2. Бех І. Д. Концептуальні засади розвитку виховання в системі освіти України // Збірник наукових праць. Педагогічні науки. – Херсон: Видавництво ХДУ, 2003. – Вип. 34. – С. 170–174.

3. Грязнов Ю. П., Сергєєв О. В. Дидактичні принципи формування професійної компетентності спеціаліста у процесі навчання фізики на модульній основі // Удосконалення навчання фізики у вищій школі в умовах ступеневої освіти : Матеріали III Всеукр. наук. конф. "Фундаментальна та професійна підготовка фахівців з фізики". – К.: НПУ ім. М.П. Драгоманова, 1998. – Ч. І. – С.72–76.

4. Диденко А. В. Педагогические условия профессионального самосовершенствования будущих офицеров : автореф. дис. на прис. науч. ст. канд. пед. наук : 13.00.04 «Профессиональное образование» / Диденко Александр Васильевич.– Хмельницкий : НАГПСУ им. Б. Хмельницкого, 2003. – 18 с.

5. Дж. Равен. Компетентность в современном обществе: выявление, развитие и реализация / Джон Равен. – М. : Когито-Центр, 2002. – 98 с.

6. Концепція військової освіти в Україні. Постанова Кабінету міністрів України від 15 грудня 1997 р., №1410. – 20 с.

7. Лаврентьев Г.В. Інноваційні навчальні технології у професійній підготовці фахівців / Г.В. Лаврентьев, Н.Б. Лаврентьева. – Барнаул : Вид-во Алтайського держ. ун-ту, 2002. – 423 с.

8. Челпанов О.С. Суб'єктивно-діяльнісний підхід до підготовки військових фахівців у вищих військових навчальних закладах / О.С. Челпанов, С.В. Залкін. – Х. : ХВУ, 1998. – 40 с.

ВІДОМОСТІ ПРО АВТОРІВ

Аврамчук Олена Євгеніївна - кандидат педагогічних наук, викладач фізики, Житомирський військовий інститут ім. С.П. Корольова.

Коло наукових інтересів: професійна підготовка курсантів.

Пасько Ольга Олександрівна - кандидат педагогічних наук, викладач фізики, Сумський державний педагогічний університет.

Коло наукових інтересів: мультимедійні засоби в підготовці майбутніх фахівців.

РОЗРОБКА НАВЧАЛЬНОЇ БІБЛІОТЕКИ ПІДПРОГРАМ ДЛЯ НИЗЬКОРІВНЕВОГО ПРОГРАМУВАННЯ

Олександр БАРАНЮК

У статті подається аналіз проблеми підвищення ефективності навчання низькорівневому програмуванню в умовах дефіциту навчальних годин і пропонується розробка навчальної бібліотеки підпрограм введення-виведення для полегшення навчання початківців.

The article presents an analysis of the problem of increasing low-level programming teaching effectiveness under conditions of training time restrictions by means of using the library of input/output subroutines facilitating teaching novices.

Постановка проблеми. Галузь комп'ютерних наук досить динамічна, постійно з'являються нові мережеві, інформаційні та мультимедійні технології, зростає обсяг знань, зазнають змін відповідні дисципліни в університетах. Це об'єктивна загальносвітова тенденція. Аналіз рекомендацій по викладанню інформатики в університетах (Computer Science Curricula), які регулярно розробляються Асоціацією обчислювальної техніки (ACM) та Інститутом інженерів електротехніки та електроніки (IEEE) показує, що кількість областей знань інформатики, які пропонуються до вивчення зросла з 14 до 18 лише за роки нинішнього століття. Знайти час на вивчення нових дисциплін в межах встановленого навчального часу стає все важче. Доводиться зменшувати кількість годин на вивчення старих дисциплін або певних тем, об'єднувати окремі дисципліни в одну або й зовсім вилучати їх з навчальних планів.

