LEARN.
ADACORE.COM

Learn Ada (Complete)

Release 2025-05

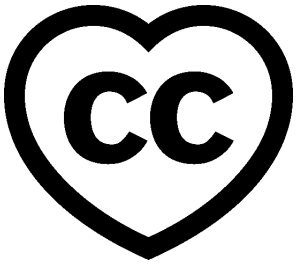
AdaCore

трав. 19, 2025

Зміст:

РОЗДІЛ 1

Керівництво для професійних програмістів



Цей посібник пропонує комп'ютерним фахівцям конкретні стилістичні рекомендації для послідовного використання можливостей мови Ada, щоб створювати якісні, читабельні та зрозумілі програми. Книга охоплює форматування коду, структуру програми, практики програмування, паралельність, портативність, повторне використання та об'єктно-орієнтовані функції Ada.

1.1 Передмова

Цей документ[2] є оновленням *Посібника з якості та стилю Ada 95*[3], що відображає останню версію мови Ada[4], яку зазвичай називають Ada 2012[5]. Мета цього посібника - допомогти комп'ютерним фахівцям створювати кращі програми на Ada шляхом визначення набору стилістичних рекомендацій, які безпосередньо впливатимуть

[1] http://creativecommons.org/licenses/by-sa/4.0
[2] https://en.wikipedia.org/wiki/Programming_style
[3] http://www.adaic.org/docs/95style/html/cover.html
[4] https://en.wikipedia.org/wiki/Ada_(programming_language)
[5] https://en.wikibooks.org/wiki/Ada_Programming/Ada_2012

на якість їхніх програм на Ada. Цей посібник зі стилю не призначено для заміни *Довідника з Ади посібником*[6] або *Обґрунтування*[7], а також він не є навчальним посібником з мови програмування Ada[8].

Посібник зі стилю поділено на розділи, які відповідають основним рішенням, котрі приймає кожен програміст, створюючи високоякісне, надійне, багаторазове та портативне програмне забезпечення на Ada. Розділи дещо перетинаються, оскільки не всі програмні рішення можуть бути прийняті незалежно. Окремі розділи присвячені представленню вихідного коду, читабельності, структурі програми, практиці програмування, паралелізму, портативності, повторному використанню, продуктивності, а також нова глава присвячена об'єктно-орієнтованим функціям.

Кожна глава розділена на керівні принципи (рекомендації), формат яких є досить гнучким у використанні, оскільки їхній зміст є одночасно і настановчим, і адаптивним. Кожна настанова складається зі стислого викладу принципів, яких слід дотримуватися, та обґрунтування, що пояснює, чому ця настанова є важливою. Настанови також містять приклади використання, а також можливі винятки із застосування настанов. Багато настанов є достатньо конкретними, щоб їх можна було прийняти як корпоративні або проектні стандарти програмування. Інші потребують управлінського рішення щодо конкретного застосування, перш ніж їх можна буде використовувати як стандарти. У таких випадках у прикладах наведено зразок конкретного варіанту, який використовується у всіх настановах.

1.1.1 Передісторія

Офіс об'єднаної програми Ada (AJPO) профінансував посібник зі стилю Ada 95, який було створено шляхом об'єднання набору настанов з використання Ada 95[9] з модифікаціями оригінальної *настанови з якості та стилю Ada: Настанови для професійних програмістів*[10], версія 02.01.01 (AQ&S 83) (Software Productivity Consortium 1992), розроблених для підтримки Ada 83[11]. Настанови Ada 95 ґрунтуються на великій кількості даних, доступних з проекту Ada 9X, бібліотеки AJPO та спільноти Ada в цілому. Авторами оновлення є технічний персонал Консорціуму продуктивності програмного забезпечення (Software Productivity Consortium), а Агентство перспективних дослідницьких проектів[12] (ARPA) брало участь у роботі над оновленням.

Попередній AQ&S 83 містив набір настанов, які допомагали програмісту дисципліновано використовувати можливості Ади. У 1992 році Консорціум завершив оновлення версії 2.1 посібника зі стилю за контрактом з AJPO. AJPO назвала цей посібник зі стилю "рекомендованим посібником зі стилю для всіх програм Міністерства оборони".

Наразі не існує офіційного оновлення посібника зі стилів для мовної версії Ada 2005 або Ada 2012. Під час тривалого обговорення на конференції SIGAda 2008[13] посібника зі стилю у зв'язку з оновленням Ади 2005 і його постійною актуальністю, кілька учасників зголосилися викласти посібник у вікіпедію[14], щоб заохотити спільне оновлення посібника, передавши його в руки фахівців-практиків мови. Частково це стало можливим за умовами безоплатної, всесвітньої, невиключної, безвідкличної ліцензії на необмежене використання матеріалів посібників зі стилю Ada 83 і 95, наданої Об'єднаним програмним офісом DoD з питань Ada, що діє у всьому світі.

[6] http://www.ada-auth.org/standards/ada12.html
[7] http://www.ada-auth.org/standards/rationale12.html
[8] https://en.wikibooks.org/wiki/Ada_Programming
[9] https://en.wikibooks.org/wiki/Ada_Programming/Ada_95
[10] http://archive.adaic.com/docs/style-guide/83style/html/
[11] https://en.wikibooks.org/wiki/Ada_Programming/Ada_83
[12] https://en.wikipedia.org/wiki/Advanced_Research_Projects_Agency
[13] http://www.sigada.org/conf/sigada2008/
[14] https://en.wikibooks.org/wiki/Main_Page

1.2 Вступ

Стиль - це часто недооцінений, але дуже важливий атрибут для написання чого завгодно. Стиль безпосередньо впливає на читабельність і зрозумілість кінцевого продукту. Стиль програмування, як написання коду комп'ютерною мовою, також страждає від нехтування цим атрибутом. Програми повинні бути читабельними і зрозумілими для людини, а не тільки для машинного виконання. Ця вимога є важливою для створення якісних продуктів, які не тільки задовольняють потреби користувачів, але й можуть розроблятись за передбачуваним бюджетом і графіком. Ця книга покликана допомогти фахівцям створювати кращі програми на мові Ada. Вона представляє набір конкретних стилістичних рекомендацій для дисциплінованого використання потужних можливостей мови Ada 95 (Ada Reference Manual 1995).

Кожна настанова складається зі стислого викладу принципів, яких слід дотримуватися, та обґрунтування необхідності кожної настанови. У більшості випадків наводиться приклад застосування вказівки, а в деяких випадках також додається приклад з можливими наслідками порушення вказівки. Окремо зазначено можливі винятки із застосування вказівки, а також додаткові пояснення, там де це доречно. Деякі вказівки надають більше конкретизації, тож вони можуть бути застосовані в якості стандарту. А у примітках щодо автоматизації обговорюється, як можна автоматизувати виконання вказівки.

Ada була спроектована для розробки високоякісного, надійного, багаторазового та переносного програмного забезпечення. З ряду причин жодна мова програмування не може забезпечити досягнення цих бажаних цілей самостійно. Наприклад, програмування повинно бути вбудоване в дисциплінований процес розробки, що включає в себе аналіз вимог, проектування, реалізацію, верифікацію, валідацію та супровід в організований спосіб. Використання мови має відповідати належній практиці програмування, що базується на усталених принципах програмної інженерії. Ця книга покликана допомогти подолати розрив між теоритичними принципами та реальною практикою програмування мовою Ada.

