МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ БІЗНЕС-КОЛЕДЖ

***Циклова комісія програмування***

**РЕФЕРАТ**

з курсу «Тренінг-курс з основ програмування»

**з теми «Стилі кодування та написання якісного коду»**

Ткач Анни Богданівни та Чернишова Романа Святославовича

*ПІБ студентів*

студентів групи **1П-21**

Викладач Марченко С. В.

Черкаси-2022

**Зміст**

1.Вступ ---------------------------ст.3

2.Стандарт оформлення коду ---------------------ст.4-6

3.Стилі програмування ----------------------------ст.7-16

4.Висновок --------------------------------------------ст.17

5.Джерела----------------------------------------------ст.18

**Вступ**

Відповідаючи на це питання одним рядком, можна сказати, що хороший стиль програмування має на увазі наявність структури у написаної програми. Відповідно, якщо програма містить багато рядків програмного коду в одній процедурі, відділена на смислові блоки, нагадує "спагетті", тобто має безліч переходів управління, містить мало коментарів і, як наслідок, стає заплутаною, громіздкою, важко модифікується і керується, то це і є "поганий стиль програмування".

Стандарт оформлення коду

**Стандарт оформлення коду**

 набір правил та угод, що використовуються при написанні сирцевого коду на деяких мовах програмування**.**

**Використання**

Стандарт оформлення коду зазвичай приймається та використовується групою розробників програмного забезпечення для єдиного стилю оформлення коду, над яким йде спільна праця. Метою прийняття та використання стандарту є спрощення сприйняття програмного коду людиною, мінімізація навантаження на пам'ять та зір при читанні програми.

Зразком для стандарту кодування може стати набір угод, прийнятих в якій-небудь поширеній печатній праці з мови програмування (наприклад, стандарт кодування мовою С, що отримав коротку назву K&R, виходить з класичного описання С його творцями — Керніганом та Рітчі), поширена бібліотека або API (так, на поширення угорської нотації вплинув той факт, що її використовували в MS-DOS та Windows API, а більшість стандартів кодування для Delphi використовують манеру кодування бібліотеки VCL). Рідше розробник мови програмування випускає детальні рекомендації по кодуванню на ній; випущені, наприклад, стандарти кодування на C# від Microsoft та на Java від Sun. Запропонована розробником та прийнята в загально відомих джерелах манера кодування може бути доповнена та уточнена у корпоративних стандартах.

**Склад**

Стандарт дуже залежить від використовуваної мови програмування. Наприклад, стандарт оформлення коду для [С](https://uk.wikipedia.org/wiki/C_(%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)) буде сильно відрізнятися від стандарту для мови [BASIC](https://uk.wikipedia.org/wiki/BASIC). Зазвичай метою стандарту є досягнення такого стану, коли [програміст](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%96%D1%81%D1%82) достатньої кваліфікації міг би дати висновки про функцію, яку виконує конкретна ділянка коду, а в ідеалі — також визначити його коректність, вивчивши тільки цю ділянку коду, або мінімально вивчивши інші частини програми. Іншими словами, сенс коду повинен буди зрозумілим з самого коду без необхідності вивчати контекст. Тому стандарти кодування будуються так, щоб шляхом визначеного візуального оформлення елементів програми збільшити інформативність коду для людини.

Зазвичай, стандарт оформлення коду описує:

