I. ПРОБЛЕМИ МЕТОДИКИ НАВЧАННЯ МАТЕМАТИЧНИХ ДИСЦИПЛІН

ВИКОРИСТАННЯ СТРУКТУРНОГО ПРОГРАМУВАННЯ В ПРОЦЕСІ ВИВЧЕННЯ МОВИ АСЕМБЛЕРА

Олександр БАРАНЮК

У статті подається аналіз проблеми використання структурного підходу в процесі навчання програмуванню мовою асемблера і розкриваються основні складові структурованості програм.

The article presents an analysis of the problem of using the structured approach in learning of assembly language programming and reveals the main components of program structuring.

Більшість прикладного і значна частина системного програмного забезпечення створюється мовами високого рівня. Програми на мовах високого рівня легше пишуться і читаються, значно швидше розробляються і відлагоджуються, їх легко переносити на інші платформи. Мови високого рівня ефективно застосовують для створення Web-додатків, «хмарних» обчислень (Cloud Computing), мультимедійних додатків та системного програмування.

Серед вимог до мов програмування, які слід обирати для навчального процесу вчені відзначають простий і читабельний синтаксис, високий рівень ключових абстракцій, невеликий обсяг конструкцій мови, тісний зв'язок із реальним програмуванням. Звичайно, ці вимоги певною мірою реалізуються у мовах програмування високого рівня. Важливим фактором на користь високорівневого програмування залишається швидкість розробки програм.

Мова асемблера належить до низькорівневих і слабко корелюється з викладеними вище вимогами. Асемблер дає можливість доступу до ресурсів комп'ютера і створювати ефективний код, хоча це не забезпечується автоматично. Мовою асемблера пишуться найбільш критичні ділянки коду, системне програмне забезпечення, драйвери та утиліти. Стати професіоналом у програмуванні мовою асемблера непросто, потрібно близько десяти років, щоб перетворити початківця на професійного програміста.

Проблемам навчання мовам програмування високого рівня присвячено багато досліджень вітчизняних та зарубіжних вчених, питання підвищення ефективності засвоєння асемблера практично не піднімаються – вважається, що мова асемблера – це доля професіоналів.

Відомо, що студенти відчувають труднощі при вивченні мов програмування. У своєму дослідженні з цієї проблеми А. Гомес і А. Мендес [8] відзначають, що ці труднощі пов'язані з тим, що стандартні методи подачі матеріалу, орієнтовані на синтаксис, не сприяють виробленню навичок розв'язування конкретних задач програмування; студенти не прикладають достатньо зусиль для глибокого засвоєння необхідних компетенцій; однією з найбільших проблем є слабка здатність студентів до розв'язання логічних та математичних задач.

Значна частина літератури з проблеми свідчить, що початківці відчувають брак спеціальних знань та фахових навичок. Леон Вінслоу [12] основними проблемами початківців у програмуванні вважає відсутність у них адекватної ментальної моделі предметної області, поверхневі й неміцні знання предмету, нездатність застосування загальних стратегій розв'язку задач до конкретних завдань, схильність до програмування «рядок за рядком» замість глибокого аналізу проблеми і пошуку ефективного розв'язку.

Новачки витрачають мало часу на аналіз задачі, планування, проектування та тестування коду. Їх основним підходом стає метод спроб і помилок (code-and-fix), який має на меті у будь-який спосіб одержати працюючу програму. Новачки, зазвичай, досить слабкі у відстеженні власного коду, їм бракує належного алгоритмічного мислення, а тому вони мають погане розуміння основного потоку виконання програми.

Рон Портер відзначає, що більшість проблем з програмуванням пов'язані скоріше із загальними аспектами розв'язування задач, ніж із деталями мови програмування [11]. Це означає, що перш, ніж писати програму, потрібно розв'язати задачу в загальних термінах. Повинен існувати етап, який передбачає аналіз і розуміння задачі та пошук її розв'язку в концептуальному плані, що передує її реалізації в термінах програмування. Етап програмування, у вузькому сенсі, це процес перекладу концептуального рішення в послідовність конструкцій мови програмування і

на цьому рівні вже мало творчості.