Багато настанов у цій книзі розроблено для того, щоб сприяти створенню зрозумілого вихідного тексту. Мета цих настанов - полегшити розвиток, адаптацію та супровід програм. Зрозумілий вихідний текст з більшою ймовірністю буде правильним і надійним. Легка адаптація вимагає глибокого розуміння програмного забезпечення; цьому значно сприяє ясність. Ефективна адаптація коду є передумовою для повторного використання коду - підходу, який має потенціал для значного зниження вартості розробки системи. Нарешті, оскільки підтримка програмного забезпечення (насправді еволюція) є дорогим процесом, який триває протягом усього життя системи, ясність відіграє одну з ключових ролей у зниженні витрат. Протягом усього життєвого циклу продукту код доводиться читати і розуміти набагато частіше, ніж писати; отже, інвестиції в написання читабельного, зрозумілого коду завжди виправдовують себе.

Решта розділів цього вступу обговорюють організацію цієї книги і те, як представлений матеріал може бути використаний людьми з різних ролями в комнаді, включаючи програмістів-початківців, досвідчених програмістів на Ada, об'єктно-орієнтованих програмістів, менеджерів програмних проектів, підрядних організацій, організацій, що встановлюють стандарти, і планувальників переходу на Ada 95 з існуючих програм на Ada 83 (Ada Reference Manual 1983 року).

1.2.1 Організація цієї книги

Формат цієї книжки відповідає добре відомому формату настанов з якості та стилю Ada: Керівництво для професійних програмістів, версія 02.01.2001 (AQ&S 83) (Software Productivity Consortium 1992). Посібник зі стилю поділено на розділи, які відповідають основним рішенням, які доводиться обирати кожному програмісту при створенні якісного, надійного, багаторазового та переносного програмного забезпечення на мові Ada. Розділи дещо перетинаються, оскільки не всі програмні рішення можуть бути прийняті незалежно.

Окремі розділи акцентують увагу на кінцевому вигляді коду, читабельності, структурі програми, практиці програмування, паралелізації, портативність, повторному використанню та ефективності виконання, а новий розділ присвячена об'єктно-орієнтованим функціям. Кожний розділ закінчується стислим списком настанов, що містяться у ній. Останній розділ показує повну реалізацію прикладу Dining Philosophers, наданого доктором Майклом Б. Фельдманом та паном Бйорном Каллбергом. При створенні цього прикладу було використано багато рекомендацій з цієї книги. У додатку наведено матрицю перехресних посилань між розділами довідника Ada Reference Manual (1995) та рекомендаціями цього посібника зі стилів.

Ця книга написана з використанням загального словника програмної інженерії, розробленого за останні 20 років. Інженерія програмного забезпечення - це дисципліна, що швидко розвивається, з відносно новими поняттями та термінологією. Однак, щоб встановити загальну систему координат, необхідні визначення взяті з Ada Reference Manual (1995) та Rationale (1995).

У книзі є посилання на інші джерела інформації про стиль написання на мові Ada та інші питання, пов'язані з цим. Список використаних джерел наведено наприкінці книги. Також подано бібліографію.

У цій книзі термін "Ada" відноситься до останнього стандарту Ada, випущеного в лютому 1995 року (іноді також відомий як Ada 95). Посилання на більш ранній стандарт Ada чітко позначаються як "Ada 83".

Представлення та читабельність вихідного коду

Розділи 2 і 3 порушують питання створення чіткого, читабельного і зрозумілого вихідного тексту. Розділ 2 присвячений форматуванню коду, а Розділ 3 розглядає питання використання коментарів, угод про імена та типи.

Існує два основних аспекти зрозумілого коду: (1) ретельне і послідовне розташування вихідного тексту на сторінці або екрані, про що йдеться у Розділі 2, що може значно покращити його читабельність; (2) ретельна увага до структури коду, про що йдеться у Розділі 3, що може зробити код легшим для розуміння. Це стосується як дрібних деталей (наприклад, ретельний вибір імен ідентифікаторів або дисциплінованe використання циклів), так і великих (наприклад, правильне використання пакетів). Ці настанови стосуються як макета, так і структури.

Уподобання щодо форматування коду та іменування, як правило, є дуже особистими. Ви повинні збалансувати свої особисті уподобання з думкою інших інженерів у проекті, щоб узгодити єдиний набір правил, яких дотримуватиметься вся команда проекту, а автоматичні форматори коду можуть допомогти вам з цим.

Структура програми

Розділ 4 присвячений загальній структурі програми. Правильна структура покращує зрозумілість програми. Це аналогічно читабельності на нижчих рівнях і включає питання структури високого рівня, зокрема використання пакетів та дочірніх пакетів, видимість та винятки. Більшість настанов у цьому розділі стосуються застосування правильних принципів програмної інженерії, таких як приховування інформації, абстрагування, інкапсуляція та розділення завдань.

Практики програмування

Розділ 5 містить настанови, які визначають послідовне та логічне використання можливостей мови. Ці настанови стосуються необов'язкових частин синтаксису, типів, структур даних, виразів, операторів, видимості, винятків та помилкового виконання.

Паралельність

Розділ 6 визначає правильне використання паралелізму для розробки передбачуваного, надійного, повторно використовуваного та переносного програмного забезпечення. Розглядаються такі теми, як постановка задач, захищені модулі, комунікація та завершення роботи. Однією з основних сфер вдосконалення мови Ada була краща підтримка спільного використання даних. Механізм задач був єдиним доступним підходом до захисту спільних даних. Настанови у цьому розділі підтримують використання захищених типів для інкапсуляції та синхронізації доступу до спільних даних.

Переносивність і можливість багаторазового використання

Розділи 7 і 8 розглядають питання проектування з урахуванням змін з дещо інших точок зору. Розділ 7 присвячений основам переносимості, простоті зміни програмного забезпечення з однієї комп'ютерної системи або середовища на іншу, а також впливу використання певних функцій на переносимість. Розділ 8 присвячен повторному використанню коду, тобто ступеню, до якого код може бути використаний у різних додатках з мінімальними змінами.

Настанови щодо перенесення, розглянуті в Розділі 7, потребують ретельної уваги. Дотримання їх є важливим, навіть якщо потреба у перенесенні отриманого програмного забезпечення наразі не передбачається. Дотримання настанов покращує потенційну можливість повторного використання отриманого коду у проектах, які використовують різні реалізації Ada. Вам слід наполягати на тому, щоб у випадках, коли потреби конкретного проекту змушують послабити деякі з настанов щодо перенесення, непереносимі особливості вихідного тексту були чітко вказані.

Настанови щодо повторного використання, наведені у Розділі 8, ґрунтуються на принципах інкапсуляції та проектування для змін. Ці настанови підкреслюють, що розуміння і ясність, надійність, адаптивність і незалежність є корисними і бажаними навіть тоді, коли повторне використання не передбачається, оскільки отриманий код є більш стійким до запланованих і незапланованих змін.

Об'єктно-орієнтовані функції

Розділ 9 визначає набір настанов у загальних об'єктно-орієнтованих термінах, які використовують деякі можливості Ada 95, яких немає в Ada 83. У настановах обговорюється використання нових можливостей Ada, таких як розширення типів (мічені типи), абстрактні мічені типи та абстрактні підпрограми для реалізації одиночного успадкування, множинного успадкування та поліморфізму.

