**Тема 1. Вступне заняття. Правила оформлення коду.**

За матеріалами [«Stanford C++ Style Guide»](https://stanford.edu/class/archive/cs/cs106b/cs106b.1158/styleguide.shtml)

## Пробіли та відступи

Відделяйте пробілами фігурні дужки:

// Погана практика

int x = 3, y = 7; double z = 4.25; x++;

if (a == b) { foo(); }

// Добра практика

int x = 3;

int y = 7;

double z = 4.25;

x++;

if (a == b) {

foo();

}

Ставте пробіли між операторами та операндами:

int x = (a + b) \* c / d + foo();

Коли рядок стає довше 100 символів, розділіть її на дві, зробивши перенесення на новий рядок після оператора, и продовжуйте писати:

int result = reallyLongFunctionOne() + reallyLongFunctionTwo() +

reallyLongFunctionThree() + reallyLongFunctionFour();

int result2 = reallyLongFunction(parameterOne, parameterTwo, parameterThree,

parameterFour, parameterFive, parameterSix);

Залишайте пусті лінії (рядки) між функціями та між групами виразів:

void foo() {

...

}

// пустая линия

void bar() {

...

}

## Назви та змінні

Надавайте змінним описові імена, як firstName або homeworkScore. Уникайте однолітерних назв як x або c, за виключенням ітераторів як i.

Називайте змінні та функції, використовуючи верблюжийРегистр. Називайте класи ПаскальнимРегистром, а константи — в ВЕРХНЬОМУ\_РЕГИСТРІ.

Якщо змінна використовується лише всередені певного if, то робіть її локальною, оголошуючи в тому же блоці коду, а не глобальною.

Вибирайте відповідний тип даних для ваших змінних. Якщо  змінна містить лише цілі числа, то визначайте її як int, а не double.

Використовуйте текстовий рядок, стандартний для C++, а не С. С++ лякає тем, що має два види текстових рядків: клас string з С++ та старий char\* (масив символів) з С:

// Погана практика: текстовий рядок в стилі C

char\* str = "Hello there";

// Добра практика: текстовий рядок в стилі C++

string str = "Hello there";

Якщо визначена константа часто використовується в вашому коді, то позначте її як const та завжди посилайте на цю константу, а не на її значення:

const int VOTING\_AGE = 18;

Ніколи не оголошуйте глобальну змінну, що змінює значення. Глобальними змінними повинні бути тільки константи. Замість того, щоб робити значення глобальним, робіть його параметром та повертайте значення, коли потрібно:

// Погана практика

int count; // Глобальна змінна

void func1() {

count = 42;

}

void func2() {

count++;

}

int main() {

func1();

func2();

}

// Добра практика!

int func1() {

return 42;

}

void func2(int& count) {

count++;

}

int main() {

int count = func1();

func2(count);

}

## Базові вирази С++

С++ заснований на С, тому завжди є варіант вирішити задачу «шляхом С++» або «шляхом С». Наприклад, коли ви бажаєте вивести що небуть на системну консоль, ви можете зробити це «шляхом С++» , використовуючи оператор виведення cout, в той час як «шляхом С» ви б використовували глобальну функцію як printf:

// Погана практика

printf("Hello, world!\n");

// Добра практика

cout << "Hello, world!" << endl;

Часто затруднюєтесь з вибором між for та while? Використовуйте цикл for, коли ви знаєте кількість повторів, а цикл while, коли кількість повторів невідома:

// Повторює 'size' раз

for (int i = 0; i < size; i++) {

...

}

// Повторює, доки більше не буде рядків

string str;

while (input >> str) {

...

}

Коли використовуєте оператори управління як if / else, for, while, завжди використовуйте {} та відповідні відступи, навіть якщо тіло всього оператора управління складається лише з одного рядка:

// Погана практика

if (size == 0) return;

else

for (int i = 0; i < 10; i++) cout << "ok" << endl;

// Добра практика

if (size == 0) {

return;

} else {

for (int i = 0; i < 10; i++) {

cout << "ok" << endl;

}

}

Намагайтесь уникати виразів break або continue. Використовуйте їх тільки у випадку, якщо це абсолютно необхідно.

В C++ є функція exit, яка негайно завершує програму. Не рекомендується використовувати цю функцію. Програма завжди повинна закінчуватися природньо, досягаючи оператору return функції main.

Коли використовуєте вираз if / else, відповідним чином вибирайте між різними шаблонами if та else в залежності від умов співвідношення оди до одного. Потрібно уникати надлишкових if:

// Погана практика

if (grade >= 90) {

cout << "You got an A!";

}

if (grade >= 80 && grade < 90) {

cout << "You got a B!";

}

if (grade >= 70 && grade < 80) {

cout << "You got a C!";

}

// Добра практика

if (grade >= 90) {

cout << "You got an A!";

} else if (grade >= 80) {

cout << "You got a B!";

} else if (grade >= 70) {

cout << "You got a C!";

}

...

Якщо у вас є вираз if / else, яке повертає логічне значення, повертайте результати перевірки напряму:

// Погана практика

if (score1 == score2) {

return true;

} else {

return false;

}

// Добра практика

return score1 == score2;

Ніколи не перевіряйте значення логічного типу, використовуючи == або != з true або false:

// Погана практика

if (x == true) {

...

} else if (x != true) {

...

}

// Добра практика

if (x) {

...