* засоби вибору значень та використовуваний регістр символів для імен [змінних](https://uk.wikipedia.org/wiki/%D0%97%D0%BC%D1%96%D0%BD%D0%BD%D0%B0) та інших ідентифікаторів.
  + запис [типу](https://uk.wikipedia.org/wiki/%D0%A2%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85) змінної в її ідентифікаторі ([угорська нотація](https://uk.wikipedia.org/wiki/%D0%A3%D0%B3%D0%BE%D1%80%D1%81%D1%8C%D0%BA%D0%B0_%D0%BD%D0%BE%D1%82%D0%B0%D1%86%D1%96%D1%8F" \o "Угорська нотація)).
  + регістр символів (нижній, верхній, «верблюжий», «верблюжий» з малої букви), використання знаку підкреслення для розділу слів.
* стиль відступів при оформленні логічних блоків — чи використовуються [табуляції](https://uk.wikipedia.org/wiki/%D0%A2%D0%B0%D0%B1%D1%83%D0%BB%D1%8F%D1%86%D1%96%D1%8F), ширина відступу.
* спосіб розстановки дужок, що обмежують логічні блоки.
* використання пробілів при оформленні логічних та [арифметичних](https://uk.wikipedia.org/wiki/%D0%90%D1%80%D0%B8%D1%84%D0%BC%D0%B5%D1%82%D0%B8%D0%BA%D0%B0) виразів.
* стиль коментарів та використання документуючих коментарів.

Поза стандартом існують правила про:

* відсутність магічних чисел
* обмеження розміру коду по горизонталі на вертикалі.

## Стандарти кодування і синтаксис мов програмування

Основні принципи поширених стандартів кодування в останній час впливають на синтаксис нових мов програмування. В деяких з них, угоди, що раніше використовувались тільки в стандартах кодування, стають обов'язковими елементами синтаксису. Так, у деяких сучасних мовах ([Python](https://uk.wikipedia.org/wiki/Python), [Nemerle](https://uk.wikipedia.org/wiki/Nemerle" \o "Nemerle)) відступи впливають на логіку виконання (тобто блоки коду виділяються не ключовими словами, а розміром відступів). В інших стали частиною мови угоди про регістри літер та префіксів для типів, [констант](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%81%D1%82%D0%B0%D0%BD%D1%82%D0%B0), змінних та полів [класів](https://uk.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)).

Якщо раніше недисциплінований програміст міг ігнорувати стандарти кодування з особистих міркувань, заради зручності та швидкості написання коду, то тепер, при роботі з новими мовами, дотримання стандартів деякою мірою контролюється [транслятором](https://uk.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D1%81%D0%BB%D1%8F%D1%82%D0%BE%D1%80)

**Стилі програмування:**

У цьому розділі будуть пояснені різні стилі програмування:

1. Функціональне програмування
2. Модульне програмування
3. Об'єктно-орієнтоване програмування

**1. Функціональне програмування**

Що таке функціональне програмування?

Функціональне програмування-це стиль програмування, в якому акцент робиться на обчисленні виразів, а не на виконанні команд. Це означає, що програма, написана функціональною мовою програмування, складається з набору визначень функцій і виразу, значення якого виводиться в якості результату програми. При обчисленні виразу немає побічних результатів( наприклад, присвоювання), тому вираз завжди буде обчислюватися з одним і тим же значенням, якщо його обчислення завершується. Через це вираз завжди можна замінити його значенням без зміни загального результату. Це називається посилальною прозорістю. Слово 'функція' насправді може бути замінено терміном 'дія'. Функція виконує щось, наприклад, Drive (brand\_of\_car)

Посилальна прозорість:

Чисто функціональні мови досягають посилальної прозорості, забороняючи присвоювання глобальним змінним. Кожен вираз є константою або функцією програми, обчислення якої не має побічного ефекту, воно тільки повертає значення, і це значення залежить тільки від визначення функції і значень її аргументів.

**2. Модульне програмування**

Що таке модульне програмування?

Модуль в модульному програмуванні-це набір функцій (або процедур), які якимось чином пов'язані між собою. Цей спосіб програмування використовується в багатьох додатках.

Спосіб роботи

З огляду на проблему, ви починаєте з аналізу проблеми. Ви уважно вивчаєте вимоги завдання і переконайтеся, що на всі питання дано відповіді, перш ніж приступати до проектування модулів. Потім ви розбиваєте проблему на підзадачі. Кожну з цих завдань ви вирішуєте в одному або декількох модулях. Всі модулі разом вирішують повну проблему. Самі модулі можуть бути розділені на більш дрібні модулі.

