

удосконалення загальних навичок розв'язування задач із використанням нових понять, структур, можливостей, засобів.

Наведемо приклади розгортання понять, які розглядаються на різних етапах процесу вивчення асемблера.

Поняття підпрограми. Перше знайомство з підпрограмами відбувається вже в найпростішій асемблерній програмі для виведення на екран тексту типу "Hello, World!" за

допомогою однієї з функцій DOS. Далі вивчаються команди виклику підпрограм і повернення з них (CALL і RET для 80x86), команди роботи зі стеком, потім розглядаються способи передачі параметрів у підпрограми. Нові знання про підпрограми додаються при вивченні домовленостей про виклики підпрограм та питань взаємодії асемблерних модулів з модулями, написаними на мовах високого рівня у мультимодульних проектах.

Поняття адресації даних. Поняття адресації даних розглядається при вивченні типів даних, способів адресації операндів у командах, засобів роботи з масивами та структурами, способів передачі параметрів у підпрограми та команд для роботи з ланцюжками даних.

Поняття операцій введення-виведення. Відомо, що асемблер не має власних засобів введення-виведення, оскільки він апаратно залежний. Але без операцій введення-виведення не обходиться практично жодна програма. Тому асемблер використовує засоби взаємодії з користувачем, які має BIOS або операційна система. Вивчення цих засобів починається з найпростішої програми на асемблері, продовжується в темах, присвячених введенню-виведенню засобами BIOS та DOS, при вивченні стандартних бібліотек, файлових операцій і завершується при вивченні консольних та віконних програм, які використовують для введення-виведення системні виклики операційної системи.

Наведені приклади показують, що вивчення багатьох питань програмування мовою асемблера відбувається послідовно, в кілька етапів, причому на кожному новому етапі рівень вивчення питання поглиблюється, здобуті навички програмування стають все ширшими, а програми – складнішими. Отже, можна організувати вивчення матеріалу курсу на основі інкрементного підходу.

Розв'язування задач. Процес розв'язування задач програмування можна умовно поділити на кілька частин [9]:

1) розуміння проблеми;

2) визначення способу розв'язання задачі;

3) специфікація алгоритму розв'язання задачі;

4) переведення розв'язку на відповідну мову програмування;

5) тестування і відлагодження програми.

Початкові етапи розв'язування задач є найбільш важливими. Розуміння проблеми починається із детального вивчення предметної області і побудови моделей. Цей етап вимагає від студента, перш за все, уважного вивчення і глибокого розуміння умови задачі. Наступним етапом є визначення способу розв'язання задачі. Задача цього етапу полягає в тому, щоб переконатися, що задачу взагалі можливо розв'язати і яким чином це можна зробити. Для того, щоб перевірити, чи спосіб розв'язання задачі знайдено, потрібно дати відповідь на питання: «Чи готовий я розв'язати задачу вручну, скільки б часу на це не знадобилося?» Як тільки наступає розуміння алгоритму розв'язання задачі, можна приступати до його формалізації. Є багато способів формалізації алгоритмів. Найчастіше для цього використовують алгоритмічні мови, блок-схеми, діаграми діяльності UML, або навіть детальний словесний опис алгоритму. Слід віддавати перевагу таким способам специфікації алгоритму як алгоритмічна мова або блок-схема, тому що вони містять основні керуючі структури, які використовуються і в мовах програмування. Це значить, що пізніше значно полегшується переведення алгоритму на конкретну мову

програмування. Алгоритмічні мови цінні також тим, що їх синтаксис вільний, тому кожний може підлаштувати засоби алгоритмічної мови під свої потреби або навіть використовувати свою власну алгоритмічну мову.

Коли алгоритм розв'язання задачі розроблений, останні два етапи розв'язування задачі являють собою скоріше технічні моменти і, зазвичай, не складають проблеми. Речення псевдокоду алгоритмічної мови можуть бути використані як коментарі в майбутній асемблерній програмі [9]. Достатньо лише попереду кожного такого коментаря поставити команду (команди) конкретного процесора.

Слід звернути увагу, що алгоритм, за яким передбачається створення програми мовою асемблера, повинен бути більш детальним у порівнянні з мовами високого рівня і бути орієнтованим на використання типових команд процесора. Саме тому при вивченні питань алгоритмізації часто використовують поняття виконавця алгоритму [2]. При програмуванні мовою асемблера виконавцем алгоритму виступає процесор.