Проведено цілий ряд досліджень проблеми навчання програмуванню, зокрема [8, 11, 12]. З'явилися навіть терміни «навчальне програмування» (teaching programming) та «педагогіка програмування» (programming pedagogy). Але більшість досліджень присвячено проблемі навчання мов програмування високого рівня, переважно в межах вступного курсу з комп'ютерних наук. Дуже мало досліджень присвячено навчанню мови програмування асемблера. Вважається, що асемблер – мова програмування для професіоналів і вони здатні успішно її опанувати. Але кожен професіонал проходить через студентську лаву, а від ефективності навчання залежить не лише кількість і якість майбутніх професіоналів але і довжина шляху до їх становлення.

В сучасних навчальних планах на вивчення асемблера як мови програмування відводиться все менше годин, асемблер переводиться в розряд дисциплін за вибором або й зовсім вилучається з навчальних планів як окрема дисципліна. Звичайно ж, цьому можна знайти пояснення. Асемблер як мова програмування вивчається вже приблизно чотири десятиліття. За цей час з'явилися нові цікаві й змістовні курси, які можна запропонувати студентам, що пов'язано із широким розповсюдженням комп'ютерних мереж, Інтернет- та мультимедіа- технологій, баз даних. Значна увага приділяється об'єктно-орієнтованому програмуванню як одній із сучасних парадигм. Проте масштабне використання мікропроцесорних засобів керування і обчислювальних систем як і раніше потребує кваліфікованих кадрів, здатних програмувати мовою асемблера.

Метою даної статі є обґрунтування доцільності та методика використання концепції структурного програмування при вивченні мови асемблера.

Аналіз труднощів, з якими мають справу студенти [1], свідчить, що підвищення ефективності засвоєння мови асемблера студентами вищих навчальних закладів можна досягти шляхом раціональної організації навчальних занять, розробки проблемних навчальних завдань, розвитку загальних навичок розв'язування задач і алгоритмізації, використання переваг структурного програмування і засобів формалізації алгоритмів, широкого застосування шаблонів проектування та програмування, залученням навчальних інтегрованих середовищ розробки програм, спеціально призначених для вивчення мови асемблера,

Основною проблемою при написанні програм мовою асемблера залишається не складність синтаксису чи велика кількість команд, а слабке розуміння задачі і шляхів її розв'язання. Підхід «кодуй і виправляй», який дуже часто використовують на практиці студенти, можна охарактеризувати як відсутність будь-якого підходу.

Більш результативним виявляється проектування програм, у якому можна виділити кілька етапів [12]: розуміння проблеми; визначення способу розв'язання задачі; специфікація алгоритму розв'язання задачі; переведення розв'язку на відповідну мову програмування; тестування і відлагодження програми. Причому, перших три етапи відносяться до процесу проектування рішення задачі, а наступних два – до процесу його програмування.

Сучасне програмне забезпечення являє собою приклад складних систем. Хоча студенти на початкових етапах вивчення мови програмування створюють порівняно прості програми, відсутність належного досвіду призводить до того, що певні задачі виявляються для них досить складними. Проблема полягає в тому, аби навчитися керувати цією складністю [4, 6]. Людський мозок не здатний одночасно утримувати в полі зору багато різноманітної інформації. У. Дал, і К. Хоор відзначають: «Ми обмежені самою природою нашого інтелекту: точне і зв'язне мислення можливе лише в термінах невеликої кількості елементів у кожний окремий відрізок часу» [7, с. 199]. Тому при проектуванні складних систем застосовують принцип розподілу уваги, відомий здавна як принцип «розділяй і володарюй», який у структурному програмуванні отримав назву алгоритмічна декомпозиція.

Структурне програмування, засновником якого вважають голландського науковця Едсгара Дейкстру, відоме з 70-х років XX ст. Воно стосується послідовнісних машин, в яких обчислення виконуються послідовно в часі, тобто команда за командою. За твердженням Е. Дейкстри, бажано, щоб текст програми відображав структуру і послідовність обчислень. Керовані програми повинні не тільки давати правильний результат, а й бути написаними так, щоб інтелектуальні зусилля, необхідні для їх розуміння, відповідали розміру програми. Якщо ми хочемо контролювати обчислення за текстом програми, то нам слід вдаватися до систематичних механізмів слідування, що гарантує відповідність між просуванням обчислень і просуванням по

тексту [7]. При цьому в тексті програми і в обчисленнях, які вона виконує, можна виділити точки відповідності.

