ВИКОРИСТАННЯ ІНСТРУМЕНТІВ СТАТИЧНОГО ТА ДИНАМІЧНОГО АНАЛІЗУ ПІД ЧАС НАВЧАННЯ ПРОГРАМУВАННЮ МОВОЮ С

І.М. ДАНИЛЮК

Ручне керування пам'яттю є фундаментальною особливістю мови програмування С, яка забезпечує високу ефективність виконання програм, але водночас створює ризики появи помилок, пов'язаних із витоками пам'яті, подвійним звільненням або зверненням до вже звільнених комірок. Такі помилки часто залишаються непоміченими під час компіляції та можуть проявлятися лише під час виконання програми, що ускладнює навчання та перевірку студентських робіт. Тому у процесі підготовки майбутніх фахівців з програмування важливо не лише пояснити принципи динамічного виділення пам'яті, а й сформувати практичні навички аналізу, виявлення та усунення помилок.

У сучасному освітньому процесі навчання мовою С потребує поєднання класичних підходів з інструментами, що використовуються у професійному середовищі. Дослідження показують, що використання статичного аналізу коду дозволяє виявляти значну частину помилок ще до етапу виконання програми [1, 2]. Впровадження таких інструментів у навчальні курси сприяє розвитку у студентів критичного мислення та підвищує якість програмного коду.

Серед інструментів статичного аналізу, які доцільно застосовувати у навчанні, слід відзначити Cppcheck та Clang Static Analyzer. Cppcheck аналізує вихідний код мов С і С++, виявляючи типові помилки, як-от використання неініціалізованих змінних, потенційні витоки пам'яті та порушення меж масивів. Його просто інтегрувати у середовище Visual Studio Code або застосовувати у сценаріях автоматизованої перевірки студентських завдань. Clang Static Analyzer здійснює глибший аналіз шляхів виконання програми, допомагаючи студентам зрозуміти, як неправильна логіка циклів чи умов може призводити до втрати пам'яті [2]. Завдяки використанню таких засобів студенти краще засвоюють принцип «один malloc — один free» і вчаться писати більш передбачуваний та безпечний код [1].

Окрім статичного аналізу, важливим компонентом підготовки програмістів є навчання динамічним методам діагностики. Динамічні аналізатори досліджують поведінку програми під час її виконання, дозволяючи

виявляти помилки керування пам'яттю в реальному часі. Найбільш поширеними інструментами є Valgrind (Memcheck) і AddressSanitizer (ASan). Valgrind виконує програму у спеціальному віртуальному середовищі, фіксуючи всі операції з пам'яттю, що дає змогу точно визначати, які блоки не були звільнені або використані повторно [3]. AddressSanitizer, вбудований у компілятори GCC та Clang, забезпечує ефективне виявлення аналогічних помилок із мінімальними витратами ресурсів та створює детальні звіти з трасуванням помилок [4]. Використання цих засобів у навчальному процесі дозволяє студентам усвідомити механізми роботи оперативної пам'яті та формує навички відповідального підходу до написання коду [1].

Методика впровадження інструментів статичного та динамічного аналізу у навчальний процес передбачає послідовне поєднання теорії і практики. На першому етапі студенти ознайомлюються з принципами виділення і звільнення пам'яті в мові C. Далі демонструються типові помилки, що призводять до витоків, після чого проводяться практичні заняття із застосуванням Cppcheck і Clang Static Analyzer. Наступний етап передбачає роботу з Valgrind та AddressSanitizer, порівняння звітів цих засобів і самостійне усунення виявлених дефектів. Такий підхід сприяє розвитку навичок тестування, відлагодження та самоконтролю коду, що особливо важливо при підготовці студентів до роботи в командних або промислових проєктах.

Використання інструментів статичного та динамічного аналізу під час навчання програмуванню мовою C є ефективним засобом підвищення якості підготовки студентів. Воно формує у них цілісне розуміння взаємозв'язку між архітектурою комп'ютера, управлінням пам'яттю та стабільністю програмного забезпечення. Запропонований підхід доцільно інтегрувати у курси «Програмування» та «Системне програмування», що дозволить поєднати академічну підготовку з вимогами сучасної IT-індустрії.

Література

[1] Seacord R.C. *Effective C: An Introduction to Professional C Programming. 2-nd ed.*. — San Francisco: No Starch Press, 2020. — 272 p.

[2] Bessey A., Block K., Chelf B., Chou A., Fulton B., Hallem S., Henri-Gros C., Kamsky A., McPeak S., Engler D. *A Few Billion Lines of Code Later: Using Static Analysis to Find Bugs in the Real World* // Communications of the ACM. — 2010. — Vol. 53, No 2. — PP. 66-75.

[3] Valgrind: Memcheck Manual [Електронний ресурс]. 2025. Режим доступу: http://valgrind.org/docs/manual/mc-manual.html.

[4] AddressSanitizer [Електронний ресурс]. 2025. Режим доступу: https://clang.llvm.org/docs/AddressSanitizer.html.

ЧНУ ім. Ю. Федьковича, Чернівці, Україна
Email address: i.danyluk@chnu.edu.ua