На етапі алгоритмізації часто використовують шаблони. Це пов'язано з тим, що комп'ютерні програми у своїй більшості виконують певні стандартні процедури. Наприклад переважна більшість програм, які створюють студенти під час вивчення мов програмування використовують шаблон типу: введення-обробка-виведення (Input-Process-Output або IPO) [9].

Структурне програмування. Концепція структурного програмування передбачає наявність кількох керуючих структур, з яких може бути побудована будь-яка програма [3]. При цьому забезпечується ясність програми і простота керування ходом її виконання. Кожна структура має одну точку входу і одну точку виходу.

Всього існує чотири основні керуючі структури: слідування, вибір, повторення, розгалуження. Слідування або лінійна структура забезпечується автоматично, оскільки кожний рядок програми на асемблері перетворюється компілятором на відповідний машинний код команди, а процесор послідовно виконує команди, представлені машинними кодами.

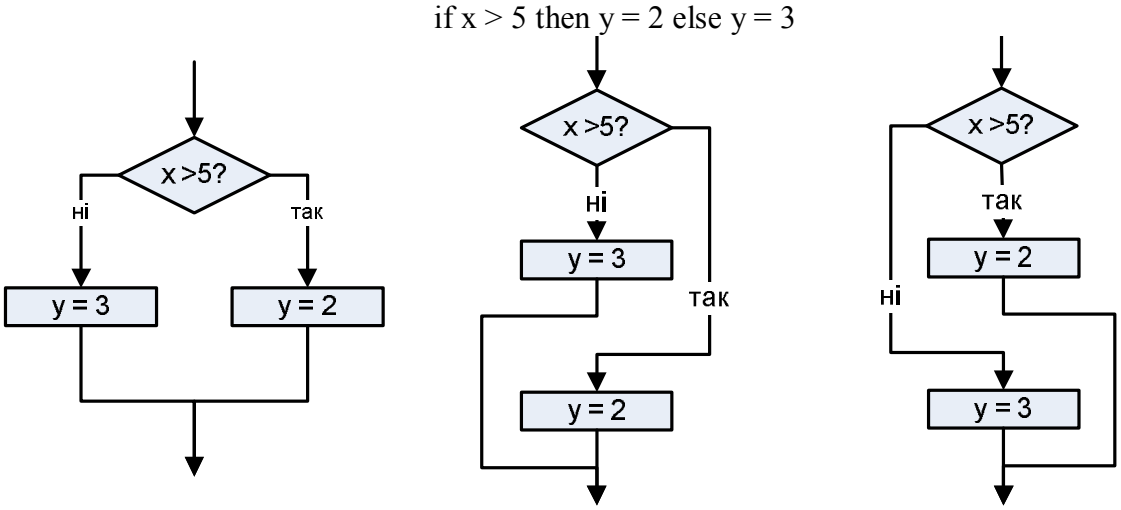
Концепція структурного програмування передбачає мінімальне використання структури розгалуження. В сучасних мовах високого рівня це виправдано і більшість програм не використовують оператор "goto" в явному вигляді. Проте для програм, написаних мовою асемблера, виконавцем є процесор, який має команду безумовного переходу (JMP для 80x86) – аналог "goto". При реалізації деяких керуючих структур без цієї команди обійтися неможливо.

Усі інші структури (крім слідування) основані на використанні команд умовних переходів. Очевидно, що всі структури будь-якої мови високого рівня можуть бути реалізовані мовою низького рівня, оскільки кожна програма з мови високого рівня переводиться компілятором у машинні коди, а програма мовою асемблера складається з мнемокодів (умовних позначень) машинних команд. Причому, мова асемблера має багато варіантів команд умовних переходів, чого достатньо для ефективної реалізації будь-яких керуючих структур, аналогічних мовам високого рівня.

На нашу думку, під час програмування мовою асемблера слід вести мову не про категоричну відмову від оператора goto, а про те, що використання структурного програмування дає більш чітку, прозору і надійну програму, що значно

полегшує її розуміння, розробку і відлагодження. Компроміс полягає в тому, щоб використовувати структурне програмування на етапі розробки програм з наступною заміною типових керуючих структур асемблерним кодом на основі численних шаблонів на етапі кодування.

