І. ПРОБЛЕМИ МЕТОДИКИ НАВЧАННЯ МАТЕМАТИЧНИХ ДИСЦИПЛІН

РОЛЬ ШАБЛОНІВ У НАВЧАННІ НИЗЬКОРІВНЕВОМУ ПРОГРАМУВАННЮ

Олександр БАРАНЮК

У статті подається аналіз проблеми підвищення ефективності навчання низькорівневому програмуванню в умовах дефіциту навчальних годин шляхом використання шаблонів та розкриваються основні напрями застосування шаблонів.

The article presents an analysis of the problem of increasing low-level programming teaching under conditions of training time restrictions by means of using patterns and main scopes of patterns applying are considered.

Постановка проблеми. Серед фахівців галузі комп'ютерних наук вже давно триває дискусія з приводу змісту та обсягу дисциплін, присвячених вивченню архітектури обчислювальних систем та основ низькорівневого програмування.

Обсяг знань в комп'ютерних науках стрімко зростає, з'являються нові мережеві, інформаційні та мультимедійні технології і відповідні дисципліни в університетах. Це загальносвітова тенденція. Аналізуючи рекомендації по викладанню інформатики в університетах (Computer Science Curricula), які регулярно розробляються Асоціацією обчислювальної техніки (ACM) та Інститутом інженерів електротехніки та електроніки (IEEE), можна зробити висновок про те, що кількість областей знань інформатики, які пропонуються до вивчення зросла з 14 до 18 лише за роки нинішнього століття.

Знайти час на вивчення усіх дисциплін стає все важче. З появою нових комп'ютерних дисциплін, зменшується кількість часу на вивчення старих дисциплін або певних тем, окремі дисципліни об'єднуються з іншими або й зовсім вилучаються з навчальних планів. Так, вітчизняний галузевий стандарт з напряму підготовки «Інформатика» 2010 року 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-виводу, роботи з пам'яттю, низькорівневої реалізації складних логічних структур мов програмування високого рівня та навичок трансляції програм у машинні коди. Вимоги Computer Science Curricula 2013 ще ширші. В результаті на формування навичок низькорівневого програмування вдається виділити кілька лекцій і кілька лабораторних робіт.

Проблема полягає в тому, аби за рахунок сучасних педагогічних прийомів студент зміг оволодіти основами програмування на асемблері в короткий термін. Один із можливих підходів до вирішення цієї проблеми – використання простих і ефективних шаблонів на всіх етапах створення комп'ютерної програми, що може дати реальну економію часу при написанні та можливість одержати правильний, читабельний і працездатний код.

Аналіз останніх досліджень і публікацій. Студенти відчувають значні труднощі при вивченні низькорівневого програмування. Дослідження з проблеми свідчать, що ці труднощі пов'язані не стільки із складністю чи незвичністю синтаксису асемблера, скільки з відсутністю загальних навичок розв'язування задач. Новачки витрачають мало часу на аналіз задачі, планування та конструювання програми. Основним підходом залишається метод 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 мають погане розуміння основного потоку виконання програми. Однією з найбільших проблем є слабка здатність до розв'язування логічних та математичних задач [5].

Традиційний підхід до навчання програмуванню, більшість підручників та посібників передбачає 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 [9]. Такі складні структури даних як списки, черги, стеки та дерева вивчаються, відповідно, в кінці курсу, коли часу на їх ґрунтовне засвоєння вже не залишається. Причому, занадто багато уваги приділяється механізмам їх реалізації. Цей підхід до навчання часто 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.

У випадку асемблера вивчення всіх архітектурних та синтаксичних особливостей мови потребує значної кількості годин, що не вкладається в сучасні навчальні плани. Мінімальні навички низькорівневого програмування студентам потрібно здобути в короткий термін.

Інший підхід відомий як програмування «зверху вниз» (top-down), низхідне програмування або покрокове уточнення. Застосування цього підходу в навчанні передбачає, що студенти можуть приступати до розв'язування задач за допомогою наявних інструментів (бібліотечних компонентів) замість того, аби витрачати час на самостійну реалізацію цих компонентів «з нуля» [9]. Таким чином, студенти здатні робити цікаві і серйозні речі значно раніше, ніж за традиційного підходу. При цьому вони зосереджуються переважно на високорівневій логіці роботи програми і зайняті конструюванням програм на основі готових компонентів та бібліотечних функцій.