Серед фахівців у галузі комп'ютерних наук триває дискусія з приводу змісту та обсягу дисциплін, присвячених вивченню архітектури комп'ютера та основ низькорівневого програмування [3]. Вітчизняний галузевий стандарт з напряму підготовки «Інформатика» передбачає вивчення мови асемблера лише в складі дисципліни «Архітектура обчислювальних систем».

Разом з тим перелік питань, якими повинен оволодіти студент залишається досить широким. Освітньо-професійна програма підготовки бакалавра передбачає володіння основами програмування мовою асемблера на рівні знання системи команд, операцій введення-виведення, роботи з пам'яттю, низькорівневої реалізації високорівневих керуючих структур та навичок трансляції програм у машинні коди.

Зважаючи на ущільнення навчальних програм, формуванню навичок низькорівневого програмування вдається виділити лише кілька лекцій і кілька лабораторних занять. Проблема полягає в тому, аби за рахунок сучасних педагогічних прийомів студент зміг оволодіти основами асемблерного програмування в короткий термін. Один із можливих підходів до вирішення цієї проблеми – використання готових рішень на всіх етапах створення комп'ютерної програми у вигляді шаблонів та бібліотечних підпрограм, що може дати реальну економію часу і зусиль при написанні програм та прискорити одержання працездатного коду.

Аналіз останніх досліджень і публікацій. Низькорівневе програмування нелегко дається новачкам, а становлення професіонала в цій галузі триває роками. Психологи вважають, що перетворити новачка на експерта в галузі програмування за чотири роки неможливо, проте можливо зробити його компетентним [9]. Труднощі опанування асемблером пов'язані не стільки зі складністю чи незвичністю його синтаксису, скільки з відсутністю загальних навичок розв'язування задач. Новачки витрачають мало часу на аналіз задачі, планування та конструювання програми. Основним підходом залишається створення програми з нуля методом спроб і помилок (code-and-fix), який скоріше означає відсутність будь-якого підходу. Студентам бракує належного алгоритмічного мислення, вони досить слабкі у відстеженні власного коду і погано розуміють основний потік виконання програми [5]. Хоча не завжди у цьому винні студенти.

Традиційний підхід до навчання програмуванню, більшість підручників та посібників передбачають послідовне вивчення тем, присвячених деталям синтаксису та основним операторам певної мови програмування, ілюструючи їх прикладами використання. Виклад матеріалу здійснюється за принципом «від простого до складного». Зазвичай починають з простих операторів, переходять до логічних виразів і умовних операторів, процедур, закінчуючи масивами і структурами [8]. Складні питання вивчаються в кінці курсу, коли часу на їх ґрунтовне засвоєння і закріплення обмаль. Занадто багато уваги приділяється механізмам реалізації, які спантеличують студентів. Цей підхід до навчання називають «знизу вгору» (bottom-up) за аналогією з висхідним програмуванням, при цьому більшість зусиль так і залишаються «внизу», оскільки при традиційному навчанні рідко йдеться про створення реальних програм.

Інший підхід відомий як програмування «зверху вниз» (top-down). Застосування цього підходу в навчанні означає, що студенти можуть приступати до розв'язування більш-менш складних задач на ранніх етапах навчання, використовуючи готові

інструменти – шаблони та бібліотечні модулі, не витрачаючи часу і зусиль на самостійну реалізацію компонентів [8]. Таким чином, студенти здатні робити цікаві і серйозні речі значно раніше, зосереджуючись переважно на високорівневій логіці роботи програми і займаючись конструюванням програм із готових компонентів.

Пошук правильного підходу до навчання спричинений тим, що дуже часто студенти «не можуть написати прийнятну програму навіть після завершення семестрового курсу програмування» [4]. Не секрет, що і після другого семестру навчання виявляються такі студенти. Отримавши вступний інструктаж та кілька типових прикладів, студенти швидко починають «ліпити» власні програми з відомих їм операторів, одержуючи примітивні або відверто нікчемні програми [4]. Їм просто бракує досвіду.