Продуктивність

Розділ 10 визначає набір керівних принципів, спрямованих на підвищення продуктивності. Визнано, що деякі підходи до продуктивності суперечать принципам ремонтопридатності та переносивності. Більшість настанов у цьому розділі починаються словами "... якщо виміряні показники продуктивності вказують". "Вказують" означає, що ви визначили, що вигода від підвищення продуктивності вашої програми у вашому середовищі переважає негативні побічні ефекти на зрозумілість, супроводжуваність та переносимість отриманого коду.

1.2.2 Як користуватися цією книгою

Ця книга призначена для тих, хто займається безпосередньо розробкою програмних систем, написаних на мові Ada. У наступних розділах обговорюється, як найефективніше використати поданий матеріал. Читачі з різним рівнем досвіду роботи з Адою або різними ролями в програмному проекті будуть використовувати книгу по-різному.

Цю книгу можна використовувати по-різному: як посібник з гарного стилю Ada; як вичерпний перелік настанов, що сприятимуть створенню кращих програм Ada; або як довідник, де можна знайти приклади використання та обговорити специфічні особливості цієї мови. Книга містить багато настанов, деякі з яких є досить складними. Вивчати їх усі одночасно не обов'язково; навряд чи ви будете використовувати всі можливості мови одночасно. Однак, рекомендується, щоб усі програмісти (і, за можливості, інші співробітники проекту Ada) доклали зусиль, щоб прочитати і зрозуміти Розділи 2, 3, 4 і Розділ 5 аж до Розділу 5.7. Деякий матеріал є досить складним (наприклад, Розділ 4.2, де обговорюється видимість), але він охоплює питання, які є фундаментальними для ефективного використання Ada і є важливими для будь-якого програміста, що займається створенням систем на Ada.

Ця книга не є вступним текстом про Ada або повним посібником з мови Ada. Передбачається, що ви вже знаєте синтаксис Ada і маєте елементарне розуміння семантики. На цьому тлі настанови будуть для вас корисними, інформативними, а часто й повчальними.

Якщо ви вивчаєте мову Ada, вам слід підготувати всебічний вступ до неї. Два хороших вступних тексти про Ada 83 належать Барнсу (Barnes, 1989) та Коену (Cohen, 1986). Обидва автори опублікували нові книги, які охоплюють Ada 95 (Barnes 1996, Cohen 1996). Ознайомившись з цими текстами, ми рекомендуємо використовувати їх разом з Rationale (1995). Довідковий посібник Ada (1995) слід розглядати як доповнення до цих книг. Більшість настанов посилаються на розділи довідника Ada (1995), які визначають особливості мови, що обговорюються. Додаток A містить перехресні посилання на розділи Довідника з мови Ada в настановах.

Новому програмісту Ada

На перший погляд, Ada пропонує вражаючу різноманітність функцій. Це потужний інструмент, призначений для вирішення складних завдань, і майже кожна функція має законне застосування в певному контексті. Це робить особливо важливим дисципліноване та організоване використання можливостей Ada. Дотримання цих вказівок може полегшити вивчення Ada і допомогти вам опанувати її позірну складність. З самого початку ви зможете писати програми, які використовують найкращі можливості мови так, як було задумано розробниками.

Програмісти, які мають досвід роботи з іншими мовами програмування, часто піддаються спокусі використовувати Ada так, ніби це їхня звична мова, але з дратівливими синтаксичними відмінностями. Цієї пастки слід уникати будь-якою ціною; вона може призвести до заплутаного коду, який підриває саме ті аспекти Ada, які роблять її такою придатною для побудови високоякісних систем. Ви повинні навчитися "думати Адою". Дотримання рекомендацій цієї книги та ознайомлення з прикладами їх використання допоможе вам зробити це якомога швидше і безболісніше.

Певною мірою програмісти-початківці, які вивчають Ada, мають перевагу. Дотримання рекомендацій з самого початку допомагає виробити чіткий стиль програмування, який ефективно використовує мову. Якщо ви належите до цієї категорії, рекомендуємо вам прийняти ці вказівки для тих вправ, які ви виконуєте в рамках вивчення Ada. На початковому етапі розвитку навичок програмування, зосереджуючись на самих настановах і прикладах, що їх підтримують, важливіше, ніж розуміння обґрунтування кожної настанови.

Обґрунтування багатьох настанов допомагає досвідченим програмістам зрозуміти і прийняти пропозиції, представлені в настановах. Деякі з настанов також написані для досвідчених програмістів, яким доводиться йти на інженерні компроміси. Це особливо актуально у сферах перенесення, багаторазового використання та продуктивності. Ці настанови та обґрунтування допоможуть вам усвідомити проблеми, що впливають на кожне програмне рішення. Потім ви зможете використовувати цю обізнаність для розпізнавання інженерних компромісів, які вам зрештою доведеться робити, коли ви станете досвідченим програмістом Ada.

Досвідченому програмісту Ada

Як досвідчений програміст Ada, ви вже пишете код, який відповідає багатьом настановам цієї книги. Однак у деяких сферах ви, можливо, прийняли особистий стиль програмування, який відрізняється від представленого тут, і, можливо, не бажаєте його змінювати. Уважно перегляньте ті настанови, які не відповідають вашому поточному стилю, переконайтеся, що ви розумієте їх обґрунтування, і розгляньте можливість їх прийняття. Загальний набір рекомендацій у цій книзі втілює послідовний підхід до створення високоякісних програм, який був би ослаблений надто великою кількістю винятків.

Узгодженість - ще одна важлива причина для загального прийняття єдиних настанов. Якщо всі співробітники проекту пишуть вихідний текст в одному стилі, то багато критично важливих проектних дій стають простішими. Узгоджений код спрощує формальний і неформальний перегляд коду, системну інтеграцію, повторне використання коду в рамках проекту, а також надання і застосування допоміжних інструментів. На практиці корпоративні або проектні стандарти можуть вимагати, щоб відхилення від настанов були чітко прокоментовані, тому застосування нестандартного підходу може вимагати додаткової роботи.

Деякі настанови в цій книзі, зокрема в розділах про паралелізм, переносивність, повторне використання, об'єктно-орієнтовані можливості та продуктивність, зосереджені на компромісах у проектуванні. Ці настанови просять вас над тим,

чи використання можливостей Ada є правильним проектним рішенням для вашого додатку. Часто існує декілька способів реалізації певного проектного рішення, і ці настанови обговорюють компроміси, які ви повинні враховувати при прийнятті рішення.

Досвідченим об'єктно-орієнтованим програмістам

Як досвідчений об'єктно-орієнтований програміст, ви оціните зусилля, які було докладено для елегантного розширення мови Ada, щоб включити в неї потужні об'єктно-орієнтовані можливості. Ці нові можливості тісно інтегровані з існуючими функціями та словником мови. Ця книга навмисно написана таким чином, щоб надати погляд з точки стилю, тому об'єктно-орієнтовані можливості Ada використовуються в ній повсюдно. Дисципліноване використання цих можливостей сприятиме створенню програм, які легше читати і модифікувати. Ці можливості також дають вам гнучкість у створенні компонентів, які можна використовувати повторно. Розділ 9 присвячено об'єктно-орієнтованому програмуванню та питанням успадкування і поліморфізму. Попередні розділи містять перехресні посилання на рекомендації Розділу 9.