Структурне програмування добре узгоджується з концепцією низхідного проектування програм (top-down approach), при якому спочатку створюється головна процедура програми, що складається з послідовних викликів майбутніх підпрограм. Імена підпрограм визначаються необхідною функціональністю задачі, а замість самих підпрограм на початку розробки використовують процедури-заглушки (stubs). Згодом визначають вимоги до підпрограм і надають їм необхідної функціональності. Цей метод був описаний Н. Віртом як метод покрокового уточнення [3]. На кожному кроці деталізується опис чергової підзадачі і супроводжується уточненням даних, призначених для взаємодії підзадач. За допомогою цього методу можна реалізувати достатньо складні задачі програмування і отримати зрозумілий і добре документований код.

Якщо на початку проектування важко визначити склад основних функцій системи, можна почати з типового набору «введення-обробка-виведення» (Input-Process-Output або IPO), який підходить для більшості студентських проектів. Далі аналізується кожен із модулів набору, виділяються підзадачі, які здійснюють певні дії над певними даними і проводиться їх декомпозиція на окремі складові. Так робиться до тих пір, поки одержимо модулі, легкі для безпосереднього програмування.

Оскільки розробнику програми доводиться постійно перемикатися із загальної концепції програми на її деталі, програмний код повинен бути чітко структурованим і легко читатися. Один із визнаних лідерів індустрії програмного забезпечення Кент Бек на основі багаторічного досвіду розробки програм доходить висновку, що під час програмування значна кількість часу витрачається на читання вже написаного коду. «Значно більше вкладень займає не створення нових програм, а модифікація старих» [2, с. 23]. А тому «усунення надлишкової складності полегшує розуміння програм у плані їх читання, використання і модифікації» [2, с. 27].

Ясність програми і легкість її читання повинні бути в центрі уваги програмістів-початківців. Слід пам’ятати, що програмний код пишеться для людини, а не для компілятора. Уайрд Сміт доходить висновку, що ясність коду навіть більш важлива ніж його правильність, тому що ясний код легко виправити ще під час написання програми. Ясний код читабельний і самодокументований, а основу ясності складають хороші імена об’єктів. Ясний код легко підтримувати і використовувати повторно, отже він має довше життя [13].

Оскільки мову програмування Асемблер традиційно вважають важкою для вивчення студентами, про що свідчить і власний досвід автора, при вивченні низькорівневого програмування на перше місце слід поставити читабельність коду і чітку структурованість програми. Підтвердженням цьому може бути основна теорема форматування [4, с. 715], яка говорить, що хороше візуальне форматування виявляє логічну структуру програми.

Серед складових структурованості асемблерного програмного коду можна виділити сегментну організацію програм, використання процедур, модулів та бібліотечних функцій, застосування аналогів високорівневих керуючих структур, належне форматування та доцільні коментарі.

Сегментна організація програм. Ніклаус Вірт визначив програму як сукупність даних і алгоритму. «Програми являють собою в кінцевому рахунку конкретні формулювання абстрактних алгоритмів, засновані на конкретних представленнях і структурах даних». Крім цього, «…ми інтуїтивно відчуваємо, що дані передують алгоритмам: потрібно мати деякі об’єкти, перш ніж виконувати дії над ними» [3, с. 10].

Асемблерна програма може складатися з кількох сегментів. Фізичні сегменти існують в пам’яті машини під час виконання програми, а логічні сегменти виділяються в тексті програми згідно з їх функціональним призначенням (сегменти даних, коду, стеку). Відповідно дані, над якими програма виконує певні дії потрапляють в окремий сегмент, який називають сегментом даних, а програмний код зосереджується в сегменті коду. Отже, навіть у найпростіших програмах на Асемблері потрібно описати і тримати в полі зору як мінімум два сегменти – даних і коду.