Пошук правильного підходу до навчання спричинений тим, що дуже часто «багато студентів не можуть написати прийнятну програму навіть після завершення семестрового курсу програмування» [4]. Не секрет, що і після другого семестру навчання легко знайти таких студентів. Після вступного інструктажу та кількох 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 значно відрізняється від студентського. Вони рідко починають творити програми «з нуля», оскільки вже озброєні досвідом попередніх, у тому числі успішних проектів, тому зайняті, у першу чергу, високорівневим проектуванням рішення, тримаючи в полі зору бібліотечні та відновлюючи в пам'яті раніше реалізовані власні компоненти, на які можна опиратися в наступних проектах.

Аналіз труднощів, з якими стикаються студенти, свідчить, що підвищення ефективності засвоєння мови асемблера у вищих навчальних закладах можна досягти різними шляхами, зокрема шляхом активного застосування шаблонів проектування та програмування [1]. Аби швидше сформувати навички створення «правильних» програм студентові потрібен набір готових шаблонів та приклади їх застосування.

Проведено багато наукових досліджень, присвячених використанню шаблонів на початкових етапах навчання програмуванню [3], [4], [6], [8] та ін., які показують значний прогрес студентів при вивченні мов програмування високого рівня за рахунок використання готових бібліотечних компонентів. Зокрема в роботі [8] пропонується будувати процес навчання та розвивати здібності студентів до розв'язування задач на основі використання шаблонів, даються вказівки щодо складання набору необхідних шаблонів та приклади задач, розв'язування яких може бути полегшене на основі шаблонів. В науковому обігу сформувався та закріпився термін «навчання на основі шаблонів» (pattern-based instruction) як педагогічний прийом, основною метою якого є розвиток навичок алгоритмізації розв'язування задач [6], [8] та ін.

На жаль, питанням застосування шаблонів при навчанні низькорівневому програмуванню приділяється дуже мало уваги.

Метою даної статі є обґрунтування доцільності, сфери застосування, та методики використання шаблонів при вивченні мови асемблера.

Виклад основного матеріалу. Шаблон – це зразок, придатний для наслідування. Батьком шаблонів вважають Крістофера Александера, який розробив і описав значну

кількість архітектурних шаблонів для будівництва та став одним із засновників «мови шаблонів» [3].

Шаблон у програмуванні це спроба описати успішне рішення повторюваної проблеми. Шаблони дають можливість використовувати колективний досвід професійних програмістів. Найбільш відомими є шаблони проектування (design patterns), які набули поширення в об'єктно-орієнтованому програмуванні. Слід зазначити, що програмні шаблони спеціально не придумують, їх помічають в попередніх реалізованих програмах, описують, поміщають в каталоги і використовують в наступних проектах. Шаблони покликані поширювати фахові знання, завдяки чому вони є цінними для студентів та новачків у програмуванні.

Шаблони у програмуванні корисні не лише при написанні коду. Їх можна застосовувати на різних етапах створення програми. Відомо, що процес розв'язування задачі шляхом програмування може включати такі етапи [10]: 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.

Практично на кожному з цих етапів можна використовувати шаблони.

Існують різні спроби класифікації шаблонів. Зокрема, в роботі [3] виділяють шаблони проектування, шаблони рівня коду або ідіоми, шаблони аналізу, шаблони програмної архітектури, організаційні шаблони та шаблони процесів.

Група вчених з ізраїльських університетів [8] виділяють *елементарні шаблони* (вибір, повторення, робота з масивами), *алгоритмічні шаблони* (підрахунок, накопичення, пошук максимуму, перевірка на рівність/нерівність, зміна порядку елементів, аналіз та синтез чисел) та *педагогічні шаблони*.

В статті [4] пропонуються наступні категорії шаблонів: *прості дії* (введення даних із запрошенням, обмін значеннями), *умовні вирази* (якщо, за виключенням, залежні/вкладені, послідовні умовні вирази), проблеми, засновані на *використанні циклів* (підрахунок, накопичення, введення з файлу, лінійний пошук, двійковий пошук), проблеми, засновані на *керуванні циклами* (сторожова умова чи контроль лічильника, контроль кількох умов) та більш *складні логічні шаблони* (переривання керування).

Для того, аби можна було поширювати і повторно використовувати шаблони їх треба відповідним чином описати. Структура опису залежить від типу шаблону та прийнятої практики і може містити від кількох до кільканадцяти пунктів. На думку Александера в описі кожного шаблону повинні бути: ім'я шаблону; призначення шаблону і опис задачі, яку він покликаний розв'язати; спосіб розв'язання поставленої задачі; обмеження і вимоги, які слід приймати до уваги при розв'язанні задачі [3].

