УДК 681.3.06

# АНАЛІЗ БАЗОВИХ СТРУКТУР ДАНИХ СУЧАСНИХ МОВ ПРОГРАМУВАННЯ

**Щербаков Є.В., Щербакова М.Є.**

# BASIC DATA STRUCTURE ANALYSIS FOR MODERN PROGRAMMING LANGUAGES

**Shcherbakov E.V., Shcherbakova M.E.**

*Розглянуті особливості імперативних, функційних та скриптових мов програмування з точки зору використовуваних ними базових структур даних, таких як структури мови С або списки мови LISP, які відповідають покладеним в основу цих мов теоретичним моделям обчислень: машині Тюринга або лямбда-численню Черча. Проаналізований вплив використовуваних мовами програмування теоретичних моделей та базових структур даних на тривалість розробки та швидкість виконання програм, написаних на цих мовах. З урахуванням результатів цього аналізу окреслена підмножина мов програмування, які найбільш підходять для розробки сучасних додатків для різних обчислювальних пристроїв: С та С++ - для програмування робочих станцій та контролерів; Java, Objective-C та C# - для створення додатків настільних комп'ютерів, планшетів та смартфонів; JavaScript - для розробки Інтернет-базованих додатків для всіх комп'ютерних платформ.*

***Ключові слова:*** *лямбда-числення, машина Тюринга, мова програмування, структура даних, асоціативний список, об'єктно-орієнтоване програмування, додаток.*

**Вступ.** В 1936 році, у відповідь на формально-математичне питання (проблему розв'язності - Entscheidungsproblem), Алонсо Черч розробив лямбда-числення [1] - набір формальних засобів опису функцій і змінних, які призначені для використання в теоретичній кібернетиці при дослідженні визначень функцій, їх застосування та рекурсії. В 1937 році, у відповідь на те ж питання, Алан Тюринг описав формальну мову у вигляді машини зі стрічкою, що містить дані, і головкою, яка може переміщатися уздовж стрічки для читання і запису даних на стрічку [1]. Пізніше було показано, що лямбда-числення Черча та машина Тюринга є еквівалентними - будь-які обчислення, які можуть бути реалізовані за допомогою одного з цих формальних механізмів, можуть бути реалізовані і за допомогою іншого. Ці два підходи до організації обчислень збереглися до теперішнього часу, і конструкції Черча та Тюринга, як і раніше, лежать в основі структурування даних та алгоритмів в програмах. Відмінність між ними полягає в тому, що лямбда-числення відповідає функціональній парадигмі визначення алгоритмів, а машина Тюринга, натомість — імперативній. Тобто, машина Тюринга має певний «стан» — перелік символів, що можуть змінюватись із кожною наступною інструкцією. На відміну від цього, лямбда-числення уникає станів, воно має справу з функціями, котрі отримують значення аргументів та повертають результати обчислень (можливо, інші функції), але не спричиняють зміни вхідних даних.

**Метою роботи є** аналіз особливостей базових структур даних мов програмування, їх вплив на час розробки та швидкість виконання програм, а також визначення на цій основі критеріїв вибору мови програмування при розробці додатків.

**Імперативні мови програмування.** В машині Тюринга пов'язана між собою інформація представляється структурою, для якої виділяється блок стрічки. Наприклад, якщо потрібно зберігати інформацію про людину, перші кілька елементів блоку будуть містити ім'я, декілька наступних елементів - вік і так далі. Майже через століття стрічка Тюринга як і раніше непогано описує комп'ютерну пам'ять і арифметику вказівників мови програмування [2], в якій для доступу до елементів структури використовується адресація база-плюс-зміщення. Використовуючи мову С, можна написати:

```
typedef struct {
char * name;
double age, height;
} person;
person Ivanov = {.name="Іванов", .age=28, .height=173};
```

Тут змінна Ivanov буде вказувати на блок пам'яті, а Ivanov.height буде вказувати на стрічку відразу ж після імені та віку (і після будь-яких додаткових елементів, які включаються з метою вирівнювання на межі слова).

В більш сучасних С-подібних мовах програмування, таких як С++, C# та Java, окрім

засобів процедурного програмування, запозичених практично без істотних змін з мови С, є також засоби об'єктно-орієнтованого програмування на основі класів [2]. Класи в цих мовах введені як природне продовження і розвиток стандартних структур struct мови С, які є типами користувача.

Тому при оголошенні класу створюється новий тип даних, який можна використовувати подібно до вбудованого типу. Проте, на відміну від вбудованих типів, класи містять як дані, так і функції. Клас дозволяє інкапсулювати всі функції та дані, необхідні для управління приватними компонентами програми (наприклад, вікном на екрані; малюнком, побудованим за допомогою графічної програми; пристроєм, підключеним до комп'ютера; завданням, виконуваним операційною системою).