Модульне програмування. Н. Вірт відзначав [3], що програмування – це мистецтво конструювання. З метою конструювання слід виділити найпростіші будівельні блоки із вже існуючих програм і дати їх систематизований опис. Використання частин коду попередніх програм є нормальною і схвальною практикою програмування. Створюючи асемблерні програми, які розв’язують більш-менш реальні задачі, студенти помічають, як дуже швидко зростає обсяг

тексту. Тримати всю програму в полі зору стає все важче. Тому доцільно завершені фрагменти коду, оформлені як підпрограми, винести в окремий модуль (модулі), який можна легко підключити до своєї програми на етапі трансляції. Для цього використовують як директиви асемблера (include, includelib) так і можливості командного рядка.

У процесі поділу модулів програми відбувається поступова деталізація алгоритму. Поділ програми на модулі дає можливість не тільки знизити її складність, але й дати набір добре визначених і документованих інтерфейсів, що вже є корисним для розуміння програми. Розвиток модульності вважається хорошою ознакою структурованості коду. «Один із способів покращення системи полягає у підвищенні її модульності – збільшенні кількості добре визначених і вдало пойменованих процедур, які добре роблять одну річ», – відзначає МакКоннелл [4, с. 553].

Використання підпрограм у програмах покликане зменшити їх складність і покращити читабельність. Підпрограми дають можливість уникнути повторень програмного коду, приховати довгі послідовності команд або складні логічні перевірки за вдалими назвами підпрограм. Рекомендується використовувати підпрограми, навіть якщо вони викликаються один раз, аби полегшити розуміння коду. Зазвичай у підпрограми виносять код для обробки особливих випадків або код, залежний від платформи чи апаратних засобів.

Підпрограми повинні мати правильні змістовні назви, які показують, що саме робить підпрограма, для цього назви підпрограм повинні бути достатньо довгими аби передати їх призначення. Назва функції свідчить про значення, яке вона повертає (isDigit, keyPressed), а назва процедури відображає дію над об'єктом (openFile, readName). Параметри, які передаються у процедури, розташовують у порядку, який відповідає послідовності їх використання: вхідні, змінювані, вихідні. Кількість параметрів повинна бути мінімальною.

Використання бібліотек. Не потрібно кожного разу винаходити велосипед. Існує велика кількість бібліотечних процедур. В них є тисячі реалізованих функцій. Можна і треба їх використовувати, Це значно спрощує розробку програм. Спочатку слід дослідити наявні бібліотечні функції і вибрати необхідні, а вже потім створювати свої. Власні вдалі функції, придатні для повторного використання, потрібно поміщати у власні бібліотеки.

Програмування на асемблері у структурованому вигляді. Сучасні мови програмування високого рівня автоматично підтримують концепцію структурного програмування. Асемблер не має такої підтримки, тому пропонується програмувати на асемблері у структурованому вигляді [10]. Структурне програмування використовує три основні керуючі структури (послідовність, вибір, повторення), з яких може бути побудована будь-яка програма. Кожна така структура має одну точку входу і одну точку виходу. Всі інші програми будуються шляхом пакетування або вкладення структур. При цьому підтримується ясність програми, необхідна послідовність обчислень, простота керування і можливість відстеження просування обчислень.

Мови високого рівня мають у своєму арсеналі відповідні керуючі структури (if, if … else, for, while, do … while), позначені ключовими словами, за якими їх легко розпізнати в тексті. Мова асемблера, зазвичай, не має таких структур, хоча набір команд асемблера, придатних для побудови будь-якої структури керування досить широкий. Оскільки мова йде про чітку логічну структуру програми, легкість її читання і наявний попередній досвід програмування, можна встановити відповідність між певними високорівневими структурами і їх низькорівневими відповідниками та запропонувати шаблони, за якими студенти зможуть легко орієнтуватися в програмному коді.

Асемблерний код, побудований на основі високорівневих шаблонів як логічно так і за формою відповідає знайомим структурам коду або псевдокоду. Реалізація керуючих структур мовою асемблера не обходиться без оператора безумовного переходу goto (jmp для x86). Але в даному випадку цей оператор використовується для структуризації коду програми, сприяє його чіткій організації, а тому його застосування виправдане.

Враховуючи порівняно невелику кількість типових високорівневих керуючих структур, можна легко створити набір асемблерних шаблонів-відповідників і використовувати їх під час програмування з метою підтримки чіткої логічної структури програми. Один із таких підходів передбачає, що спочатку програма записується у вигляді псевдокоду, а потім рядки псевдокоду замінюються на відповідні рядки асемблерного шаблону [4, 9].