Підхід професійних програмістів до написання програм відрізняється від студентського. Вони рідко починають творити програми «з нуля», оскільки вже озброєні досвідом попередніх успішних проектів, тому зайняті, у першу чергу, плануванням та високорівневим проектуванням рішення, тримаючи в полі зору бібліотечні та відновлюючи в пам’яті раніше реалізовані власні компоненти.

Власне, початківці теж починають не зовсім з нуля. Їх перша С-програма типу «Hello, world!» використовує функцію стандартної бібліотеки printf(), невдовзі доповнюючись функцією scanf(). Це вписується у відомий шаблон для побудови програм «введення-обробка-виведення» (Input-Processing-Output або IPO) [9]. Тобто, студенти зазвичай самостійно реалізують лише частину «обробка» шаблону IPO, а для частин «введення» та «виведення» застосовують готові бібліотечні функції.

Аналіз труднощів, з якими стикаються студенти, свідчить, що підвищення ефективності засвоєння мови асемблера можна досягти різними шляхами, зокрема за допомогою шаблонів – зразків успішних рішень [1]. Аби швидше сформувати навички створення «правильних» програм студентові потрібен набір готових будівельних блоків, а також шаблони та приклади їх застосування. Наукові дослідження, присвячені ефективності навчання програмуванню початківців [4, 5, 7], показують значний прогрес студентів при використанні готових компонентів. Так, в роботі [7] пропонується будувати процес навчання та розвивати навички розв’язування задач на основі готових модулів – підпрограм, визначається набір таких модулів, принципи формування структури програми з готових модулів та приклади типових задач, пропонується також залучати студентів до написання корисних підпрограм. Останнім часом все частіше вживається термін «навчання на основі шаблонів» (pattern-based instruction) як педагогічний прийом, основною метою якого є, скоріше, розвиток навичок алгоритмізації розв’язування задач, ніж деталі синтаксису мови програмування [5, 6] та ін. Питанням розробки системи ефективних дидактичних засобів для підтримки процесу вивчення низькорівневого програмуванню приділяється недостатньо уваги.

Метою даної статі є обґрунтування доцільності та сфери застосування бібліотечних модулів при вивченні мови асемблера, а також розробка навчальної бібліотеки для вивчення низькорівневого програмування.

Виклад основного матеріалу. Традиційний курс мови асемблера включає наступні питання [2]: системи числення та двійкова арифметика, організація комп’ютера та архітектура процесора, сегментна організація програм, синтаксис асемблера, система

команд процесора, типові керуючі структури, структури даних, макрозасоби, модульне програмування. За наявності часу можуть додаватися окремі додаткові теми: ланцюжкові команди, програмування математичного співпроцесора, дискові операції, інтерфейс з мовами високого рівня, програмування апаратних пристроїв та ін.

Класичний підхід до вивчення асемблера відбивається й у структурі підручників з дисципліни, наприклад відомого підручника з мови асемблера [2]. За винятком двох останніх глав, присвячених спеціальним питанням, і додатків, навчальний матеріал розподіляється таким чином. Близько двох третин займають питання апаратної архітектури, арифметичних основ обчислень, синтаксису та опису команд процесора. Третина підручника присвячена питанням, які представляють інтерес для практичного програмування – керуючі структури, ланцюжкові команди, складні структури даних, макрозасоби та модульне програмування. Процесор має кілька сотень команд, тому їх послідовне вивчення окремим блоком слабко зацікавлює і мотивує студентів.