**Функційні мови програмування.** Лямбда-числення покладається на іменовані списки. Наприклад, на лямбда-подібному псевдокоді наведена раніше інформація про людину може бути представлена у формі наступного списку:

```
(Ivanov (
    (name "Іванов")
    (age 28)
    (height 173)
))
```

Haskell, F# та інші LISP-подібні мови [3] базуються на лямбда-численні, так що іменовані списки є природним засобом реалізації структур. Щоб розширити список, потрібно просто додати до нього нові елементи. Як наслідок, функційні мови програмування мають ефективні структури даних високого рівня та прості, але ефективні засоби функційного та об'єктно-орієнтованого програмування.

Функційне програмування є способом створення програм, в яких основною дією є виклик функції, основним способом розбиття програми є створення нового імені функції та завдання для цього імені виразу, який обчислює значення функції, а основним правилом композиції є оператор суперпозиції функцій. Ширша концепція функційного програмування визначає набір спільних правил та тем замість переліку відмінностей від інших парадигм. До важливих концепцій функційного програмування також належать функції вищого порядку та функції першого класу: замикання та рекурсія. До інших поширених можливостей функційних мов програмування належать продовження, система типізації Хіндлі-Мілнера, нечіткі обчислення та монади.

**Скриптові мови програмування.** Сучасні скриптові мови програмування, такі як JavaScript та PHP [4], ввібрали в себе найбільш корисні можливості як імперативних, так і функціональних мов програмування. Базовою структурою даних скриптової мови програмування JavaScript є об'єкт. Будь-яке значення, що не є рядком, числом, true, false, null або undefined, є об'єктом. Подібно функціональним мовам програмування, об'єкт – це асоціативний список, який зіставляє імена властивостей об'єкта їх значенням. Імена властивостей є рядками, тому можна говорити, що об'єкти відображають рядки в значення. Інформація про людину, яка наводилась в якості прикладу раніше при опису імперативних та функціональних мов програмування, на мові JavaScript може представлятися об'єктом - асоціативним списком:

```
var Ivanov = {
    name : "Іванов",
    age : 28,
    height : 173
};
```

Хоча JavaScript-об'єкти є динамічними, тобто властивості об'єктів можуть вільно додаватися і видалятися, вони, однак, можуть використовуватися і для моделювання статичних об'єктів та структур, характерних для імперативних мов програмування. Якщо ігнорувати значення властивостей, об'єкти також можуть бути використані для представлення множин.

Базова структура JavaScript, котра представляє собою асоціативний список, є основою всіх можливостей об'єктно-орієнтованого програмування мови JavaScript. JavaScript використовує модель прототипного спадкування, яке, звісно, відрізняється від спадкування класів в імперативних об'єктно-орієнтованих мовах програмування, таких як C++ або Java. В JavaScript, завдяки тому, що асоціативні списки, які представляють об'єкти, легко зв'язуються один з одним, замість класів використовуються прототипи об'єктів. Нові об'єкти автоматично успадковують методи та атрибути свого батьківського об'єкта через ланцюжок прототипів. Прототип об'єкта можна змінювати в будь-який момент, що робить JavaScript дуже гнучкою, динамічною мовою.

В даний час JavaScript є найбільш широко використовуваною мовою програмування. Майже всі, хто має комп'ютер або смартфон, має всі інструменти, необхідні для виконання вже наявних програм JavaScript та створення нових. Все, що для цього потрібно, - браузер і текстовий редактор.

Створення програми на JavaScript так само просто, як редагування текстового файлу та відкриття його в браузері. Тут немає складних середовищ розробки, які потрібно завантажувати і встановлювати, і не потрібно вивчати складні IDE. Крім того, мова JavaScript проста в освоєнні. Базовий синтаксис відразу знайомий будь-якому програмісту, який вже мав справу хоча б з однією мовою сімейства С. Жодна інша мова не може похвалитися таким низьким бар'єром входження, як JavaScript.

**Аналіз базових структур даних.** Основні відмінності між списковим підходом і використанням блоку пам'яті, а також вплив цих відмінностей на програми наступні:

• Якщо розглядати комп'ютери, то звернення по заданому зміщенню відносно деякої адреси все ще є однією з найбільш швидких операцій, які машина може виконувати. Компілятор С навіть допомагає в цьому, транслюючи мітки в зміщення під час компіляції. І навпаки, для знаходження чогось в списку потрібен пошук: якщо задана мітка "age", який елемент списку їй відповідає і де її дані в пам'яті? Кожна система має свої технічні прийоми, щоб зробити пошук таким швидким, наскільки це можливо, але пошук завжди буде вимагати більше роботи, ніж просте звернення база-плюс-зміщення.