Точно так само, як ви розбиваєте проблему на менші завдання, ви розбиваєте модуль на підмодулі.

Як вже було сказано, кожна підзадача буде вирішуватися в різних підмодулях. Зрештою, вся проблема вирішується у всьому модулі.

Модуль-це набір функцій (дій, див.главу другу), які пов'язані один з одним. Наприклад, модуль під назвою "водіння" містить функції drive(brand\_of\_car), accelerate(прискорення) і stop (зупинка). Модульна програма завжди більш зрозуміла, ніж чисто функціональна програма, тому що різні модулі класифікуються на основі релевантності.

Рекомендація:

* Модульне програмування

<a href="http://www.eee.bham.ac.uk/dsvp\_gr/roxby/eem1e1/lecture7/sld004.htm">http://www.eee.bham.ac.uk/dsvp\_gr/roxby/eem1e1/lecture7/sld004.htm

* Модульне проектування та програмування

<a href="http://www.cs.runet.edu /~jchase/pdl.html ">http://www.cs.runet.edu /~jchase/pdl.html

**3.Об'єктно-орієнтоване програмування (ООП)**

Що таке ООП?

Об'єктно-орієнтоване програмування (або ООП) - це революційна концепція, яка змінила правила розробки комп'ютерних програм. ООП організовано навколо об'єктів, а не дій. Зазвичай програма має простий потік: введення даних, Обробка даних і виведення результату. У ООП інша точка зору; що нас дійсно хвилює, так це об'єкти, якими ми хочемо маніпулювати, а не логіка, необхідна для маніпулювання ними. Прикладами таких об'єктів можуть бути люди (наприклад, описані по імені, адресою і так далі), або будівлі і поверхи (властивості яких можна описувати і управляти ними), або навіть такі дрібниці, як кнопка або панель інструментів.

Об'єктно-орієнтоване програмування-це альтернатива модульному програмуванню. Метод проектування, пов'язаний з ООП, - це об'єктно-орієнтоване проектування. У об'єктно-орієнтованій програмі модулі є класами, а не процедурами. Клас-це набір об'єктів.

*Об'єкт:*

Як вже було сказано, ООП - це все, що пов'язано з об'єктами. Об'єкт насправді є контейнером, заповненим інформацією. Ви можете бачити об'єкт як чорний ящик, який відправляє і отримує повідомлення. Чорний ящик, або об'єкт, містить код і дані. Код-це послідовності комп'ютерних інструкцій, а дані-це інформація, з якою працюють інструкції. У C++ одиниці коду називаються функціями, а одиниці даних називаються структурами. Функції і структури не пов'язані один з одним. Наприклад, функція може працювати з більш ніж однією структурою, а структура може використовуватися в більш ніж одній функції.

Структура може бути загальнодоступними даними, до яких може отримати доступ інший об'єкт. Об'єкт також може містити особисті дані. Доступ до цих даних може отримати тільки сам об'єкт, а не інший об'єкт. Особисті дані реалізовані таким чином, що видатний об'єкт не може змінювати дані, поки в цьому немає необхідності, або дані, які не повинні бути змінені.

*Повідомлення:*

Весь зв'язок між об'єктами здійснюється за допомогою повідомлень. Об'єкт надсилає, але також отримує повідомлення. Об'єкт, якому надсилається повідомлення, є одержувачем повідомлення. Повідомлення визначають інтерфейс об'єкта. Принаймні, все, що може зробити об'єкт, представлено його повідомленнями. Насправді, об'єкт зазвичай може робити більше, тому що об'єкт також може мати приватні функції.

Зазвичай обмін повідомленнями здійснюється між об'єктами. Один об'єкт отримує повідомлення від іншого об'єкта і відправляє повідомлення назад тому ж самому або іншому об'єкту.