Вивчення мов програмування деякою мірою нагадує вивчення іноземних мов. Візьмемо такий вид діяльності як вивчення іноземних слів. Чи це корисно для вивчення мови? Звичайно. Чи це просуває до поставленої мети? Безумовно. Чи це приклад ефективної методики? Кожен, хто вчив слова напам’ять, знає відповідь на це питання. Продовжимо аналогії. Не секрет, що мову вивчають, у першу чергу, для того аби нею спілкуватися. Чи знання слів і правил граматики гарантує складання правильних речень у заданому контексті? На жаль, не гарантує. Будь-який першокласник своєю рідною мовою буде говорити краще. Потрібне систематичне слухання і відтворення правильних мовних шаблонів, авторами яких є, безумовно, носії мови. Заклики про те, що знаючи дві-три сотні слів, можна вільно спілкуватися іноземною мовою, скоріше просто спонукають до вивчення мови, ніж описують реальну картину.

Система команд процесора містить сотні команд, кількість найбільш уживаних можна обмежити кількома десятками. Кінцевою метою вивчення мови програмування є створення правильних і ефективних програм. Проте, навіть досконале знання команд асемблера не дає навичок їх вірного застосування. Аналогічна ситуація з синтаксисом. Знання синтаксису важливе, програма, що містить хоча б одну помилку, просто не буде скомпільована. Але немає жодної гарантії, що синтаксично вірна програма буде належно виконувати передбачені функції. Програмісту потрібні навички ефективного розв’язування практичних задач засобами обраної мови програмування. Звідки студенту взяти ефективні прийоми розв’язку тих чи інших задач? У першу чергу, з досвіду професійних програмістів, який передається через шаблони, бібліотеки підпрограм та код успішно реалізованих проектів. Програми корисно і потрібно не лише писати, а й читати.

Вивчення всіх архітектурних та синтаксичних особливостей мови асемблера потребує значної кількості годин, що не вкладається в сучасні навчальні плани. Разом з тим, мінімальні навички низькорівневого програмування студентам потрібно здобути в короткий термін. У цьому може допомогти аналіз навчальних програм, виявлення пріоритетних тем для аудиторного вивчення, ретельний добір задач та їх розподіл між видами занять, розробка ефективних компенсаторів для тем, недостатньо висвітлених в аудиторії або винесених на самостійне вивчення.

На сьогодні до лекційного курсу з «Архітектури обчислювальних систем» реально

включити питання основ програмування мовою асемблера, структури та синтаксису програм, принципів реалізації типових керуючих структур мовою асемблера, основ модульного програмування з використання підпрограм, організації стеку, роботи з масивами та структурами даних. Лабораторні заняття можуть охопити створення та відлагодження програм на асемблері, програмування із використанням арифметичних та логічних команд, керування ходом виконання програми на асемблері, та основ модульного програмування. Для розгляду найбільш потужного засобу асемблера – системи команд та організації введення-виведення аудиторного часу явно недостатньо.

Досвід автора свідчить, що однією з найбільших проблем на початкових етапах вивчення асемблера є повна відсутність високорівневих і складність низькорівневих засобів введення-виведення. Поширена думка, що написання програм на «чистому» Асемблері – заняття надзвичайно важке і малоприємне. Навіть проста асемблерна програма вимагає значних зусиль, особливо на початкових етапах освоєння мови. Для прикладу, виведення на екран відомого «Hello, world!» на чистому асемблері із доступом до апаратних ресурсів системи може містити сотні команд. Хоча системні виклики BIOS та DOS полегшують завдання, вони все ще залишаються значно складнішими, ніж WRITELN() на Pascal чи printf() на C. Створення програм, які не мають засобів діалогу з користувачем, слабко мотивують студентів до навчання.

Пропонується з питань, на вивчення яких відводиться недостатньо часу, надати студенту відповідні компенсатори, які допоможуть заповнити прогалини і успішно опанувати інші питання. Роль таких компенсаторів можуть виконувати шаблони (організаційні, структурні, алгоритмічні та ідіоми) [1], бібліотеки підпрограм, зразки коду, контекстні довідники з команд процесора, калькулятори та симулятори команд.