} else {

...

}

## Надлишковість

Якщо ви використовуєте один й той же код двічі або більше, то знайдіть спосіб вилучити надлишковий код, щоб він не повторювався. наприклад, його можна помістити у допоміжну функцію. Якщо повторюваний код схожий, але не зовсім, то намагайтесь зробити допоміжну функцію, яка приймає параметри та представляє частину, яка різниться:

// Погана практика

foo();

x = 10;

y++;

...

foo();

x = 15;

y++;

// Добра практика

helper(10);

helper(15);

...

void helper(int newX) {

foo();

x = newX;

y++;

}

Переміщуйте спільний код з виразу if / else, щоб він не повторювався:

// Погана практика

if (x < y) {

foo();

x++;

cout << "Привіт!";

} else {

foo();

y++;

cout << "Привіт!";

}

// Добра практика

foo();

if (x < y) {

x++;

} else {

y++;

}

cout << "Привіт!";

## Коментарі

**Заглавний коментар**. Розміщуйте заглавний коментар, який описує призначення файлу, зверху кожного файлу. Вважайте, що читач вашого коментаря є просунутим досвіченим програмістом, але не кимось, хто вже бачив ваш код раніше.

**Заголовок функції / конструктора**. Розміщуйте заголовковий коментар на кожному конструкторі та функції вашого файлу. Заголовок повинен описувати поведінку та / або ціль функції.

**Параметри / повернення**. Якщо ваша функція приймає параметри, то кратко опишіть їх ціль та смисл. Якщо ваша функція повертає значення — кратко опишіть, що вона повертає.

**Виключення**. Якщо ваша функція навмисно видає будь-які виключення для визначених помилкових випадків, то це потребує опису.

**Коментарі на одному рядку**. Якщо всередині функції є секція коду, яка довга, складна або незрозуміла, то коротко опишіть її призначення.

**TODO**. Потрібно вилучити усі // TODO коментарі щодо намірів написання коду перед тим, як закінчувати та здавати програми.

## Ефективність

Викликаючи велику функцію та використовуючи результат декілька разів, збережіть результат в змінній замість того, щоб постійно викликати дану функцію:

// Погана практика

if (reallySlowSearchForIndex("abc") >= 0) {

remove(reallySlowSearchForIndex("abc"));

}

// Добра практика

int index = reallySlowSearchForIndex("abc");

if (index >= 0) {

remove(index);

}

## Функції та процедурне проектування

Добре спроектована функція має такі характеристики:

* Повністю виконує чітко поставлену задачу;
* Не бере на себе занадто багато роботи;
* Не зв‘язана з іншими функціями без певної мети;
* зберігає дані максимально стисло;
* Допомагає  розпізнати та розділити структуру програми;
* Допомагає позбавитися від надлишків, які в іншому випадку були б присутні в програмі.

Використовуйте параметри, щоб відправляти інформацію з функції або коли функції потрібно повертати декілька значень. Не використовуйте параметри без необхідності. Зауважте, що a,b, и с не є параметрами в нижче наданій функції, так як це не потрібно:

/\*

\* Рішаємо квадратне рівняння ax^2 + bx + c = 0,

\* вносячи результати в root1 и root2.

\* Передбачається, що дане рівняння має два кореня.

\*/

void quadratic(double a, double b, double c,

double& root1, double& root2) {

double d = sqrt(b \* b - 4 \* a \* c);

root1 = (-b + d) / (2 \* a);

root2 = (-b - d) / (2 \* a);

}

Коли потрібно повернути значення з функції, використовуйте значення return:

// Погана практика

void max(int a, int b, int& result) {

if (a > b) {

result = a;

} else {

result = b;

}

}

// Добра практика

int max(int a, int b) {

if (a > b) {

return a;

} else {

return b;

}

}

Передаючи об‘єкт в функцію як параметр, ви повинні передавати його по посиланню, оскільки якщо він буде переданий як значення, то буде скопійований весь об‘єкт. Копіювання об‘єктів потребує більших витрат пам‘яті.

Використовуйте змінні-посилання, а не покажчики. Одна з причин — це те, що змінні-посилання, на відміну від покажчиків, не можуть приймати значення NULL:

// Погана практика

// Приймає покажчик

void process(BankAccount\* account) {

...

}

// Добра практика

// Приймає адресне посилання

void process(BankAccount& account) {

...

}

Якщо ви передаєте об‘єкт в функцію і код не змінить виду об‘єкта — передайте його як const-посилання:

// Погана практика

// Приймає покажчик

void display(BankAccount account) {

...

}

// Добра практика

// Приймає константне посилання

void display(const BankAccount& account) {

...

}

Уникайте “ланцюгових” викликів, коли  багато функцій викликають одна одну по ланцюгу, не повертаючи значення в main. Впевнитись, що main є коротким описом всієї програми:

// Погана практика

main

|

+-- function1

|

+-- function2

|

+-- function3

|

+-- function4

|

+-- function5

|

+-- function6

// Добра практика

main

|

+-- function1

|

+-- function2

| |

| +-- function3

| |

| +-- function4

|

+-- function5

| |

| +-- function6

Посилання на документацію по гіт-хабу

<https://docs.github.com/ru/get-started/onboarding/getting-started-with-github-team>

<https://git-scm.com/doc>

<https://habr.com/ru/articles/541258/>