Вам буде легше скористатися багатьма концепціями з Розділу 9, якщо ви вже виконали об'єктно-орієнтоване проектування. Результатом об'єктно-орієнтованого проектування буде набір змістовних абстракцій та ієрархія класів. Абстракції повинні включати визначення об'єктів проектування, включаючи структуру і стан, операції над об'єктами і передбачувану інкапсуляцію для кожного об'єкта. Деталі проектування цих абстракцій та ієрархії класів виходять за рамки цієї книги. Існує низка хороших джерел, зокрема Rumbaugh та ін. (1991), Jacobson та ін. (1992), ADARTS Guidebook (Software Productivity Consortium 1993) та Booch (1994).

Керівнику проекту з розробки програмного забезпечення

Технічний менеджмент відіграє ключову роль у забезпеченні того, щоб програмне забезпечення, створене в ході проекту, було правильним, надійним, підтримуваним та переносним. Керівництво повинно створити в рамках всього проекту прихильність до створення високоякісного коду; визначити стандарти кодування та настанови для конкретного проекту; сприяти розумінню того, чому однакове дотримання обраних стандартів кодування є критично важливим для якості продукту; а також розробити політику та процедури для перевірки та забезпечення дотримання цих стандартів. Настанови, що містяться в цій книжці, можуть допомогти в цих зусиллях.

Важливою діяльністю для менеджерів є визначення стандартів кодування для проекту або організації. Ці настанови самі по собі не повним набором стандартів, проте вони можуть слугувати основою для стандартів. Деякі настанови вказують на діапазон рішень, але не прописують конкретного рішення. Наприклад, друга в книзі (Настанова 2.1.2) рекомендує використовувати однакову кількість пробілів для відступів і вказує в обґрунтуванні, що розумним буде від двох до чотирьох пробілів. Разом зі старшим технічним персоналом ви повинні переглянути кожну таку настанову і прийняти рішення щодо її застосування, яке стане стандартом вашого проекту або організації.

Дві інші сфери потребують управлінських рішень щодо стандартизації. Настанова 3.1.4 радить уникати довільних скорочень у назвах об'єктів або підрозділів. Ви повинні підготувати глосарій прийнятних скорочень для проекту, який дозволяє використовувати коротші версії специфічних термінів (наприклад, ШПФ - швидке перетворення Фур'є або SPN - стохастична мережа Петрі). Цей глосарій має бути коротким і обмежуватися лише тими термінами, які потрібно часто використовувати як частину назв. Необхідність постійно звертатися до великого глосарію для розуміння вихідного тексту ускладнює його читання.

Настанови щодо перенесення, наведені в Розділі 7, потребують ретельної уваги. Дотримання їх є важливим, навіть якщо потреба у перенесенні отриманого програмного забезпечення наразі не передбачається. Дотримання настанов покращує потенційну можливість повторного використання отриманого коду у проектах, які використовують різні реалізації Ada. Ви повинні наполягати на тому, щоб у випадках, коли потреби конкретного проекту змушують послабити деякі з настанов щодо перенесення, непереносимі особливості вихідного тексту були чітко вказані. Дотримання настанов Розділу 7 вимагає визначення та стандартизації специфічних для проекту або організації числових типів, які використовуватимуться замість (потенційно непереносимих) попередньо визначених числових типів.

Ваші рішення щодо питань стандартизації повинні бути включені в документ зі стандартів кодування проекту або організації. Після створення стандартів кодування вам потрібно забезпечити їх дотримання. Дуже важливо заручитися щирою прихильністю ваших програмістів до використання стандартів. Маючи таку прихильність і приклад високоякісної Ada, яку створюють ваші програмісти, буде набагато легше проводити ефективні формальні огляди коду, які перевіряють дотримання стандартів проекту.

Деякі загальні питання, що стосуються управління проектами Ada, обговорюються в роботі Hefley та ін. (1992).

Підрядним організаціям та організаціям зі стандартизації

Багато з настанов, представлених тут, є достатньо конкретними, щоб їх можна було прийняти як корпоративні або проектні стандарти програмування. Інші потребують управлінського рішення щодо конкретного застосування, перш ніж їх можна буде використовувати як стандарти. У таких випадках у прикладах представлено зразок конкретної реалізації, який використовується у всіх прикладах. Такі приклади слід визнати слабшими рекомендаціями, ніж самі настанови. У деяких випадках, коли приклади взяті з опублікованих робіт, авторський стиль використано без змін.

Інші рекомендації, представлені в цій книзі, навмисно сформульовані в термінах вибору дизайну для розгляду. Ці настанови не можуть бути сприйняті як непорушні правила, яких проєкт зобов'язаний дотримуватися. Наприклад, не слід тлумачити настанови 6.1.1 та 6.1.2 як такі, що забороняють використовувати задач у проекті. Скоріше, ці настанови мають на меті допомогти проектувальнику знайти компроміс між використанням захищених об'єктів і задач, таким чином, щоб він міг зробити більш усвідомлений вибір між цими можливостями.

Настанови, наведені в цьому документі, не призначені для використання в якості стандарту. Часом неясно, чи можна змусити дотримуватися настанови, бо вона лише покликана ознайомити інженера з можливими варіантами та їхніми наслідками. В інших випадках все ще залишається вибір щодо настанови, наприклад, скільки пробілів використовувати для кожного рівня відступу.

Якщо настанова є надто загальною для того, щоб навести приклад, у розділі "Конкретизація" кожної настанови містяться більш конкретні вказівки. Ці конкретизації можна вважати стандартом, і вони мають більше шансів бути застосованими. Будь-яка організація, яка намагається витягти стандарти з цього документа, повинна оцінити весь контекст. Кожна настанова працює найкраще, коли застосовуються пов'язані з нею настанови. Ізольовано настанова може бути малоефективною або взагалі не мати жодної користі.

Планувальники переходу від Ada 83 до Ada 95

Проблеми переходу поділяються на дві основні категорії: несумісність між мовами, зокрема, висхідна сумісність, та використання нових мовних особливостей.

Сумісність Ada 95 з попередником була основною метою розробки мови. На практиці можливі лише незначні несумісності між Ada 83 і Ada 95, які легко піддаються вирішенню. (див. Обґрунтування Ada 95 [1995], Додаток X під назвою "Висхідна сумісність"). Детальну інформацію про проблеми сумісності можна знайти в Taylor (1995) та Intermetrics (1995).

Планувальник переходу може отримати з цієї книги інформацію про використання мовних особливостей двома способами. По-перше, у таблиці 1 показано вплив нових можливостей мови Ada 95 на розділи посібника зі стилів. По-друге, у Додатку A зіставлено розділи Довідкового посібника з Ada (1995) з конкретними щодо стилю.

1.2.3 Типографічні конвенції

У цьому посібнику зі стилю використано такі типографічні конвенції:

- Шрифт із зарубками Загальне представлення інформації.
- *Курсивний шрифт із зарубками* Назви публікацій та виділення.
- **Жирний шрифт із зарубками** Заголовки розділів.
- **Жирний шрифт без засічок** Підзаголовки для вказівок, конкретизації, прикладів, обґрунтувань, приміток, винятків, приміток щодо автоматизації, застережень та підзаголовків у розділах резюме.
- `Шрифт друкарської машинки`:

```
Синтаксис коду.
```

1.3 Презентація вихідного коду

Фізичне розташування вихідного тексту на сторінці або екрані має значний вплив на його читабельність. Цей розділ містить настанови щодо представлення вихідного коду, призначені для того, щоб зробити його більш читабельним.