Існують асемблерні бібліотеки підпрограм різних виробників. Серед них можна виділити комерційні бібліотеки для професійних програмістів, основним завданням яких є прискорення розробки програм і одержання стабільного коду. Іншу категорію складають навчальні бібліотеки, створені з метою полегшення вивчення мови Асемблера. Прикладом є «The UCR Standard Library for 80x86 Assembly Language Programmers», створена в Каліфорнійському університеті США відомим фахівцем Рендалом Хайдом. UCR Standard Library – це кілька сотень процедур за зразком стандартної бібліотеки "C": функції введення, виведення, обробки рядків, перетворень, порівнянь, перевірок, керування пам’яттю, обробки чисел з плаваючою комою, роботи з файлами та ін. Назви функцій бібліотеки короткі і не завжди інформативні, передача параметрів здійснюється різними способами, у тому числі через потік коду, що утруднює опанування функцій бібліотеки.

З метою забезпечення студентів простими у використанні функціями введення-виведення автором була запропонована і розроблена навчальна бібліотека асемблерних підпрограм, яка отримала умовну назву asmio16. Бібліотека містить десять функцій введення-виведення та кілька допоміжних функцій.

Функції бібліотеки з точки зору користувача займають проміжний рівень. Назви функцій визначено на високому рівні в стилі WinAPI, що дає можливість легко зрозуміти їх призначення навіть без довідки. Виклик функцій і передача параметрів здійснюються в стилі асемблера, аби відчувати «дух низького рівня». Функції мають від 0 до 2 параметрів, тому вони передаються через регістри. Параметр-число передається через регістр AX,

параметр-адреса через регістр DX, параметр-лічильник через CX. Функції повертають значення через AX, а прапорець CF свідчить про помилки уведення чисел функціями ReadInt та ReadHex. Документація містить інструкції з встановлення та підключення бібліотеки, а також опис функцій з прикладами їх використання.

Таблиця 1

Функції введення-виведення навчальної бібліотеки asmio16

`WriteChar`	виводить символ на екран
`WriteString`	виводить рядок символів на екран
`WriteInt`	виводить на екран двобайтове число як десяткове із знаком
`WriteUnsigned`	виводить на екран двобайтове число як десяткове беззнакове
`WriteByteHex`	виводить однобайтове число на екран як шістнадцяткове
`WriteWordHex`	виводить двобайтове число на екран як шістнадцяткове
`ReadChar`	вводить один символ з клавіатури
`ReadString`	вводить рядок символів з клавіатури
`ReadInt`	вводить з клавіатури десяткове число із знаком і перетворює його в двобайтове двійкове число
`ReadHex`	вводить з клавіатури шістнадцяткове число і перетворює його в двобайтове двійкове число

В подальшому доцільно зосередитися на розробці системи завдань для лабораторних робіт та самостійної роботи, аналізі проблем початківців у розв'язуванні цих задач і доповненні навчальної бібліотеки новими функціями.

Висновки. В межах курсу «Архітектура обчислювальних систем» розглядаються лише найголовніші питання низькорівневого програмування, а формуванню практичних навичок присвячується всього кілька лабораторних занять. Тому пропонується надати студентам засоби, покликані компенсувати нестачу аудиторної практики – шаблони і бібліотечні модулі. Розроблена навчальна бібліотека підпрограм охоплює одну з найважчих для студента проблем – введення-виведення. Бібліотека має функції введення-виведення символів, рядків, десяткових та шістнадцяткових чисел, що дає можливість студентам зосередитися на інших питаннях високорівневої логіки програм.

БІБЛІОГРАФІЯ

1. Баранюк О. Роль шаблонів у навчанні низькорівневому програмуванню / О. Баранюк // Наукові записки. Серія : проблеми методики фізико-математичної і технологічної освіти. – Кіровоград : РВВ КДПУ ім. В. Винниченка, 2014. – Вип. 5. – С. 7–12.