Розглянемо для прикладу реалізацію мовою асемблера 80x86 структури вибору.



а) б) в)

Рис. 1. Варіанти реалізації структури вибору.

На відміну від програмування мовами високого рівня, які передбачають вищий рівень абстракцій, структура, представлена на рис. 1, а) не може бути безпосередньо реалізована мовою асемблера, оскільки різні ділянки програмного коду (y=3 та y=2) не можуть бути розташовані у пам'яті паралельно, а лише послідовно. На практиці це означає, що дана структура може бути реалізована у вигляді варіантів 1, б) або 1, в). Відомо [3], що кожна керуюча структура повинна мати один вхід і один вихід. У програмі на асемблері це означає, що звести виходи двох ділянок коду в одну точку можна лише за допомогою команди безумовного переходу (наприклад, із виходу блоку y=3 на вихід блоку y=2 на рис. 1, б)). Приклади реалізації структур 1, б) та 1, в) мовою асемблера 80x86 показані на рис. 2, а) та 2, б) відповідно. Кожна із таких програм використовує одну команду умовного переходу (JA або JNA) та одну команду безумовного переходу JMP.

На нашу думку варто виділити ще один напрям, який необхідно враховувати з метою підвищення ефективності вивчення мов асемблера студентами. Цей напрям стосується організації і використання середовищ розробки програм.

Середовища розробки програм. Мінімальний набір засобів для створення програм на асемблері включає текстовий редактор, асемблер і компонувальник, а також відлагоджувальник. Зазвичай при вивченні мов програмування студенти використовують повнофункціональні професійні інтегровані середовища розробки (ICP) програм (Integrated Development Environment або IDE), які мають дуже багатий інструментарій і розвинену функціональність. Такі середовища призначені для професійної розробки програм, а не для вивчення мови програмування. Це призводить до

того, що студенти-початківці змушені опановувати навички роботи в середовищі замість того, щоб зосередитися на вивченні конструкцій мови програмування.

```
а)      cmp ax, 5       ; x > 5?
        ja thenbl       ; так, перейти на блок THEN
        mov y, 3        ; ні, виконати блок ELSE
        jmp endif       ; вийти із структури thenbl:
        mov y, 2        ; виконати блок THEN
       endif:           ; кінець структури
б)      cmp ax, 5       ; x > 5?
        jna elsebl; ні, перейти на блок ELSE
        mov y, 2        ; так, виконати блок THEN
        jmp endif       ; вийти із структури elsebl:
        mov y, 3        ; виконати блок ELSE
       endif:           ; кінець структури
```

Рис. 2. Приклади реалізації керуючих структур вибору мовою асемблера.

Прикладами середовищ, призначених для програмування з використанням асемблера, можуть бути WinAsm Studio, RadAsm, Chromatic IDE, ASM Editor for Windows, NASMEdit IDE та ін. До інструментів, необхідних студентам-початківцям можна віднести: двійково-десятково-шістнадцятковий калькулятор, калькулятор команд процесора, перевірка синтаксису під час набору, довідкова система з мови асемблера та команд процесора, бібліотека скелетів програм та шаблонів програмних фрагментів, компіляція-компонування-запуск програми, візуалізація ходу виконання програми, інформативний синтаксис повідомлень про помилки компіляції та деякі інші.

Висновки. Аналіз труднощів, з якими мають справу студенти, свідчить, що підвищення ефективності засвоєння мови асемблера студентами закладів можна досягти шляхом раціональної організації навчальних занять із використанням інкрементного ітеративного підходу; розробки навчальних завдань, що мають на меті не лише опанування синтаксисом і конструкціями асемблера, а, насамперед, розвиток загальних навичок розв'язування задач і алгоритмізації; використання при розробці асемблерних програм студентами структурного програмування із залученням засобів формалізації алгоритмів у вигляді блок схем, діаграм діяльності UML та алгоритмічної мови, широкого застосування шаблонів проектування та програмування; розробки навчальних інтегрованих середовищ розробки програм, спеціально призначених для вивчення мови асемблера, які мають засоби полегшення створення та відлагодження програм на початкових етапах навчання.

БІБЛІОГРАФІЯ

1. Ларман К. Применение UML 2.0 и шаблонов проектирования : пер. с англ. / К. Ларман. – 3-е изд. – М. : ООО «И.Д. Вильямс», 2007. – 736 с.

