

Харківський національний університет радіоелектроніки  
Факультет комп'ютерних наук  
Кафедра програмної інженерії  
ЗВІТ  
з дисципліни "Аналіз та рефакторинг коду"  
до практичної роботи №1  
на тему: " Методи рефакторингу коду програмного забезпечення"

Виконав ст. гр. ПЗП-22-2  
Д'яченко Микита Олександрович

Доц. кафедри ПІ  
Лещинський Володимир Олександрович

Харків 2024

Мета роботи Дослідити основні рекомендації щодо написання коду мовою С і представити їх у вигляді презентації.

Хід роботи: Я прочитав основні рекомендації щодо написання коду на мові С з офіційних джерел та кілька інших статей, і створив презентацію. Загалом, С є однією з найстаріших та найвпливовіших мов програмування, що вплинула на багато сучасних мов. Мова підтримує статичну типізацію, що допомагає уникати помилок у типах під час компіляції. Однією з особливостей є необхідність явного визначення типу кожної змінної. Також важливим аспектом є управління пам'яттю, що виконується вручну за допомогою функцій `malloc` та `free`. Це надає більше контролю, але також вимагає уважності, щоб уникнути помилок, таких як витоки пам'яті чи подвійне звільнення пам'яті. Конвенції мови С спрямовані на забезпечення простоти, читабельності та ефективності коду, що особливо важливо в системному програмуванні.

Висновки: Синтаксис С, хоч і більш базовий, ніж у сучасних мов, забезпечує високу продуктивність і дозволяє створювати ефективний код. Дотримання конвенцій допомагає писати код, який легше підтримувати та розуміти іншим програмістам.

ДОДАТОК А  
Слайди презентацій

# ОСНОВНІ РЕКОМЕНДАЦІЇ НАПИСАННЯ КОДУ НА МОВІ C

Д'яченко Микита Олександрович  
ПЗПІ-22-2

```
#include <stdio.h>

int main() {
    printf("Hello, World!\n");
    return 0;
}
```

# План презентації

- 01 • Чому важливо отримуватися правил написання коду?
- 02 • Використання змістовних імен
- 03 • Рекомендації щодо функцій
- 04 • Форматування коду
- 05 • Обробка помилок
- 06 • Тестування
- 07 • Висновки



# Чому важливо отримуватися правил написання коду?

---

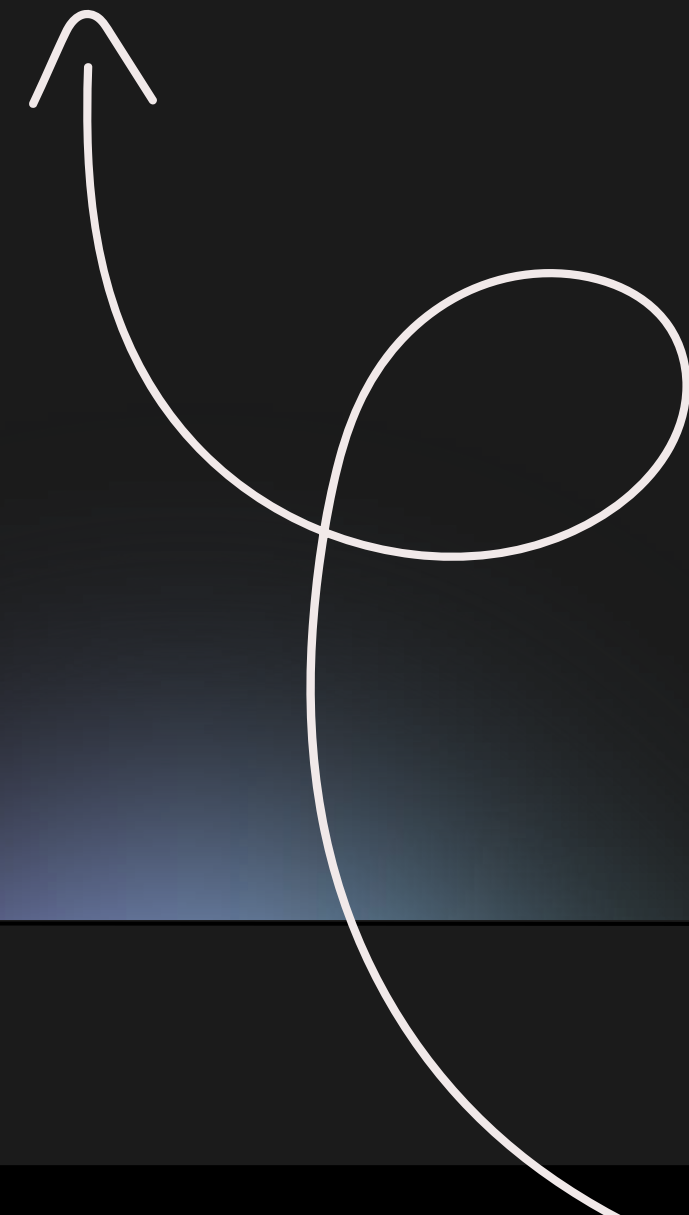
- Правильний код є основою якісного програмного забезпечення.
- Дотримання рекомендацій допомагає уникнути помилок та полегшує підтримку.
- Мета: забезпечити простоту, зрозумілість і надійність коду.

```
if (bmi < 18.5) {  
    printf("You are underweight.\n");  
} else if (bmi >= 18.5 && bmi < 24.9) {  
    printf("You have a normal weight.\n");  
} else if (bmi >= 25 && bmi < 29.9) {  
    printf("You are overweight.\n");  
} else {  
    printf("You are obese.\n");  
}
```

