BO·TT·EGA
IT minds

Program szkolenia:

Zaawansowane programowanie w C++

Informacje:

Nazwa:	**Zaawansowane programowanie w C++**
Kod:	**ccpp-C++ Advanced**
Kategoria:	C i C++
Grupa docelowa:	developerzy
Czas trwania:	3 dni
Forma:	40% wykłady / 60% warsztaty

Zaawansowane szkolenie C++, które pozwala zrozumieć od podszewki zaawansowane elementy składni języka C++. W nieksiążkowy i nieszablonowy sposób, bazując na użytecznych idiomach i wzorcach kolejno odsłaniane są zaawansowane niuanse języka.

Atutem szkolenia jest duży nacisk na wykorzystywanie dobrych praktyk i dostępnych narzędzi/bibliotek do: sprawdzania poprawności kodu mierzonej w najważniejszych wymiarach: poprawność, stabilność, jakość i przenośność, oraz dzielenia się wiedzą w zespole.

Szkolenie przeznaczone jest dla programistów dobrze znających podstawową składnię C++, chcących usystematyzować lub poszerzyć swoja wiedzę o zaawansowanych mechanizmach oraz chcących poznać szereg istniejących bibliotek i narzędzi podnoszących wartość dostarczanego kodu.

Zalety szkolenia:

- Praktyczne podejście do składni języka
- Nacisk na bezpieczną implementację opartą na wzorcach i testowaniu
- Narzędzia wspomagające
- Rzeczywiste przykłady

BO·TT·EGA
IT minds

Szczegółowy program:

1. Systematyzacja

1.1. Niskopoziomowy C++

1.1.1. Natura języka C++

1.1.2. Główne problemy

1.1.3. Zarządzanie pamięcią - co mówi nam standard?

1.1.3.1. Programowanie systemowe i kod przenośny (pliki, współbieżność, IO, ...)

1.1.3.2. Kod legacy

1.2. OOP – Object Oriented Programming – przypomnienie podstawowych zasad programowania obiektowego w świetle rzeczywistych problemów.

1.2.1. Paradygmat programowania obiektowego

1.2.1.1. Analiza paradygmatu programowania obiektowego i jego poprawna interpretacja

1.2.1.2. GRASP – General Responsibility Assignment Software Patterns (Principles).

1.2.1.3. SOLID – Single Responsibility Principle (SRP), the Open/Closed Principle (OCP), the Liskov Substitution Principle (LSP), the Dependency Inversion Principle (DIP), and the Interface Segregation Principle (ISP).