2. Юров В.И. Assembler : учебник для вузов / В.И. Юров. – 2-е изд. – СПб. : Питер, 2003. – 637 с.

3. Agarwal K.K., Agarwal A.A. Do We Need a Separate Assembly Language Programming Course? / K.K. Agarwal, A.A. Agarwal // Journal of Computing Sciences in Colleges. – 2004. – V. 19. – No. 4. – pp. 246–251.

4. Devedzic V. Software Patterns / V. Devedzic // Chang, S.K. Handbook of Software Engineering and Knowledge Engineering. – Singapore : World Scientific Publishing Co., 2002. – pp. 645–671.

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

6. Haberman B., Muller O. Teaching Abstraction to Novices: Pattern-Based and ADT-Based Problem-Solving Processes / B. Haberman, O. Muller // Frontiers in Education Conference. – New York, 2008. – pp. F1C7–F1C12.

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

8. Reek M.M. A Top-down Approach to Teaching Programming / M.M. Reek // ACM SIGCSE Bulletin. – 1995. – Vol. 27. – No. 1. – pp. 6–9.

9. Winslow L. Programming Pedagogy : A Psychological Overview / L. Winslow // SIGCSE Bulletin. – 1996. – № 28 (3). – pp. 17–22.

ВІДОМОСТІ ПРО АВТОРА

Баранюк Олександр Филимонович – доцент кафедри інформатики Кіровоградського державного педагогічного університету імені Володиимра. Винниченка, кандидат технічних наук.

Коло наукових інтересів: моделювання інформаційних систем, проблеми викладання комп'ютерних наук у вищій школі.

РОЛЬ І МІСЦЕ КУЛЬТУРНО-ІСТОРИЧНОЇ СКЛАДОВОЇ ЗМІСТУ ОСВІТИ У ПІДГОТОВЦІ МАЙБУТНІХ ВЧИТЕЛІВ МАТЕМАТИКИ В ПЕДАГОГІЧНОМУ УНІВЕРСИТЕТІ

Павло БЄЛЬЧЕВ, Анатолій ПАВЛЕНКО

У статті розглянуті теоретичні засади цілісності змісту вищої математичної освіти. Визначені роль і місце культурно-історичної складової змісту математичної освіти у підготовці майбутніх вчителів математики.

The article describes the theoretical foundations of integrity content of higher mathematics education. Defines the role and place of cultural and historical component content of mathematics education in training future teachers of mathematics.

Постановка проблеми. В Україні законодавчо закріплені положення про те, що освіта є основою інтелектуального, культурного, духовного, соціального, економічного розвитку суспільства і держави. Освіта цілеспрямована на формування громадян, здатних до свідомого суспільного вибору, збагачення на цій основі інтелектуального, творчого, культурного потенціалу народу. Провідними в освіті визнані засади гуманізму, демократії, національної свідомості, взаємоповаги між націями і народами (закони України «Про освіту», «Про вищу освіту»).

На цих засадах у статті 6 були визначені основні принципи освіти, де серед інших задекларовано: гуманізм, демократизм, пріоритетність загальнолюдських духовних цінностей; органічний зв'язок із світовою та національною історією, культурою, традиціями.

Якщо наукові знання з історії математики знаходять свою актуалізацію у змісті навчального курсу «Історія математики» для майбутніх вчителів математики в педагогічному університеті, науково-популярних виданнях (В.Г.Бевз, М.І.Кованцов, Я.І.Перельман та ін.), то проблема теоретичного обгрунтування і визначення ролі і місця перелічених вище актуальних освітніх принципів і, зокрема, культурно-історичної складової змісту математичної освіти ще чекає на своє вирішення.

Зв'язки математичної освіти із світовою та національною історією, культурою, традиціями повинні увійти до змісту (курикулуму) математичної освіти та інтегровані у єдиній культурно-історичній освітній складовій. Ця єдність зумовлена наступними вихідними положеннями: