# Zaprzyjaźnij się z kompilatorem

## Krótki przewodnik po flagach

Zaawansowane metody optymalizacji mogą przyczynić się do generowania trudnych do wykrycia błędów, jeśli kod wejściowy nie jest napisany zgodnie ze standardem. Wykrycie źródła nieprawidłowości może być trudne i zależne od wielu czynników, np. wersji kompilatora i stopnia optymalizacji. Część problemów można wyeliminować, korzystając z rozbudowanej diagnostyki, jaką oferują kompilatory.

Standard języka C zakłada, że pewne operacje są niezdefiniowane (np. dzielenie przez zero) i nie mogą one wystąpić w prawdziwym kodzie. Twórcy kompilatorów wykorzystują ten fakt w celu opracowywania coraz bardziej wyrafinowanych metod optymalizacji kodu. Zakładając, że opisane w standardzie sytuacje nie mogą mieć miejsca, kompilator może rozszerzyć obszar optymalizacji i wygenerować bardziej wydajny kod wynikowy. Warto zaznaczyć, że kompilator nie nie musi wykrywać wyrażeń niezdefiniowanych podczas procesu kompilacji. Według standardu to programista jest zobowiązany do tworzenia poprawnego kodu. Jeśli programista popełni błąd i dopuści do pojawienia się operacji niezdefiniowanej, to kompilator ma pełną dowolność, w jaki sposób taki kod zostanie przetworzony. Sposób obsługi niepoprawnego kodu wejściowego może zależeć od wielu czynników: wersji kompilatora, stopnia optymalizacji czy architektury procesora. Istnieje szansa, że w pewnych warunkach skompilowany program będzie zachowywał się zgodnie z intencją programisty, podczas gdy na innej platformie będą obserwowane nieskorelowane błędy. W konsekwencji wykrycie przyczyny może być żmudne i czasochłonne. Na szczęście można ograniczyć ryzyko zaimplementowania niedozwolonej operacji poprzez dodanie odpowiednich opcji diagnostycznych na etapie budowania i testowania kodu.

## KORZYŚCI Z ROZBUDOWANEJ DIAGNOSTYKI

Dodatkowe opcje diagnostyczne pozwalają również na szybkie wykrycie sytuacji, w których kod wejściowy jest napisany zgodnie ze standardem, niemniej jednak jest on niewłaściwie sformatowany i może wprowadzać w błąd. Przykład takiego kodu przedstawiono w Listingu 1.

**Listing 1. Sytuacja, w której błąd logiczny jest trudny do zauważenia ze względu na formatowanie kodu**

```
int isEven(unsigned int number) {
    int res = 0;
    if (number % 2)
        res = 0;
    //nadmiarowy średnik
    else;
        res = 1;
    return res;
}
```

Nadmiarowy średnik przy słowie kluczowym `else` powoduje, że funkcja zawsze zwraca wartość 1 niezależnie od wartości parametru `number`. Jest to mylące, ponieważ czytając jedynie nazwę funkcji, można zakładać, że wartość zwracana przez tę funkcję będzie zależała od tego, czy przekazany argument jest liczbą parzystą. Kompilator GCC 11.1 domyślnie nie zwróci uwagi, że kod jest sformatowany w sposób mylący dla człowieka, i nie wygeneruje ostrzeżenia dotyczącego możliwej pomyłki, ponieważ kod z Listingu 1 jest napisany zgodnie z wymaganiami standardu. Dodanie flagi `-Wmisleading-indentation` przy kompilacji za pomocą GCC 11.1 powoduje wygenerowanie ostrzeżenia, że formatowanie kodu z tego listingu nie zgadza się z drzewem składniowym, jakie odpowiada analizowanej funkcji. Taka niezgodność może być przyczyną przeoczenia błędu logicznego z Listingu 1. Dzięki dodatkowej informacji programista może zmodyfikować swój kod, tak aby był on czytelniejszy.

## WALL, WEXTRA, WPEDANTIC?

Kompilatory oferują wiele flag sterujących szczegółowością ostrzeżeń, które mogą być przekazane podczas procesu budowania. Część opcji jest włączona domyślnie, inne muszą zostać wywołane przez programistę. GCC i Clang klasyfikują najpopularniejsze opcje w dwie grupy, które można włączyć, przekazując kompilatorowi flagi `-Wall` lub `-Wextra`. Te dwie opcje zawierają zbiory typów ostrzeżeń, które zostały uznane za pomocne przy wykrywaniu błędów w kodzie. Opcja `-Wall` sprawia, że kompilator próbuje wykryć sytuacje, które mogą wynikać z przypadkowej pomyłki programisty i jednocześnie nie zakłócają procesu kompilacji. Te ostrzeżenia stanowią informacje dla programisty, że dane fragmenty kodu mogą zawierać ukryte błędy logiczne i powinny zostać uważnie sprawdzone pod kątem ich zgodności z oczekiwaną funkcjonalnością. Flaga `-Wextra` obejmuje dodatkowe opcje, które pozwalają rozszerzyć zakres sprawdzania możliwych pomyłek [1].

Flagi `-Wall` i `-Wextra` są respektowane przez kompilatory Clang i GCC [1] [2]. Zakres sprawdzania kodu różni się jednak zarówno pomiędzy konkurującymi ze sobą projektami, jak i poszczególnymi wersjami tego samego projektu. Sprawdzanie kodu nie tylko pod kątem zgodności z regułami gramatycznymi języka, ale także pod kątem możliwych pomyłek programistycznych jest intensywnie rozwijane. Z tego powodu nowsze wersje kompilatorów mogą oferować