Форматування коду має на меті передачу інформації читачам про структуру коду. Подібно тексту в книгах, який складається з речень, абзаців, глав, розділів програмний текст повинен бути оформленим так, аби можна було легко його читати і легко виявити його логічну структуру.

Прийоми також аналогічні роботі з текстом: групування логічно пов'язаних фрагментів коду в програмні блоки, оформлення блоків коду у вигляді підпрограм, використання заголовків (блоків коментарів), ієрархічна модульна організація програми.

Особливо слід відзначити використання білого простору (символи пропуску, табуляції, порожні рядки). Білий простір дає можливість логічно виділити частини програми, і вишикувати окремі лексичні елементи програми згідно з їх позицією в ієрархії абстракцій коду. Він також дозволяє очам відпочивати між частинами програми.

Коментування коду. З метою покращення читабельності програми та виявлення її структури широко застосовують коментування програмного коду. Відношення науковців до коментарів неоднозначне: «Ніщо не допомагає так, як доречний коментар. Ніщо не захаращує модуль так, як беззмістовні й безапеляційні коментарі» [5, с. 79]. Стиль коментування повинен бути підпорядкованим загальній меті – читабельність коду, легкість його модифікації та підтримки. Серед найважливіших прийомів коментування можна виділити такі [5]: використання коментарів, які підкреслюють структуру програми, випереджаюче коментування структурних блоків коду, коментування алгоритму, а не операторів мови програмування, синхронність коментування й написання програми, пропорційність між кількістю коментарів і розміром коду.

Використання макросів. Одним із потужних засобів полегшення написання і читання асемблерного коду виступають макрокоманди (макроси). Макроси розширюють мову програмування і дають можливість кожному створювати власні зразки високорівневих структур і використовувати їх при програмуванні. Проте, їх не рекомендують використовувати на початкових етапах вивчення мови асемблера. Х. Міллз [10] відзначає, що при компіляції програми з макросами її текст зазнає попередної обробки (виконуються макропідстановки), в результаті чого у програмі з'являються нові фрагменти коду із штучними мітками та новими змінними, що утруднює відлагодження скомпільованого коду.

Висновки. Програмування мовою асемблера викликає значно більше труднощів у студентів при вивченні у порівнянні з мовами високого рівня. Причина полягає не стільки в складності синтаксису асемблера, скільки в неправильному підході до творення програм. Одним із шляхів підвищення ефективності засвоєння мови асемблера є використання студентами принципів структурної організації програмного коду. Програмування на асемблері у структурованому вигляді реалізується через кілька механізмів, головними з яких виступають сегментна організація програм, використання у програмах структурних одиниць, аналогічним мовам високого рівня, модульне програмування, використання підпрограм та бібліотечних функцій. В зв'язку з необхідністю полегшення розуміння вже написаного коду додатково слід використовувати відповідне форматування програмного коду з достатньою кількістю коментарів та білого простору. Подальші дослідження з цього питання повинні бути спрямовані на розробку структурних шаблонів, покликаних значно полегшити конструювання асемблерних програм.

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. Бек Кент. Шаблоны реализации корпоративных приложений / Кент Бек. – М. : ООО «И.Д. Вильямс», 2008. – 176 с.

3. Вирт Н. Алгоритмы + структуры данных = программы / Н. Вирт. – М. : Мир, 1985. – 406 с.

4. Макконнелл С. Совершенный код. Мастер-класс / Пер. с англ. – М. : Издательско-торговый дом «Русская Редакция» ; СПб. : Питер, 2005. – 896 с.

5. Мартин Р. Чистый код: создание, анализ и рефакторинг / Р. Мартин. – СПб. : Питер, 2012. – 464 с.

6. Объектно-ориентированный анализ и проектирование с примерами приложений / Г. Буч, Р.А. Максимчук, М.У. Энг, Б.Дж. Янг, Дж. Коннален, К.А. Хьюстон. – 3-е изд. – М. : ООО «И.Д. Вильямс», 2010. – 720 с.