На додаток до загальних рекомендацій, у розділах, присвячених конкретним випадкам, подано конкретні рекомендації. Якщо ви не згодні з конкретними рекомендаціями, ви можете прийняти свій власний набір правил, які все ще відповідають загальним рекомендаціям. Перш за все, будьте послідовними у всьому вашому проекті.

Повністю узгодженої верстки важко досягти або перевірити вручну. Тому ви можете автоматизувати верстку за допомогою інструменту для параметризованого форматування коду або включити настанови в шаблон автоматичного кодування. Деякі з настанов і конкретних рекомендацій, представлених у цій главі, не можуть бути застосовані за допомогою інструменту форматування, оскільки вони базуються на семантиці а не синтаксисі коду Ada. Більш детальну інформацію наведено у розділах "примітки щодо автоматизації".

1.3.1 Форматування коду

"Форматування коду" вихідного коду Ada впливає на те, як код виглядає, а не на те, що він робить. Сюди входять такі теми, як горизонтальний інтервал, відступи, вирівнювання, нумерація сторінок і довжина рядків. Найважливішою настановою є послідовність як в межах одиниці компіляції, так і в усьому проєкті.

Горизонтальний інтервал

настанова

- Використовуйте однаковий інтервал навколо роздільників.
- Використовуйте той самий інтервал, що й у звичайній прозі.

конкретизація

Зокрема, залиште принаймні одне порожнє місце в наступних місцях, як показано в прикладах, наведених у цій книзі. Для вертикального вирівнювання, рекомендованого в наступних інструкціях, може знадобитися більше місця.

- До і після наступних роздільників і бінарних операторів:

```
+ - * / &
< = > /= <= >=
:= => | ..
:
<>
```

- Поза лапками для рядкових (") та символьних (') літералів, крім випадків, коли це заборонено.
- Зовні, але не всередині, дужки.
- Після коми (,) та крапки з комою (;).

Не залишайте порожніх місць у наступних місцях, навіть якщо це суперечить наведеним вище рекомендаціям.

- Після знаків плюс (+) і мінус (-) при використанні в якості унарних операторів.
- Після виклику функції.
- Усередині розділювачів міток (<< >>).
- До і після оператора піднесення до степеня (**), апострофа (') і крапки (.)
- Між кількома послідовними відкритими або закритими дужками.
- Перед комами (,) і крапкою з комою (;).

Якщо зайві дужки опускаються через правила пріоритету операторів, пробіли можна додатково видалити навколо операторів з найвищим пріоритетом у виразі.

приклад

```
Default_String : constant String :=
      "This is the long string returned by" &
      " default. It is broken into multiple" &
      " Ada source lines for convenience.";

type Signed_Whole_16 is range -2**15 .. 2**15 - 1;
type Address_Area  is array (Natural range <>) of Signed_Whole_16;

Register : Address_Area (16#7FF0# .. 16#7FFF#);
Memory   : Address_Area (       0 .. 16#7FEC#);

Register (Pc) := Register (A);

X := Signed_Whole_16 (Radius * Sin (Angle));

Register (Index) := Memory (Base_Address + Index * Element_Length);

Get (Value => Sensor);

Error_Term := 1.0 - (Cos (Theta)**2 + Sin (Theta)**2);

Z      := X**3;
Y      := C * X + B;
Volume := Length * Width * Height;
```

обгрунтування

Добре використовувати пробіли навколо розділювачів та операторів, оскільки вони зазвичай є короткими послідовностями (один або два символи), які можуть легко загубитися серед довших ключових слів та ідентифікаторів. Пробіли навколо них виділяють їх. Узгодженість у пробілах також допомагає полегшити візуальне сканування вихідного коду.

Однак багато з розділових знаків (коми, крапка з комою, дужки тощо звичні як звичайні розділові знаки. У комп'ютерній програмі вони розставлені інакше, ніж у звичайному тексті, і це відволікає увагу. Тому використовуйте ті ж інтервали, що і в тексті (без пробілів перед комами і крапкою з комою, без пробілів всередині круглих дужок і т.д.).

винятки

Єдиним помітним винятком є двокрапка (:). У мові Ada двокрапку корисно використовувати як табулятор або роздільник стовпців (див. Настанову 2.1.4). У цьому контексті має сенс ставити пробіли до і після двокрапки, а не лише після неї, як у звичайному тексті.

примітки щодо автоматизації

Настанови в цьому розділі легко застосовувати за допомогою автоматичного форматора коду.

Відступ

настанова

- Відступи та вирівнювання вкладених структур керування, ліній продовження та вбудованих блоків послідовно.
- Розрізняють відступи для вкладених керуючих структур і для ліній продовження.
- Для відступів використовуйте пробіли, а не символ табуляції (Nissen and Wallis 1984, §2.2).

конкретизація

Зокрема, рекомендується дотримуватися таких правил відступів, як показано на прикладах у цій книзі. Зверніть увагу, що описано мінімальний відступ. Для вертикального вирівнювання, рекомендованого в наступних настановах, може знадобитися більше пробілів.

- Використовуйте рекомендовану абзацну нумерацію, наведену в Довідковому посібнику Ada (1995).
- Використовуйте три пробіли як основну одиницю відступу для вкладеності.
- Використовуйте два пробіли як основну одиницю відступу для ліній продовження.

Етикетка відступає на три пробіли:

```
begin
<<label>>
   <statement>
end;
```

```
<long statement with line break>
  <trailing part of same statement>
```

Інструкція if та звичайний цикл:

```
if <condition> then
   <statements>
elsif <condition> then
   <statements>
else
   <statements>
end if;
```

```
<name>:
loop
   <statements>
   exit when <condition>;
   <statements>
end loop <name>;
```

Цикли з ітераційними схемами for та while:

```
<name>:
   for <scheme> loop
      <statements>
   end loop <name>;
```

```
<name>:
   while <condition> loop
      <statements>
   end loop <name>;
```

Блок і регістр, як рекомендовано в довіднику Ada Reference Manual (1995):

```
<name>:
   declare
      <declarations>
   begin
      <statements>
   exception
      when <choice> =>
         <statements>
      when others =>
         <statements>
   end <name>;
```

```
case <expression> is
   when <choice> =>
      <statements>
   when <choice> =>
      <statements>
   when others =>
      <statements>
end case;  --<comment>
```

Ці відмінкові оператори економлять місце порівняно з рекомендаціями Ada Reference Manual (1995) і залежать від дуже коротких списків операторів, відповідно. Що б ви не обрали, будьте послідовними:

```
case <expression> is
when <choice> =>
     <statements>
when <choice> =>
     <statements>
when others =>
     <statements>
end case;
```

```
case <expression> is
   when <choice> => <statements>
                    <statements>
   when <choice> => <statements>
   when others   => <statements>
end case;
```

Різні форми вибіркового прийому, а також часові та умовні виклики на вступ:

```
select
   when <guard> =>
      <accept statement>
      <statements>
or
   <accept statement>
   <statements>
or
   when <guard> =>
      delay <interval>;
      <statements>
or
   when <guard> =>
      terminate;
else
   <statements>
end select;
```

```
select
   <entry call>;
   <statements>
or
   delay <interval>;
   <statements>
end select;

select
   <entry call>;
   <statements>
else
   <statements>
end select;

select
   <triggering alternative>
then abort
   <abortable part>
end select;
```