1.2.2. DID – Defense In Depth – programowanie defensywne

1.2.3. POSA – Pattern Oriented Software Architecture – architektura oparta na wzorcach

1.2.3.1. Wzorzec – czym jest?

1.2.3.2. Pattern language – architektura rozwiązania (rozwiązań) oparta na wzorcach

1.2.3.3. Framework – zbiór reużywalnych, zaimplementowanych wzorców i funkcjonalności

2. Zbiór dobrych praktyk i narzędzi wspomagających - niezbędnik profesjonalisty

2.1. Testowanie punktem wyjścia do pisania zdrowego kodu

2.1.1. TDD - Test Driven Development

BOTTEGA IT minds

2.1.2. Biblioteki testów modułowych/jednostkowych: Gtest, CppUnit, BoostTest

2.2. Gdb - debugger wyjadaczy

2.3. Valgrind

2.3.1. Memcheck - sprawdzanie poprawności dynamicznego zarządzania pamięcią

2.3.2. Ptrcheck - śledzenie wskaźników

2.3.3. Helgrind - sprawdzanie aplikacji wielowątkowych

2.3.4. Callgrind - profilowanie aplikacji

2.4. Strace - debugowanie na poziomie wywołań systemowych

2.5. Code review - proces SMART

3. Zaawansowane elementy składni

3.1. Rozwinięcie dziedziczenia

3.1.1. „Las” dziedziczenia

3.1.1.1. Konstrukcja obiektu

3.1.1.2. Funkcje wirtualne od podszewki

3.1.1.3. „Wodospad” destrukcji

3.1.2. „Dżungla” wielokrotnego dziedziczenia

3.1.2.1. Problematyka

3.1.2.2. Zastosowanie

3.1.3. „Szczyty” wirtualnego dziedziczenia, czyli wyższa szkoła jazdy

3.1.3.1. Motywacja dziedziczenia wirtualnego

3.1.3.2. Mechanika funkcjonowania wirtualnego dziedziczenia

3.1.3.3. Praktyczne zastosowanie

3.1.4. Aspekt rzutowania i niejawnego przekształcania typów

3.2. Programowanie generyczne, czyli „labirynt szablonów”

BO·TT·EGA
IT minds

3.2.1. Motywacja dla programowania generycznego – aspekt programowania funkcyjnego

3.2.2. Podstawowa składnia

3.2.2.1. Funkcje szablonowe

3.2.2.2. Klasy szablonowe

3.2.2.3. Szablony specjalizowane

3.2.2.4. Użycie typów „własnych”

3.2.3. Mechanika szablonów

3.2.3.1. Aspekt kompilacji

3.2.3.2. Moment składania docelowej implementacji

3.2.3.3. Budowa obiektu z szablonu

3.2.3.4. Najczęściej spotykane problemy i najczęściej popełniane błędy

4. Implementacja funkcjonalnych wzorców

4.1. Rozróżnianie dziedziczenia i stosowania szablonów

4.2. Klasy pełnomocników z uwzględnieniem pracy z kodem legacy

4.2.1. Zarządzanie pamięcią głównym problemem C++

4.2.1.1. Problem natywnych wskazań

4.2.1.2. Typowe błędy i ich konsekwencje

4.2.2. Wykorzystanie idiomu RAII

4.2.3. Zliczanie referencji

4.2.3.1. Mechanizm

4.2.3.2. Podstawowy problem i jak sobie z nim radzić

4.2.4. Co już istnieje

4.3. Silniki zdarzeń

4.3.1. Wzorce - przykładowe użycie w domenie wielowątkowości i IO

BOTTEGA IT minds

4.3.1.1. Wrapper-fasada – refaktoryzacja „error prone” i legacy API na rzecz reużywalnych komponentów

4.3.1.2. Proxy – implementacja konkretnych funkcjonalności z użyciem przygotowanych fasad i wrapper’ów

4.3.1.3. Observer, reactor, proactor, half sync/half asyc, thread pool – niby to samo, a jednak inaczej

4.3.1.4. Event broker

4.3.2. Implementacja w oparciu o dziedziczenie

4.3.3. Implementacja szablonowa

4.3.4. Aspekt refaktoryzacji

5. STL – Co warto wiedzieć

5.1. Kontenery

5.1.1. Rodzaje – czyli co z czym się je

5.1.2. Efektywne użycie kontenerów

5.1.3. Aspekt równoległości

5.2. Algorytmy

5.2.1. Co już mamy

5.2.2. Jak poprawnie używać

5.2.3. Boost – wzmocnienie możliwości bez konieczności dopisywania kodu

5.2.3.1. Ważniejsze moduły

5.2.3.2. Bind i funkcje

5.2.3.3. Wskaźniki i referencje Smart

5.2.3.4. Przekształcanie typów

5.2.3.5. Algorytmy i kontenery

5.2.3.6. System plików, ASIO oraz wielowątkowość

5.2.3.7. Wykorzystanie w połączeniu z STL

5.2.3.8. Tworzenie przenośnego kodu

BO·TT·EGA
IT minds

- 5.3. Strumienie
  - 5.3.1. Rodzaje obiektów – poprawna interpretacja
  - 5.3.2. Poprawne użycie
- 5.4. C++11
  - 5.4.1. Co nowego w STL
  - 5.4.2. Zaawansowane elementy nowej składni
    - 5.4.2.1. Automatyczne określanie typu
    - 5.4.2.2. Pętla for oparta na zakresie
    - 5.4.2.3. Funkcje i wyrażenia lambda
    - 5.4.2.4. Aliasy dla szablonów