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

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. Mills H. Structured Programming – Retrospect and Prospect / H. Mills // The H. Mills Collection. – Режим доступу: http://trace.tennessee.edu/utk_harlan/20

11. Porter R. Design Patterns in Learning to Program : A thesis … for the degree of Doctor of Philosophy / R.

Porter. – Adelaide, 2006.

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

13. Wyrd Smythe. CS101: Clarity Trumps Everything. – Режим доступу: http://logosconcarne.com /2011/08/24/cs101-clarity-trumps-everything.

ВІДОМОСТІ ПРО АВТОРА

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

Коло наукових інтересів: моделювання інформаційних систем, проблеми викладання мов програмування.

WOLFRAM|ALPHA: МОЖЛИВОСТІ ЗАСТОСУВАННЯ У НАВЧАННІ ВИЩОЇ МАТЕМАТИКИ СТУДЕНТІВ ЕКОНОМІЧНИХ СПЕЦІАЛЬНОСТЕЙ

Світлана БАС

Використання ІКТ в процесі навчання вищої математики сприяє розвитку творчого мислення студентів, формуванню вмінь та навичок роботи в умовах комп'ютерного середовища, суттєвому підвищенню якості засвоєння навчального матеріалу, створенню та вивченню математичних моделей різноманітних явищ та процесів, демонстрації застосування математичних методів та їх дослідження. У статті розглянуто переваги та недоліки застосування Wolfram|Alpha у формуванні предметної математичної компетентності майбутніх економістів.

Using information technologies in the process of teaching mathematics helps to develop students' creative thinking and better skills of work in a computer environment. It also provides the higher level of getting knowledge, helps to create and study mathematical models of different phenomena; shows how to use and research mathematical methods. The article deals with advantages and disadvantages of using Wolfram|Alpha while forming mathematical competence of future economists.

В умовах сучасної інформатизації суспільства інформаційно-комунікативні технології (ІКТ) та сервіси мережі Інтернет складають невід'ємну частину життя кожної людини. У системі фундаментальної підготовки сучасного економіста основою розв'язання проблеми формування професійних компетентностей та забезпечення професійної мобільності є якісна математична підготовка. З цієї причини необхідна розробка певних методичних підходів до використання засобів ІКТ як для розвитку особистості студента, так і для його підготовки до майбутньої професійної діяльності. Зокрема, для формування вмінь здійснювати прогнозування результатів своєї діяльності, розробки стратегії пошуку шляхів і методів вирішення завдань як навчальних, так і практичних, а в майбутньому професійних. Не менш важливе використання можливостей ІКТ з метою інтенсифікації усіх рівнів навчально-виховного процесу [3].

Розгляду впровадження ІКТ та розробці їх методичного забезпечення присвячені роботи вітчизняних дослідників О. М. Гончарової, В. Б. Григор'євої, О. М. Гудиревої, М. І. Жалдака, В. І. Клочка, М. С. Львова, Н. В. Морзе, С. А. Ракова, Т. Г. Стріжак, Ю. В. Триуса, О. М. Смирнової-Трибульської, М. Б. Ковальчука та ін. В цих роботах основну увагу приділено створенню програмних засобів навчального призначення та методики їх застосування до вивчення різноманітних тем, розробці відповідних комп'ютерно-орієнтованих систем оцінювання роботи студентів в процесі вивчення математики.

До професійних компетентностей економіста належить вміння володіти методами математичного та алгоритмічного моделювання при розв'язуванні прикладних задач. У зв'язку зі скороченням навчальних годин, що відводиться на вивчення математики, підвищенням вимог до рівня математичної підготовки (складати математичні описи економічних процесів), виникає проблема навчити студентів застосовувати математичний апарат до розв'язування прикладних задач економічного змісту. Сформовані таким чином навички моделювання, оцінки, перевірки гіпотез та пошук інформації набувають більшого значення, ніж суто формальне вивчення навчального матеріалу.

Таким чином, використання засобів ІКТ надає можливість навчити студента грамотно формулювати практичну задачу, перекладати цю задачу на мову математики, інтерпретувати результат її розв'язку на мові реальної ситуації, а також перевіряти відповідність отриманих даних та даних досліду. Якщо студент опанує певний математичний пакет, то, використовуючи теоретичну базу, він буде здатен розв'язувати складні задачі, не зважаючи на громіздкі