Заява про згоду:

```
accept <specification> do
   <statements>
end <name>;
```

```
separate (<parent unit>)
<proper body>
```

Підрозділ:

```
separate (<parent unit>)
<proper body>
end <name>;
```

Відповідні органи програмних підрозділів:

```
procedure <specification> is
   <declarations>
begin
   <statements>
```

(continues on next page)

(continued from previous page)

```
exception
   when <choice> =>
      <statements>
end <name>;

function <specification>
  return <type name> is
   <declarations>
begin
   <statements>
exception
   when <choice> =>
      <statements>
end <name>;
```

```
package body <name> is
   <declarations>
begin
   <statements>
exception
   when <choice>=>
      <statements>
end <name>;

task body <name> is
   <declarations>
begin
   <statements>
exception
   when <choice>=>
      <statements>
end <name>;
```

Контекстні примітки до одиниць компіляції розташовано у вигляді таблиці. Загальні формальні параметри не затуляють саму одиницю компіляції. Специфікації функцій, пакетів та завдань використовують стандартні відступи:

```
with <name>; use <name>;
with <name>;
with <name>;

<compilation unit>

generic
   <formal parameters>
<compilation unit>
```

```
function <specification>
  return <type>;

package <name> is
   <declarations>
private
   <declarations>
end <name>;

task type <name> is
   <entry declarations>
end <name>;
```

Інстанції родових одиниць та відступ від запису:

```
procedure <name> is
   new <generic name> <actuals>

function <name> is
   new <generic name> <actuals>

package <name> is
   new <generic name> <actuals>
```

```
type ... is
   record
      <component list>
      case <discriminant name> is
         when <choice> =>
            <component list>
         when <choice> =>
            <component list>
      end case;
   end record;
```

Відступ для вирівнювання запису:

```
for <name> use
   record <mod clause>
      <component clause>
   end record;
```

Теговані типи та розширення типів:

```
type ... is tagged
   record
      <component list>
   end record;

type ... is new ... with
   record
      <component list>
   end record;
```

приклад

```
Default_String : constant String :=
      "This is the long string returned by" &
      " default.  It is broken into multiple" &
      " Ada source lines for convenience.";

...

   if Input_Found then
      Count_Characters;

   else  --not Input_Found
      Reset_State;
      Character_Total :=
        First_Part_Total  * First_Part_Scale_Factor  +
        Second_Part_Total * Second_Part_Scale_Factor +
        Default_String'Length + Delimiter_Size;
   end if;
```

(continues on next page)

(continued from previous page)

```
end loop;
```

обгрунтування

Відступи покращують читабельність коду, оскільки дають вам візуальний індикатор структури програми. Рівні вкладеності чітко ідентифікуються за допомогою відступів, а перше та останнє ключові слова в конструкції можна візуально співставити.

Хоча існує багато дискусій щодо кількості пробілів для відступів, причиною відступів є зрозумілість коду. Той факт, що код має однакові відступи, є більш важливим, ніж кількість пробілів, що використовуються для відступів.

Крім того, у Довідковому посібнику з Ada (1995, §1.1.4) зазначено, що макет, показаний у прикладах і синтаксичних правилах посібника, є рекомендованим макетом коду, який слід використовувати для програм на Ada: "Правила синтаксису, що описують структуровані конструкції, подано у формі, яка відповідає рекомендованій абзацній розмітці.... Для частин синтаксичного правила використовуються різні рядки, якщо відповідні частини конструкції, що описується правилом, повинні знаходитися на різних рядках.... Рекомендується, щоб усі відступи були кратними базовому кроку відступу (кількість пробілів для базового кроку не визначено)."

Важливо робити відступи для рядків продовження інакше, ніж для вкладених керуючих структур, щоб візуально їх розрізняти. Це запобігає ускладненню сприйняття структури коду під час його перегляду.

Розміщення контекстних клауз (специфікації контексту?) на окремих рядках полегшує супровід; зміна контекстної клаузи є менш схильною до помилок.

Відступ за допомогою пробілів є більш переносним, ніж відступ за допомогою табуляції, оскільки символи табуляції по-різному відображаються на різних терміналах і принтерах.

винятки

Якщо ви використовуєте шрифт змінної ширини, табуляція вирівнюється краще, ніж пробіли. Однак, залежно від налаштувань табуляції, рядки з послідовними відступами можуть мати дуже малу довжину.

примітки щодо автоматизації

Настанови в цьому розділі легко застосовувати за допомогою автоматичного форматора коду.

Вирівнювання операторів

настанова

- Вирівняйте оператори по вертикалі, щоб підкреслити локальну структуру та семантику програми.

приклад

```
   if Slot_A >= Slot_B then
      Temporary := Slot_A;
      Slot_A    := Slot_B;
      Slot_B    := Temporary;
   end if;

   ----------------------------------------------------------------
   Numerator   := B**2 - 4.0 * A * C;
   Denominator := 2.0 * A;
   Solution_1 := (B + Square_Root(Numerator)) / Denominator;
   Solution_2 := (B - Square_Root(Numerator)) / Denominator;
   ----------------------------------------------------------------

   X := A * B +
        C * D +
        E * F;

   Y := (A * B + C) +  (2.0 * D - E) -  -- basic equation
        3.5;                            -- account for error factor
```

обгрунтування

Вирівнювання полегшує бачення розташування операторів і, отже, робить візуальний акцент на тому, що робить код.

Використання рядків і інтервалів у довгих виразах може підкреслити терміни, пріоритет операторів та іншу семантику. Це також може залишити місце для виділення коментарів у виразі.

винятки

Якщо вертикальне вирівнювання операторів призводить до того, що оператор розбивається на два рядки, особливо якщо розрив відбувається у невідповідному місці, можливо, краще послабити настанову щодо вирівнювання.

примітки щодо автоматизації

Останній приклад вище демонструє своєрідне "семантичне вирівнювання", яке зазвичай не застосовується і навіть не зберігається автоматичними форматерами коду. Якщо ви розбиваєте вираз на смислові частини і розміщуєте кожну з них в окремому рядку, остерігайтеся подальшого використання форматера коду. Ймовірно, він перенесе весь вираз в один рядок і збере всі коментарі в кінці. Втім, деякі форматори досить розумні, щоб залишати переведення рядка недоторканим, коли рядок містить коментар. Хороший форматер визнає, що останній приклад вище не порушує настанови, і тому збереже його в тому вигляді, в якому він був написаний.

Узгодження декларацій

настанова

- Використовуйте вертикальне вирівнювання для покращення читабельності декларацій.
- Вказуйте максимум одну декларацію в рядку.
- Відступайте всі декларації в одній декларативній частині на одному рівні.

конкретизація

Для декларацій, не розділених порожніми рядками, дотримуйтесь цих правил вирівнювання:

- Вирівняйте роздільники двокрапки.
- Вирівняти роздільник ініціалізації :=.
- Якщо використовуються кінцеві коментарі, вирівняйте роздільник коментарів.
- Коли оголошення переповнює рядок, розірвіть рядок і додайте рівень відступу для тих рядків, які обгортаються. Найкраще розривати рядки у таких місцях (1) роздільник коментарів; (2) роздільник ініціалізації; (3) роздільник двокрапки.
- Для оголошень типу перерахування, які не вміщуються в один рядок, розміщуйте кожен літерал в окремому рядку, використовуючи наступний рівень відступу. За необхідності, семантично пов'язані літерали можна розташувати по рядках або стовпчиках, щоб сформувати таблицю.

