

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Кафедра програмної інженерії

C++ Coding Conventions

Доповідь виконав:

студент III курсу

групи ПЗП-22-2

Береговий Даніїл Олександрович

Перевірив

Доц. кафедри ПІ

Лещинський Володимир Олександрович

Мета роботи

Вивчити основні рекомендації щодо написання коду мовою C++ та представити їх у вигляді презентації.

Хід роботи:

Я прочитав C++ Coding Standards та кілька інших статей на офіційних ресурсах C++ і створив презентацію.

C++ розвивався як мова, яка поєднує можливості низькорівневого управління пам'яттю та високорівневого програмування. Вона має статичну типізацію, але дозволяє створювати шаблони (templates), що надає велику гнучкість при написанні коду. Завдяки цьому C++ дає змогу розробникам створювати продуктивні рішення.

Мова підтримує маніпуляції з типами за допомогою перевантаження операторів, шаблонів і інших механізмів, що дозволяють досягти високої ефективності.

Крім того, C++ використовує вказівники, які можуть бути нульовими, даючи більше контролю, але також вимагають обережності, оскільки неправильне використання може призвести до критичних помилок, таких як помилки сегментації (segmentation fault).

Основні рекомендації для написання коду на C++ акцентують увагу на досягненні максимальної ефективності, продуктивності та зрозумілості, враховуючи специфіку та можливості мови.

Висновки:

C++ надає розробникам великий контроль над процесами виконання завдяки можливостям низькорівневого управління пам'яттю та використанню шаблонів для створення гнучких і ефективних рішень. Рекомендації по написанню коду зосереджені на досягненні високої продуктивності та підтримці зрозумілості коду. Це дозволяє досягти балансу між гнучкістю, ефективністю та простотою у розробці складних програмних систем.

Best Practices for Writing Code in C++

- Виконав: Береговий Даніїл Олександрович
- Група: ПЗПІ-22-2
- Перевірів: Лещинський Володимир Олександрович

Вступ

- C++ - одна з найпоширеніших мов програмування.
- Вона необхідна для розробки високопродуктивного програмного забезпечення.
- Дотримання найкращих практик має вирішальне значення для написання чистого та зручного для обслуговування коду.









Структура коду

- Використовуйте змістовні імена для змінних і функцій.
- Коментарі повинні пояснювати, що робить код, а не як він працює.
- Форматування: використовуйте відступи, вирівнювання та порожні рядки.

Приклад:

```
int calculateSum(int a, int b) {  
    return a + b; // Returns the sum  
}
```





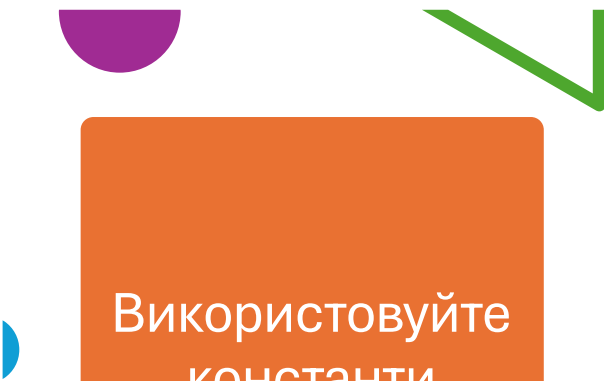
Уникайте дублювання коду

- Виділіть код, що повторюється, в окремі функції.
- Повторне використання промокоду.

Приклад:

```
void printGreeting(string name) {  
    cout << 'Hello, ' << name << '!'  
    << endl;  
}
```






Використовуйте константи

- Уникайте магічних чисел у кодї.
- Замість цього використовуйте константи.

Приклад:

```
const double PI = 3.14159;  
double calculateCircleArea(double  
radius) {  
    return PI * radius * radius;  
}
```




Керування пам'яттю

- Уникайте витоків пам'яті.
- Використовуйте `delete` для звільнення динамічно виділеної пам'яті.

Приклад:

```
int* ptr = new int(10);  
delete ptr; // Free memory
```





Використання стандартної бібліотеки

- Приклад:

```
vector<int> numbers = {1, 2, 3, 4, 5};
for (int num : numbers) {
    cout << num << ' ';
}
```

- Завжди передбачайте потенційні помилки .
- Використовуйте блоки з перехопленням для обробки помилок .

Приклад:

```
try {  
    if (b == 0) throw runtime_error("Division by  
zero!");  
} catch (const exception& e) {  
    cerr << "Error: " << e.what() << endl;  
}
```

- Завжди передбачайте потенційні помилки .
- Використовуйте блоки з перехопленням для обробки помилок .

Приклад:

```
try {
    if (b == 0) throw runtime_error("Division by
zero!");
} catch (const exception& e) {
    cerr << "Error: " << e.what() << endl;
}
```



Тестування коду

- Тестуйте кожну функцію окремо.
- Використовуйте фреймворки для модульного тестування для перевірки функціональності.

Приклад:

```
assert(add(2, 3) == 5);
```

```
assert(add(-1, 1) == 0);
```

Висновки



- ДОТРИМАННЯ НАЙКРАЩИХ ПРАКТИК ПОКРАЩУЄ ЧИТАБЕЛЬНІСТЬ КОДУ ТА ЗРУЧНІСТЬ ЙОГО СУПРОВОДУ.



- ПРОГРАМУВАННЯ- ЦЕ І НАВИЧКА, І МИСТЕЦТВО, ЯКЕ ВИМАГАЄ ДИСЦИПЛІНИ.

Список використаних джерел



1. STROUSTRUP B. THE C++
PROGRAMMING LANGUAGE.



2. OFFICIAL DOCUMENTATION
WEBSITES: CPPREFERENCE.COM,
CPLUSPLUS.COM.



3. ONLINE COURSES AND
TUTORIALS ON COURSERA.