2. Малеев В.В. Общая методика преподавания информатики : учеб. пособ. / В. Малеев. – Воронеж, ВГПУ, 2005. – 271 с.

3. Структурное программирование = Structured Programming / У. Дал, Э. Дейкстра, К. Хоор – М.: Мир, 1975. – 245 с.

4. Agarwal K.K. Do we need a separate assembly language programming course? / K. Agarwal, A. Agarwal // Journal of Computing Sciences in Colleges. – 2004, № 19 (4). – с. 246–251.

5. Ala-Mutka K. Problems in Learning and Teaching Programming – A literature Study for Developing Visualizations in the Codewitz-Minerva Project / K. Ala-Mutka // Codewitz Needs Analysis. – 2004.

6. Brusilovsky P. Teaching programming to novices : a review of approaches and tools / P. Brusilovsky, A. Kouchnirenko, P. Miller, I. Tomek // Educational Multimedia and Hypermedia. – Vancouver : AACE, 1994. – pp. 103–110.

7. Dandamudi S.P. Introduction to Assembly language programming : Pentium and RISC processors / S.P. Dandamudi. – 2nd ed. – Springer, 2004. – 690 p.

8. Gomes, A. Learning to Program – Difficulties and Solutions / A. Gomes, A.J. Mendes // International Conference on Engineering Education. – ICEE, 2007. pp. 283–287.

9. MacKenzie S. A Structured Approach to Assembly Language Programming / S. MacKenzie. – IEEE Transactions on Education. – 1988. – Vol. 31. – No. 2. – pp. 123–128.

10. Porter R. Design Patterns in Learning to Program : A thesis … for the degree of Doctor of Philosophy / R. Porter. – Adelaide, 2006.

ВІДОМОСТІ ПРО АВТОРА

Баранюк Олександр Филимонович – старший викладач кафедри інформатики КДПУ ім. В.Винниченка, кандидат технічних наук.

Наукові інтереси: системи автоматизації виробництва, проблеми викладання комп'ютерних наук у вищій школі.

ІНТЕГРАЦІЯ ПРИРОДНИЧИХ ЗНАНЬ УЧНІВ У ПРОЦЕСІ ВИВЧЕННЯ МОЛЕКУЛЯРНОЇ ФІЗИКИ В ЗАГАЛЬНООСВІТНІЙ ШКОЛІ

Вікторія БУЗЬКО

У статті розглянуто аспекти інтеграції знань фізики, хімії, біології під час вивчення молекулярної фізики.

Aspects of integrations of the knowledge in Physics, Chemistry's, Biology's while studying Molecular Physics are considered in the article.

Актуальність. В наш час відбуваються зміни у системі освіти. Однією з актуальних проблем є проблема інтеграції навчання в школі. Удосконалення навчально - виховного процесу школи вимагає здійснення мети всебічного розвитку особистості на основі комбінації найрізноманітніших видів діяльності учнів. Виникає необхідність у тому, щоб пізнання, праця, гра, спілкування в навчальному процесі своєю єдністю сприяли б формуванню активної, творчої особистості.

Інтеграція як явище відзначена у фундаментальних і прикладних її галузях. Вона виникла на тлі свої протилежності − диференціації наук і її галузей, зростаючому об'ємі знань і вимог до них у кожній галузі. Поглиблення процесу диференціації наук є однією із причин, що ведуть до протилежного ефекту − прагненню до цілісності, інтеграції знань із різних областей. Сучасний підхід до навчання важко уявити без здійснення міжпредметних зв'язків з іншими начальними предметами.

Використання міжпредметних зв'язків активізує освітній процес, стимулює пізнавальний інтерес учнів, сприяє розширенню світогляду. Так уміло розкриті й показані зв'язки фізики з біологією, хімією підсилюють практичну спрямованість фізики.

Тобто, інтеграція має на меті закласти основи цілісного уявлення про природу і суспільство, та сформувати власне відношення до законів їхнього розвитку.

Проблема реалізації міжпредметних зв'язків в навчально - виховному процесі розглядалася в працях таких видатних учених-педагогів, як Я. А. Коменський, К. Д. Ушинський та інших. До класиків, що активно працювали над впровадженням міжпредметних зв'язків слід віднести