# Використання ЗМІСТОВНИХ ІМЕН


```
int hoursWorked = 40; // Змістовне ім'я  
int hw = 40;           // Поганий приклад
```

- Імена змінних, функцій і класів повинні відображати їхню суть.
- Уникайте скорочень і неоднозначності.



# Рекомендації щодо функцій

- ФУНКЦІЇ ПОВИННІ ВИКОНУВАТИ ОДНУ ЗАДАЧУ.
- ВИКОРИСТОВУЙТЕ ОПИСОВІ НАЗВИ ДЛЯ ФУНКЦІЙ.
- ОБМЕЖТЕ КІЛЬКІСТЬ ПАРАМЕТРІВ (ДО 3-4).



```
// Функція для розрахунку суми
int calculateSum(int a, int b) {
    return a + b;
}

int main() {
    int num1 = 5, num2 = 10;
    printf("Sum: %d\n", calculateSum(num1, num2));
    return 0;
}
```





# Форматування коду

- Дотримуйтесь єдиного стилю відступів і розташування блоків.
- Розділяйте логічні блоки порожніми рядками.
- Використовуйте круглі дужки для покращення читабельності.

```
#include <stdio.h>

int main() {
    int number = 10;

    if (number > 0) {
        printf("Positive\n");
    } else {
        printf("Non-positive\n");
    }

    return 0;
}
```

## Обробка помилок

- Використовуйте винятки для критичних помилок.
- Не залишайте помилки необробленими.
- Завжди додавайте пояснення до повідомлень про помилки.

```
#include <stdio.h>
#include <stdlib.h>

int readFile(const char *filename) {
    FILE *file = fopen(filename, "r");
    if (file == NULL) {
        perror("Error opening file");
        return -1;
    }

    // Обробка файлу
    fclose(file);
    return 0;
}

int main() {
    if (readFile("data.txt") != 0) {
        printf("Operation failed\n");
    }
    return 0;
}
```



# Тестування

- Написання тестів забезпечує стабільність програми.
- Перевіряйте всі крайні випадки та можливі помилки.
- Дотримуйтесь принципу "один тест — одна перевірка".

```
#include <assert.h>

int add(int a, int b) {
    return a + b;
}

void testAdd() {
    assert(add(2, 3) == 5);
    assert(add(-1, 1) == 0);
    assert(add(0, 0) == 0);
}

int main() {
    testAdd();
    printf("All tests passed!\n");
    return 0;
}
```



# Висновки

Дотримання рекомендацій полегшує підтримку та розвиток проекту.

Чистий код підвищує продуктивність команди.

Інвестування часу у якість коду окупається у довгостроковій перспективі.



**Дякую за увагу!**