Структура опису шаблону проектування у відповідності з рекомендаціями групи розробників, скоріше відомих як банда чотирьох (Gang of Four) або GoF, наступна: назва і класифікація шаблону, призначення, альтернативна назва, мотивація, застосовуваність, структура, учасники, відношення, результати, реалізація, приклад коду, відомі застосування, пов'язані шаблони.

Узагальнена структура опису шаблонів аналізу, організаційних, шаблонів та шаблонів процесів згідно [3]:

- задача – задача, для розв'язку якої призначений шаблон;
- контекст – ситуація, в якій шаблон є корисним;
- зусилля – засоби, які приводять до формування розв'язку;
- розв'язок – результат, до якого приводять зусилля;
- пов'язані шаблони – з якими іншими шаблонами пов'язаний даний.

Отже, мінімальна структура опису шаблону повинна включати такі обов'язкові елементи.

- контекст – ситуація, що породжує проблему.
- проблема – повторювана проблема, що постає в даному контексті.
- розв'язок – випробуване рішення проблеми.

В процесі розробки програмного забезпечення можна використовувати широку гаму шаблонів. Велика кількість шаблонів може породжувати нову проблему, тим більше для новачків, – навчитися розпізнавати шаблони і правильно обирати шаблони, придатні для

вирішення конкретної проблеми. Аби було легше орієнтуватися в шаблонах, їм дають вдалі осмислені назви, ретельно описують і каталогізують. Для прикладу члени команди GoF спочатку описали 23 основних шаблони проектування, пізніше міжнародна спільнота доповнила і розвинула відомі шаблони, зараз їх кількість вимірюється сотнями.

При наявності великої кількості шаблонів не варто очікувати значного прогресу в розвитку навичок створення програм студентами, оскільки на зміну вивченню синтаксису мови програмування та алгоритмів приходить вивчення шаблонів. В роботі [4] підкреслюється, що пакет із 30 шаблонів не може допомогти студентам-початківцям швидко здобути навички застосування шаблонів, а тому слід сконцентруватися на значно меншому наборі. Якщо навчити студентів розрізняти і ефективно застосовувати кілька основних шаблонів, виявити в них точки варіації, навчити їх модифікувати і компонувати шаблони, в подальшому вони зможуть розширити діапазон використання шаблонів, а потім навчитися створювати власні.

Отже, варто чітко окреслити вузьке коло основних проблем, які доводиться вирішувати студентам під час вивчення предмету та найбільш значні труднощі, з якими вони стикаються і зосередитися на створенні компактного набору шаблонів, покликаного усунути найголовніші труднощі.

Опитування студентів, які пройшли традиційний курс вивчення асемблера, серед найскладніших питань мови асемблера виявило: способи передачі параметрів у підпрограми, команди передачі керування, складені типи даних, директиви визначення даних, способи адресації. З іншого боку, найлегшими питаннями для студентів були: компіляція програми, типи даних, синтаксис, відлагодження програми, структура програми. Досвід автора показує, що до найскладніших питань не потрапило питання керування циклами, оскільки студенти вже мали певні шаблони і приклади використання циклів. Також, студенти недооцінили серйозність питання відлагодження програм, оскільки їм не доводилося відлагоджувати більш-менш складні реальні програми.

Набір шаблонів для вивчення мови програмування повинен бути компактним та всеохоплюючим [4], базуватися не на мовних конструкціях, структурах даних чи алгоритмах, а на задачах, які пропонується студентам розв'язувати під час занять. Крім того, набір шаблонів повинен узгоджуватись з вимогами ОПП до знань та вмінь студентів з цього предмету. Це непроста задача. На нашу думку, в шаблонах для вивчення основ низькорівневого програмування мають бути відображені наступні моменти: структура асемблерної програм та її трансляція, способи визначення даних та доступу до них, введення-виведення даних, реалізація основних керуючих структур, обробка масивів даних, використання підпрограм. На відміну від мов високого рівня, способи реалізації високорівневих мовних структур мовою асемблера також не можна залишати поза увагою.

Вищенаведені міркування дають змогу виділити ключові категорії і напрями застосування шаблонів при навчанні низькорівневому програмуванню:

1. Організаційні – організація робочого середовища.
2. Структурні – типові структури асемблерних програм.
3. Алгоритмічні – основні алгоритми обробки даних та їх кодування мовою асемблера.
4. Ідіоми – шаблони рівня коду.