приклад

Змінні та константи можна оформити у вигляді таблиці з колонками, розділеними символами :, :=, та -.

```
   Prompt_Column : constant        := 40;
   Question_Mark : constant String := " ? "; -- prompt on error input
   Prompt_String : constant String := " ==> ";
```

Якщо в результаті рядки виходять надто довгими, їх можна розташувати на окремому рядку з унікальним рівнем відступу для кожної деталі:

```
   subtype User_Response_Text_Frame is String (1 .. 72);
   -- If the declaration needed a comment, it would fit here.
   Input_Line_Buffer : User_Response_Text_Frame
```

(continues on next page)

(continued from previous page)

```
          := Prompt_String &
             String'(1 .. User_Response_Text_Frame'Length -
                         Prompt_String'Length => ' ');
```

Оголошення літералів перечислення можуть бути перераховані в одному або декількох стовпчиках як:

```
type Op_Codes_In_Column is
      (Push,
       Pop,
       Add,
       Subtract,
       Multiply,
       Divide,
       Subroutine_Call,
       Subroutine_Return,
       Branch,
       Branch_On_Zero,
       Branch_On_Negative);
```

або для економії місця:

```
   type Op_Codes_Multiple_Columns is
         (Push,            Pop,               Add,
          Subtract,        Multiply,          Divide,
          Subroutine_Call, Subroutine_Return, Branch,
          Branch_On_Zero,  Branch_On_Negative);
```

або, щоб підкреслити споріднені групи цінностей:

```
   type Op_Codes_In_Table is
         (Push,            Pop,
          Add,             Subtract,          Multiply,    Divide,
          Subroutine_Call, Subroutine_Return,
          Branch,          Branch_On_Zero,    Branch_On_Negative);
```

обгрунтування

Багато документів зі стандартів програмування вимагають табличного повторення імен, типів, початкових значень та значень у коментарях до заголовків модулів. Ці коментарі є надлишковими і можуть бути неузгодженими з кодом. Вирівнювання самих декларацій у табличному вигляді (див. приклади вище) забезпечує ідентичну інформацію як для компілятора, так і для читача; змушує використовувати максимум одну декларацію у рядку; і полегшує супровід, надаючи місце для ініціалізації та необхідних коментарів. Табличний формат покращує читабельність, запобігаючи тим самим "зануренню" імен у масу декларацій. Це стосується всіх декларацій: типів, підтипів, об'єктів, виключень, іменованих чисел тощо.

примітки щодо автоматизації

Більшість рекомендацій у цьому розділі легко виконати за допомогою автоматичного форматувальника коду. Єдиний виняток - останній приклад перечислюваного типу, який розміщено у рядках відповідно до семантики літералів перечислення. Автоматичний форматер коду не зможе цього зробити і, ймовірно, перемістить літерали перечислення у різні рядки. Однак інструменти, які перевіряють тільки на порушення рекомендацій, повинні приймати табличну форму оголошення типу перечислення.

Більше про вирівнювання

настанова

- Вирівняти режими параметрів та дужки по вертикалі.

конкретизація

Зокрема, рекомендується, щоб ви:

- У кожному рядку розміщуйте по одній формальній специфікації параметрів.
- Вирівняти по вертикалі назви параметрів, двокрапки, зарезервоване слово in, зарезервоване слово out та підтипи параметрів.
- Першу специфікацію параметра слід розміщувати у тому ж рядку, що й назву підпрограми або запису. Якщо будь-які підтипи параметрів виходять за межі довжини рядка, розмістіть першу специфікацію параметра з нового рядка з таким самим відступом, як і для рядка продовження.

приклад

```
   procedure Display_Menu (Title   : in     String;
                           Options : in     Menus;
                           Choice  :    out Alpha_Numerics);
```

Наступні два приклади демонструють альтернативні варіанти застосування цієї настанови:

```
   procedure Display_Menu_On_Primary_Window
         (Title   : in     String;
          Options : in     Menus;
          Choice  :    out Alpha_Numerics);
```

або:

```
   procedure Display_Menu_On_Screen (
         Title   : in     String;
         Options : in     Menus;
         Choice  :    out Alpha_Numerics
       );
```

Вирівнювання круглих дужок робить складні реляційні вирази більш зрозумілими:

```
   if not (First_Character in Alpha_Numerics and then
           Valid_Option(First_Character))        then
```

обгрунтування

Таке вирівнювання полегшує читабельність і зрозумілість, і його легко досягти за умови автоматизованої підтримки. Вирівнювання режимів параметрів надає ефект таблиці зі стовпчиками для назви параметра, режиму, підтипу і, за необхідності, коментарів до параметра. Вертикальне вирівнювання параметрів у підпрограмах у межах одного блоку компіляції ще більше покращує читабельність.

нотатки

Для розташування підпрограм доступні різні варіанти. У другому прикладі вище вирівняно всі назви підпрограм і параметри у програмі. Недоліком цього способу є зайвий рядок, якщо назви підпрограм короткі і виглядають незграбно, якщо є лише один параметр.

Третій приклад - це формат, який зазвичай використовується для зменшення обсягу редагування, необхідного при додаванні, видаленні або переупорядкуванні рядків параметрів. Дужки не потрібно переносити з рядка в рядок. Однак останній рядок параметрів є єдиним без крапки з комою.

винятки

Коли операторна функція має два або більше формальних параметрів одного типу, зручніше оголошувати параметри в одному однорядковому списку, ніж розділяти список формальних параметрів на декілька специфікацій формальних параметрів.

```
   type Color_Scheme is (Red, Purple, Blue, Green, Yellow, White, Black, Brown,␣
↪Gray, Pink);

   function "&" (Left, Right : Color_Scheme) return Color_Scheme;
```

примітки щодо автоматизації

Більшість рекомендацій у цьому розділі легко виконати за допомогою автоматичного форматувальника коду. Єдиний виняток - останній приклад, який показує вертикальне вирівнювання круглих дужок для виділення членів виразу. Цього важко досягти за допомогою автоматичного форматувальника коду, якщо тільки відповідні члени виразу не можуть бути визначені строго через операторний пріоритет.

Порожні рядки

настанова

- Використовуйте порожні рядки для групування логічно пов'язаних рядків тексту (NASA 1987).

приклад

```
   if ... then

      for ... loop
         ...
      end loop;

   end if;
```

У цьому прикладі різні типи декларацій відокремлено порожніми рядками:

```
   type Employee_Record is
      record
         Legal_Name    : Name;
         Date_Of_Birth : Date;
         Date_Of_Hire  : Date;
         Salary        : Money;
      end record;

   type Day is
         (Monday,    Tuesday,   Wednesday, Thursday,  Friday,
          Saturday,  Sunday);

   subtype Weekday is Day range Monday   .. Friday;
   subtype Weekend is Day range Saturday .. Sunday;
```

обгрунтування

Коли порожні рядки використовуються продумано і послідовно, ділянки пов'язаного коду стають більш помітними для читачів.