• Додавання нового елемента до списку набагато простіше, ніж процес додавання до зафіксованої під час компіляції структури.

• Компілятор С може виявити під час компіляції, що мітка hieght є друкарською помилкою, тому що він може подивитися в визначення структури і побачити, що немає жодного елементу з таким іменем. Оскільки список є вільно розширюваним, не можна дізнатися, що в ньому немає елемента hieght, доки програма не запрацює і не виявить це, переглядаючи список.

• Позитивною стороною спискового підходу є те, що простим додаванням імені нового елемента досягається повна розширюваність. Негативною стороною цього підходу є те, що, на відміну від структурного підходу, не забезпечується контрольованість, і хоча за допомогою різних хитрощів можна покращити пошук по імені, все це далеко від швидкості звернення база-плюс-зміщення. Багато мов, які базуються на списках, мають систему визначення класів, так що з'являється можливість реєстрації деякого набору елементів списку і на цій основі можна контролювати, чи відповідають нові елементи вже визначеним. Це, якщо зробити все правильно, є гарним компромісом між контрольованістю і легкістю розширення.

Три останні пункти демонструють явне протиріччя: ми хочемо розширюваності, при якій можна легко додавати елементи в структуру; ми також хочемо контрольованості, при якій елементи, які не належать структурі, позначаються як помилки. Тут потрібно знаходити компромісне рішення, і тому контрольоване розширення існуючих списків в різних випадках реалізується по-різному.

Мови програмування С ++, Java, Objective-C, C# та декілька інших мов з традиційними засобами об’єктно-орієнтованого програмування мають засоби для створення нового типу, що розширює наявний тип і наслідує всі елементи старого типу. При цьому швидкість звернень база-плюс-зміщення і виявлення синтаксичних помилок (контрольованість) під час компіляції залишаються такими ж як і для мови С, хоча використання цих мов вимагає від програміста застосування великої кількості синтаксичних конструкцій; там, де мова С для оголошення структур обходиться одним ключовим словом struct та її дуже простими правилами видимості, в мові Java, наприклад, для створення нового типу використовуються конструкції implements, extends, final, instanceof, class, this, interface, private, public та protected.

Якщо повернутися до мови С, то її структури забезпечують найшвидший спосіб доступу до елементів структури, а також їх контрольованість під час компіляції, за що доводиться розплачуватися відсутністю розширюваності під час виконання. Якщо хочеться мати гнучкий список, який може зростати в міру необхідності під час виконання, необхідна спискова структура типу зв'язного списку list або асоціативного списку map з бібліотеки STL.

Що стосується мови програмування JavaScript, то завдяки оперативній компіляції (just-in-time compiling), яка застосовується в сучасних браузерах, більшість коду JavaScript - це відкомпільований, добре оптимізований і виконуваний як машинний код, і тому швидкість виконання близька до програм, написаних на С або С++. Звичайно, є ще накладні витрати, пов'язані з доступом до властивостей об’єктів, із збиранням сміття та динамічним зв'язуванням, так що, природно, можна робити деякі речі швидше; однак, різниця, як правило, не варта того, щоб з нею боротися до тих пір, поки не оптимізовано все інше. З Node.js (високопродуктивне, кероване подіями JavaScript-середовище на стороні сервера, побудоване на базі високо оптимізованого движка V8 JavaScript від Google) додатки JavaScript стають керованими подіями та не блокуючими, що, як правило, нівелює різницю в швидкості виконання коду JavaScript та програм, написаних на менш динамічних мовах програмування.

**Висновки.** У підсумку, якщо користуватися критерієм найбільшої кількості розроблюваних додатків в даний час, з усього різноманіття мов програмування можна виділити наступні: мови програмування С та С++ як найбільш ефективні по швидкості виконання програм використовуються для розробки системних програм та програмного забезпечення робочих станцій і контролерів систем управління, які працюють в режимі реального виміру часу; об'єктно-орієнтована мова програмування C# та мова розмітки візуальних інтерфейсів XAML - для розробки настільних та мобільних додатків для платформи .NET операційної системи Windows; об'єктно-орієнтована мова програмування Objective-C і мова розмітки XML (для зберігання та налаштування візуальних інтерфейсів) - для розробки мобільних додатків на базі операційної системи iOS [5]; об'єктно-орієнтована мова програмування Java та мова розмітки на базі XML - для розробки додатків для операційної системи Android [6]; скриптова мова

програмування JavaScript, мова розмітки веб-сторінок HTML та мова каскадних таблиць стилів CSS - для розробки HTML-сторінок мережі Інтернет та Інтернет-базованих додатків практично для всіх комп'ютерних платформ.