Навіть в невеликій програмі кількість об'єктів може бути дуже великим. У цьому прикладі перший об'єкт отримує вхідні дані. Це може бути повідомлення від іншого об'єкта або, наприклад, щось, що вводить користувач. Перші об'єкти обробляють цей введення, відправляють повідомлення другому об'єкту. Це повідомлення може бути результатом роботи функції, наприклад. Другі об'єкти обробляють повідомлення і відправляють результат назад першому об'єкту. З цим результатом перший об'єкт виконує нове обчислення, наприклад, і надсилає результат цього третьому об'єкту. Цей об'єкт також обробляє дані і видає вихідні дані. Цей висновок може бути новим повідомленням для іншого об'єкта або просто виведенням чого-небудь на монітор користувача.

*Приховування інформації:*

Надання доступу до об'єкта шляхом відправки та отримання повідомлень при збереженні конфіденційних відомостей про об'єкт називається приховуванням інформації. Інше слово, яке означає Те ж саме, - це інкапсуляція.< ім'я="\_Toc473031126"> приховування інформації - це добре; один об'єкт повинен знати тільки ті відомості про інший об'єкт, які мають відношення до справи. Зробіть елементи в об'єкті закритими або захищеними, коли це можливо.

*Метод:*

Об'єкт визначається його класом. Клас визначає все, що стосується об'єкта. Об'єкти-це окремі екземпляри класу. Можна мати, наприклад, більше кнопок в одному діалоговому вікні. Кожна кнопка є екземпляром buttonclass. Інший приклад: розглянемо клас 'Dog'. Об'єкт з цього класу називається 'Spot'. Клас' Dog ' визначає, що являє собою об'єкт. Можливо, у вас є більше одного об'єкта цього класу; можливо, ви захочете назвати їх "Фідо", "Ровер' і так далі. Кожен з них є екземпляром класу 'Dog'. Тепер припустимо, що клас ' Dog 'визначає, що вони можуть розуміти такі повідомлення, як" гавкіт "і " перевертання'. Дія, яку виконує повідомлення, називається методом. Це, по суті, код, який виконується, коли повідомлення отримано об'єктом.

Аргументи часто надаються як частина повідомлення. Наприклад, для повідомлення "перекидання" можуть знадобитися аргументи " як швидко "і другий аргумент"скільки разів". Аргументи визначають спосіб поведінки дії.

*Повторне використання:*

Спадкування також означає, що ви повторно використовуєте його. Вам не потрібно писати весь код заново, ви просто повторно використовуєте один або кілька класів, які мають бажану поведінку.

Тепер, що робити, якщо ми хочемо створити клас (з назвою 'Wolf'), який може робити саме те, що робить 'Dog'? Нам не потрібно переписувати весь клас; ми просто створюємо клас, похідний від 'Dog'. Цей клас успадковує всі існуючі повідомлення так званого базового класу; він має таку ж поведінку.

У класі 'Wolf' ми можемо створювати нові (додаткові) повідомлення ,наприклад'hunt'. Цей клас підтримує всі повідомлення 'Dog' , а також деякі нові, включаючи'hunt'.

**4.Моделювання даних**

Моделювання даних-це перший крок у розробці об'єктно-орієнтованої програми. В результаті моделювання даних ви можете визначити об'єкти. Простий підхід до створення моделі даних, який дозволяє вам візуалізувати модель, полягає в тому, щоб намалювати квадрат для представлення кожного окремого елемента даних, про який Ви знаєте, а потім виразити взаємозв'язки між кожним з цих елементів даних. Відносини між елементами даних можуть бути виражені такими словами, як "є частиною", або" використовується", або "використовує' і так далі. З цього загального опису ви можете створити набір класів і підкласів, які визначають всі загальні взаємозв'язки.

За допомогою цієї моделі даних ви можете легко створити набір класів або навіть повну програму, тому що ця модель даних визначає роботу програми.