На даному етапі запропоновано близько двох десятків шаблонів першої необхідності і ведеться робота по їх каталогізації. Приклади шаблонів різних категорій наведено в таблиці.

Навіть попередній аналіз показує, що перелік необхідних для програмування шаблонів може бути занадто широким. Задовольнити умову всеосяжності і компактності набору шаблонів непросто. В умовах критичного дефіциту навчальних годин виходом може бути формування комплекту задач, запропонованих до розв'язку студентам і визначення набору шаблонів, достатніх для розв'язання кожної задачі. Для прикладу, елементарна задача-програма «Hello World!», з якої починається вивчення будь-якої мови програмування, потребує застосування шаблонів e-One-Folder, s-Easy-Start, p-Write-String.

В подальшому є сенс зосередитися на розширенні та удосконаленні запропонованого набору шаблонів, що має стати предметом наукової дискусії.

Таблиця 1

Приклади шаблонів для низькорівневого програмування

Група	Назва	Короткий опис
Середовище	e-One-Folder	Всі файли в одній папці
	e-Set-of-Folders	Повний набір папок
Структурні	s-Easy-Start	Макрос .begin + бібліотечні функції
	s-IPO	Введення – обробка – виведення
Алгоритмічні	a- Guarded-Action	If <…> then <…>
	a-Alternative-Action	If <…> then <…> else <…>
	a-Process-All-Items	Обробка всіх елементів (цикл)
	a-Process-Until-Done	Обробка до виконання умови
Ідіоми	p-Item-Access	Доступ до елемента
	p-Write-String	Виведення рядка
	p-Read-String	Читання рядка
	p-String-Length	Визначення довжини рядка
	p-Bit-Test	Перевірка біту даних
	p-Bit-On	Встановити біт даних
	p-Bit-Off	Скинути біт даних

Разом з тим кожний викладач, на основі власного досвіду та практики, може сформувати набір завдань до розв'язання та укомплектувати їх шаблонами. Для студентів, які мають здібності і бажання до поглибленого вивчення низькорівневого програмування слід надати інструкції по створенню програм шляхом інтеграції шаблонів, їх модифікації та створенню власних шаблонів. Важливим напрямом майбутніх досліджень повинні стати також програмні засоби для автоматичної генерації асемблерного коду на основі каталогу шаблонів.

Висновки. Кількість годин, відведених на вивчення низькорівневого програмування в навчальних планах останнім часом скорочується. В курсі «Архітектура обчислювальних систем» нині розглядаються лише найголовніші питання, а формуванню практичних навичок вдається присвятити всього кілька лабораторних занять. В цих умовах пропонується формувати елементарні навички програмування мовою асемблера за допомогою активного впровадження шаблонів кількох основних категорій: організація робочого середовища, структура програми, алгоритмічні шаблони, програмні шаблони. Запропоновано мінімальний набір шаблонів, здатний забезпечити розв'язання першочергових завдань низькорівневого програмування, передбачених навчальною програмою.

В рамках годин, відведених навчальними планами на вивчення низькорівневого програмування, мова може йти лише про ознайомлення студентів з основами створення асемблерних програм, окреслення їх сфери застосування та демонстрацію розв'язування типових задач засобами асемблера.

БІБЛІОГРАФІЯ

1. Баранюк О. Пошук шляхів підвищення ефективності вивчення мови асемблера / О. Баранюк // Наукові записки. Серія : проблеми методики фізико-математичної і технологічної освіти. – Кіровоград : РВВ КДПУ ім. В. Винниченка, 2011. – Вип. 2. – С. 18–26.

2. 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.

3. 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.

4. East, J.P., Thomas S.R., Wallingford E., Beck W., Drake J. Pattern-based programming instruction / J.P. East, S. R. Thomas, E. Wallingford, W. Beck, J. Drake // Proceedings of the ASEE Annual Conference and Exposition. – Washington, 1996. – Режим доступу: http://www.cs.uni.edu/~wallingf/patterns/papers/asee96.pdf

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. Muller O., Haberman B., Averbuch H. (An almost) pedagogical pattern for pattern-based problem-solving instruction / O. Muller, B. Haberman, H. Averbuch // ACM SIGCSE Bulletin. – 2004. – Vol. 36. – No. 3. – pp. 102–106.

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

10. 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 КДПУ ім. В.Винниченка, кандидат технічних наук.

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