**Література**


1. Клини С. К. Математическая логика : пер. с англ. / Клини С. К. – М. : Мир, 1973. – 480 с.
2. Щербаков Є. В. Мовні засоби системного програмування: Навчальний посібник / Щербаков Є.В., Щербакова М.Є. – Луганськ : Вид-во СНУ ім. В. Даля, 2006. – 376 с.
3. Хендерсон П. Функциональное программирование. Применение и реализация : пер. с англ. / Питер Хендерсон. – М. : Мир, 1983. - 349 с.
4. Connolly R. Fundamentals of Web Development / Randy Connolly, Ricardo Hoar. - New Jersey : Pearson Education, Inc., 2015. – 1024 p.
5. Neuburg M. Programming iOS 7. Fourth Edition / Matt Neuburg . - Sebastopol : O'Reilly Media, Inc., 2014. – 929 p.
6. Gargenta M. Learning Android, Second Edition / Marko Gargenta, Masumi Nakamura. – Sebastopol : O'Reilly Media, Inc., 2014. – 288 p.


**References**


1. Stephen Cole Kleene. Mathematical Logic – John Wiley & Sons, Inc., New York, 1967. – 479 p.
2. Shcherbakov E. V. Language means of system programming: Study Guide / Shcherbakov E. V., Shcherbakova M. E. – Publisher: EUNU named after V. Dahl, Luhansk, 2006. – 376 p.
3. Henderson P. Functional Programming. Application and Implementation Hardcover / Peter Henderson. – Prentice Hall, 1980. - 350 p.
4. Connolly R. Fundamentals of Web Development / Randy Connolly, Ricardo Hoar. - New Jersey : Pearson Education, Inc., 2015. – 1024 p.
5. Neuburg M. Programming iOS 7. Fourth Edition / Matt Neuburg . - Sebastopol : O'Reilly Media, Inc., 2014. – 929 p.
6. Gargenta M. Learning Android, Second Edition / Marko Gargenta, Masumi Nakamura. – Sebastopol : O'Reilly Media, Inc., 2014. – 288 p.



**Щербаков Е. В., Щербакова М. Е. Анализ базовых структур данных современных языков программирования**

*Рассмотрены особенности императивных, функциональных и скриптовых языков программирования с точки зрения используемых ими базовых структур данных, таких как структуры языка С или списки языка LISP, которые соответствуют положенным в основу этих языков теоретическим моделям вычислений: машине Тьюринга или лямбда-исчислению Черча. Проанализировано влияние используемых языками программирования теоретических моделей и базовых структур данных на продолжительность разработки и скорость выполнения программ, написанных на этих языках. С учетом результатов этого анализа очерчено подмножество языков программирования, которые наиболее подходят для разработки современных приложений для различных вычислительных устройств: С и С++ - для программирования рабочих станций и контроллеров; Java, Objective-C и C# - для создания приложений настольных компьютеров, планшетов и смартфонов; JavaScript - для разработки Интернет-базированных приложений для всех компьютерных платформ.*



***Ключевые слова**: лямбда-исчисление, машина Тьюринга, язык программирования, структура данных, ассоциативный список, объектно-ориентированное программирование, приложение.*



**Shcherbakov E. V., Shcherbakova M. E. Basic data structure analysis for modern programing languages**

*Considered the features of imperative, functional, and script programming languages in terms of their underlying data structures, such as the C structure or LISP lists that fit behind the theoretical models of computation: Turing machine or the lambda calculus Church. Analyzed the influence of the programming language used theoretical models and underlying data structures for the duration of the development and execution speed of programs written in these languages. Taking into account the results of this analysis outlines a subset of programming languages that are most suitable for the development of advanced applications for various computing devices: C and C++ - programming workstations and controllers; Objective-C, Java, and C# - to create applications for desktops, tablets and smartphones; JavaScript - for the development of Internet-based applications for all computer platforms.*



***Keywords:** lambda calculus, Turing machine, programming language, data structure, an association list, object-oriented programming, application.*



**Щербаков Євген Васильович –** к.т.н., доцент, доцент кафедри комп'ютерної інженерії, Технологічний інститут Східноукраїнського національного університету імені Володимира Даля (м. Сєвєродонецьк), gkvarc@gmail.com

**Щербакова Марина Євгенівна –** к.т.н., доцент, доцент кафедри комп'ютерної інженерії, Технологічний інститут Східноукраїнського національного університету імені Володимира Даля (м. Сєвєродонецьк), ms432@mail.ru



*Рецензент:* **Суворін О. В.** – д.т.н., доцент

Стаття подана 27.11.2014