Відмінності між ООП і функціональним і модульним програмуванням

Різниця між ООП і функціональним і модульним програмуванням полягає в тому, що об'єкти в ООП маніпулюють тільки своїм власним станом. Наприклад: Peugeot::Drive(), Citroen::Drive(), Car::Accelerate(швидкість) і car:: Stop() . Стан всіх об'єктів разом визначає стан всієї програми. Об'єктно-орієнтовану програму завжди легше читати, ніж модульну програму, тому що всі об'єкти представляють свою інформацію. Спосіб роботи oo-програми можна визначити, намалювавши відносини між об'єктами. Саме тому завжди корисно намалювати модель даних з потрібними Вам об'єктами і їх відносинами.

Крім того, OO-програму простіше підтримувати, тому що зміна функціональності об'єкта (зазвичай) не робить ніякого впливу на стан інших об'єктів. Для реалізації нової функціональності може бути досить просто додати в додаток один або кілька об'єктів і визначити їх відносини.

Факт повторного використання коду можна побачити за допомогою можливості засновувати спеціалізовану інформацію (Peugeot, Citroen) на загальній інформації (автомобіль).

*Додатково:*

На C нам довелося кодувати наші власні помилки. У C++ ми можемо успадковувати їх.

**Стиль в архітектурі ПЗ.**

Архітектура розробки програмного забезпечення – це розроблена структура програми, яка включає визначення взаємодії компонентів інтерфейсу з внутрішніми процесами програми. Простіше кажучи, це своєрідний підхід, який визначає які функції за що відповідають та як вони взаємодіють між собою.

Точного розуміння та чіткого формулювання цього процесу немає. Основним завданням є створення логічної структури програми та спрощення взаємодії між розробниками. Це дає можливість надалі вносити зміни до програми, опрацьовуючи конкретні аспекти, а не переробляючи все програмне забезпечення. Архітектура та проектування ПЗ забезпечують гарантію того, що застосунок буде виконувати завдання та слідувати своєму призначенню, визначеному під час початкових етапів розробки.

Головна ідея архітектури полягає у тому, щоб знизити складність сприйняття системи внаслідок розмежування повноважень та створення чіткої структури. Архітектура та дизайн програмного забезпечення дозволяють створити чітку структуру, за якою зручно працювати програмістам. Від її якості залежить, наскільки просто проходитиме обслуговування ПЗ, його зміни, доповнення та підтримка.

Архітектура та проектування може мати атрибут, іменований стилем. Він дозволяє надати певну однаковість. Він визначається якраз за допомогою вищеописаних шаблонів, а також конкретних з'єднувачів та компонентів. Ключова роль стилю – створити зрозумілу та цілісну архітектуру.

Стиль – це певний набір принципів, який дозволяє використовувати шаблони та зводити структуру до єдиного та простого для сприйняття виду. Це шлях до спільного розуміння та мови. Зокрема, якщо архітектуру розробляє одна команда розробників, а надалі зміни до функціоналу вносить інша команда фахівців, стиль – це зв'язуюча їх ланка, яка дозволяє зрозуміти структуру. Ось кілька прикладів стилів, що використовуються при архітектурі програмного проекту:

* **Клієнт/сервер**. Стиль, за якого вся система програми поділяється на дві програми, де клієнт може виконати запит до сервера.
* **Компонентна архітектура**. Весь дизайн повністю розкладається на логічні та функціональні компоненти. Їх можна використовувати повторно, а інтерфейси зв'язку ретельно опрацьовуються.
* **Проблемно-орієнтований дизайн**. Це стиль, що дозволяє опрацювати бізнес-процеси, та орієнтований створення моделей ділової активності програми.
* **Багатошарова архітектура програмного забезпечення**. Кожна функціональна область поділяється на шари, що дозволяє окремо працювати з кожним із них.
* **Шина повідомлень**. Стиль, що дозволяє надсилати повідомлення кількома каналами, що дозволяє взаємодії модулів, у своїй конкретні дані модулям не надаються.
* **3-рівнева архітектура**. Функціональність поділяється на певні сегменти, стиль схожий за своєю структурою з багатошаровою архітектурою. Різниця лише в тому, що сегменти фізично знаходяться на різних комп'ютерах.
* **Об'єктно-орієнтований стиль**. Кожен об'єкт є самостійним та може багаторазово використовуватися, він містить набір даних та поведінки, а відповідальність системи розподіляється між цими об'єктами.
* **Сервісно-орієнтований стиль**. У ньому використовуються окремі послуги, що забезпечують функціональність програми. Вони обмінюються між собою повідомленнями та утворюють єдине середовище, при цьому є незалежними одна від одної.