примітки щодо автоматизації

Автоматичні форматувальники не дуже добре дотримуються цього принципу, оскільки рішення про те, куди вставляти порожні рядки, є семантичним. Втім, багато форматерів мають можливість залишати наявні порожні рядки недоторканими. Таким чином, ви можете вручну вставити рядки і не втратити ефект при запуску такого форматера.

Розбиття на сторінки

настанова

- Виділіть верхню частину кожного пакета або специфікації завдання, верхню частину тіла кожного програмного блоку та оператор завершення кожного програмного блоку.

конкретизація

Зокрема, рекомендується, щоб ви:

- Використовуйте прологи файлів, заголовки специфікацій та заголовки тіла, щоб виділити ці структури, як рекомендовано у Настанові 3.3.
- Використовуйте лінію тире, починаючи з того ж стовпчика, що й поточний відступ, щоб виділити визначення вкладених одиниць, вбудованих у декларативну частину. Вставляйте лінію тире безпосередньо перед визначенням і відразу після нього.
- Якщо дві пунктирні лінії є суміжними, опустити довшу з них.

приклад

```
with Basic_Types;
package body SPC_Numeric_Types is
   ----------------------------------------------------------------------
   function Max
         (Left  : in     Basic_Types.Tiny_Integer;
          Right : in     Basic_Types.Tiny_Integer)
         return Basic_Types.Tiny_Integer is
   begin
      if Right < Left then
         return Left;
      else
         return Right;
      end if;
   end Max;
   ----------------------------------------------------------------------
   function Min
         (Left  : in     Basic_Types.Tiny_Integer;
          Right : in     Basic_Types.Tiny_Integer)
         return Basic_Types.Tiny_Integer is
   begin
      if Left < Right then
         return Left;
      else
         return Right;
      end if;
   end Min;
   ----------------------------------------------------------------------
   use Basic_Types;
begin  -- SPC_Numeric_Types
   Max_Tiny_Integer := Min(System_Max, Local_Max);
   Min_Tiny_Integer := Max(System_Min, Local_Min);
   -- ...
end SPC_Numeric_Types;
```

обгрунтування

Легко не помітити частини програмних блоків, які не видно на поточній сторінці або екрані. Довжина сторінок у презентаційному обладнанні та програмному забезпеченні може бути різною. Чітко позначивши логічні межі сторінки програми (наприклад, пунктирною лінією), ви даєте змогу читачеві швидко перевірити, чи всі програмні блоки видно. Таке розбиття на сторінки також полегшує швидке сканування великого файлу в пошуках певної програмної одиниці.

винятки

Ця настанова не розглядає розміщення коду на фізичній "сторінці", оскільки розміри таких сторінок дуже різняться, і немає єдиної настанови, яка б підходила для цього.

примітки щодо автоматизації

Настанови в цьому розділі легко застосовувати за допомогою автоматичного форматора коду.

Кількість тверджень у рядку

настанова

- Починайте кожне твердження з нового рядка.
- Пишіть не більше одного простого твердження в рядку.
- Розбиття складених операторів на декілька рядків.

приклад

Використовуй:

```
   if End_Of_File then
      Close_File;
   else
      Get_Next_Record;
   end if;
```

замість того, щоб:

```
   if End_Of_File then Close_File; else Get_Next_Record; end if;
```

винятковий випадок:

```
   Put("A=");    Natural_IO.Put(A);    New_Line;
   Put("B=");    Natural_IO.Put(B);    New_Line;
   Put("C=");    Natural_IO.Put(C);    New_Line;
```

обгрунтування

Окреме твердження в кожному рядку покращує здатність читача знаходити твердження і допомагає запобігти пропуску тверджень. Аналогічно, структура складнопідрядного речення стає зрозумілішою, коли його частини розташовані на окремих рядках.

винятки

Якщо оператор довший за місце, що залишилося у рядку, продовжте його на наступному рядку. Ця настанова стосується оголошень, контекстних клаузул та параметрів підпрограм.

Згідно з довідником Ada (1995, §1.1.4), "Інші розриви рядка краще після крапки з комою".

примітки щодо автоматизації

Вказівки цього розділу легко виконати за допомогою автоматичного форматора коду, за винятком останнього прикладу, в якому показано семантичне групування кількох тверджень в одному рядку.

винятки

Приклад інструкцій Put та New_Line показує виправданий виняток. Таке групування тісно пов'язаних операторів в одному рядку робить зрозумілим структурний зв'язок між групами.

Довжина рядка вихідного коду

настанова

- Дотримуйтесь обмеження на максимальну довжину рядка вихідного коду (Nissen and Wallis 1984, §2.3).

конкретизація

Зокрема, рекомендується, щоб ви:

- Обмежте довжину рядка вихідного коду до 72 символів.

обгрунтування

Під час перенесення коду на мові Ada з однієї системи на іншу можуть виникати обмеження на розмір запису вихідних операторів, можливо, з однієї з таких причин: деякі операційні системи можуть не підтримувати записи змінної довжини для стрічкового вводу/виводу, або деякі принтери та термінали підтримують 80-символьну ширину рядка без переведення рядка на новий рядок. Див. додаткове обґрунтування у примітці до Настанови 7.1.2.

Вихідний код іноді потрібно публікувати з різних причин, а папір Letter не такий пробачливий, як комп'ютерний лістинг з точки зору кількості корисних стовпців.

Крім того, існують людські обмеження щодо ширини поля зору для розуміння на рівні, необхідному для читання вихідного коду. Ці обмеження приблизно відповідають діапазону від 70 до 80 стовпчиків.

винятки

Альтернативним варіантом є обмеження довжини вихідного коду до 79 символів. Обмеження у 79 символів відрізняє код від 72-символьного обмеження мови FORTRAN. Це також дозволяє уникнути проблем з терміналами, де символ в останньому стовпчику може бути надрукований неправильно.

примітки щодо автоматизації

Настанови в цьому розділі легко застосовувати за допомогою автоматичного форматора коду.

1.3.2 Підсумок

форматування коду

- Використовуйте однаковий інтервал навколо роздільників.
- Використовуйте той самий інтервал, що й у звичайній прозі.
- Відступи та вирівнювання вкладених структур керування, ліній продовження та вбудованих блоків послідовно.
- Розрізняють відступи для вкладених керуючих структур і для ліній продовження.
- Для відступів використовуйте пробіли, а не символ табуляції (Nissen and Wallis 1984, §2.2).
- Вирівняйте оператори по вертикалі, щоб підкреслити локальну структуру та семантику програми.
- Використовуйте вертикальне вирівнювання для покращення читабельності декларацій.
- Вказуйте максимум одну декларацію в рядку.
- Відступайте всі декларації в одній декларативній частині на одному рівні.
- Вирівняти режими параметрів та дужки по вертикалі.
- Використовуйте порожні рядки для групування логічно пов'язаних рядків тексту (NASA 1987).

- Виділіть верхню частину кожного пакета або специфікації завдання, верхню частину тіла кожного програмного блоку та оператор завершення кожного програмного блоку.
- Починайте кожне твердження з нового рядка.
- Пишіть не більше одного простого твердження в рядку.
- Розбиття складених операторів на декілька рядків.
- Дотримуйтесь обмеження на максимальну довжину рядка вихідного коду (Nissen and Wallis 1984, §2.3).