Архітектура та дизайн програмного забезпечення рідко обмежуються одним стилем. Як правило, використовується поєднання, яке створює повноцінну систему. Кожен стиль описується у технічній документації, щоб команда розробників змогла передати дані. Крім того, вимоги до системи безпеки зобов'язують використовувати різні стилі, зокрема із застосуванням багаторівневої архітектури.

На вибір архітектурного стилю впливає безліч факторів, включно із обмеженням інфраструктури середовища розробки, стеком технологій, досвідченістю розробників. У компанії Wezom підбирають оптимальне рішення, спираючись на поставлені завдання під час розробки. Насамперед стиль визначається шляхом опрацювання виконуваних процесів. Ми враховуємо можливість подальшого масштабування програми та особливості впровадження її на підприємство клієнта. Тому, якщо вам потрібна архітектура програмного забезпечення, зателефонуйте нам або завітайте до офісу, щоб обговорити деталі.

**Висновок**

Загалом, працюючи над програмою, програміст, особливо початківець, повинен добре уявляти, що програма, яку він розробляє, призначена, з одного боку, для користувача, з іншого – для самого програміста. Текст програми потрібен насамперед самому програмісту, а також іншим людям, з якими він спільно працює над проектом. Тому для того, щоб робота була ефективною, програма повинна бути легко читається, її структура повинна відповідати структурі та алгоритму розв’язуваної завдання. Як цього добитися? Треба дотримуватися правил хорошого стилю програмування. Стиль програмування – це набір правил, яких слід програміст (усвідомлено або тому, що так роблять інші) в процесі своєї роботи. очевидно, що хороший програміст повинен слідувати правилам хорошого стилю.

Хороший стиль програмування передбачає:

• використання коментарів;

• використання несуть смислове навантаження імен змінних, процедур і функцій;

• використання відступів;

• використання порожніх рядків.

Дотримання правил хорошого стилю програмування значно зменшує ймовірність появи помилок на етапі набору тексту, робить програму легко читається, що, в свою чергу, полегшує процеси налагодження та внесення змін.

Чіткого критерію оцінки ступеня відповідності програми хорошого стилю програмування не існує. Разом з тим досить одного погляду, щоб зрозуміти, чи відповідає програма хорошого стилю чи ні.

Зводити поняття стилю програмування лише до правил запису тексту програми було б невірно. Стиль, якого дотримується програміст, проявляється під час роботи програми. Хороша програма повинна бути перш за все надійної і дружній по відношенню до користувача.

Надійність має на увазі, що програма, не покладаючись на розумне поведінку користувача, контролює вихідні дані, перевіряє результат виконання операцій, які з якої-небудь причини можуть бути не виконані, наприклад, операцій з файлами.

Дружність припускає добре спроектовані діалогові вікна, наявність довідкової системи, розумне і передбачуване, з точки зору користувача, поведінка програми.

Використане джерело:

1. Different styles of programming by Alex Marbus. URL: <https://www.codeproject.com/Articles/569/Different-Styles-of-Programming>
2. Delphi. URL: <http://delphi.dp.ua/styl-programuvannya/>
3. URL: <https://wezom.com.ua/ua/blog/arhitektura-programmnogo-obespecheniya>

Використане джерело (user 1) :

<https://uk.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82_%D0%BE%D1%84%D0%BE%D1%80%D0%BC%D0%BB%D0%B5%D0%BD%D0%BD%D1%8F_%D0%BA%D0%BE%D0%B4%D1%83>