BOTTEGA IT minds

Program szkolenia:

Dobry start do profesjonalnego programowania w C++ dla początkujących programistów

Informacje:

Nazwa:	**Dobry start do profesjonalnego programowania w C++ dla początkujących programistów**
Kod:	**ccpp-C++ Dobry Start**
Kategoria:	C i C++
Grupa docelowa:	developerzy
Czas trwania:	3 dni
Forma:	50% wykłady / 50% warsztaty

Szkolenie pozwalające zrozumieć od podszewki kontekst oraz podstawową składnię i „ukryte” mechanizmy języka C++. W nieksiążkowy i nieszablonowy sposób, bazując na użytecznych idiomach i wzorcach kolejno odsłaniane są rozmaite niuanse języka.

Zdobyta wiedza w sposób natychmiastowy przekłada się na pełne wykorzystanie języka i jego możliwości. Wyciągnięte z rzeczywistych projektów zadania pomogą zdecydowanie podnieść jakość (w kontekście czystości i bezpieczeństwa), testowalność i elastyczność kodu.

Szkolenie przeznaczone jest dla programistów znajacych podstawowa skladnie C++ chcacych usystematyzowac i poszerzyć swoją wiedzę o tym niebanalnym języku.

Zalety szkolenia:

- Szkolenie obwarowane zbiorem najlepszych praktyk
- Praktyczne podejście do składni języka
- Nacisk na bezpieczną implementację opartą na składni i testowaniu
- Przykłady wyjęte z rzeczywistych projektów

BOTTEGA IT minds

Szczegółowy program:

1. Systematyzacja

- 1.1. Korzenie języka C++
  - 1.1.1. Język C
  - 1.1.2. Nisko-poziomowa natura C++
  - 1.1.3. Kompilacja i linkowanie
    - 1.1.3.1. Rozróżnienie obu pojęć
    - 1.1.3.2. Preprocesor
  - 1.1.4. Sekcje code, text i data oraz stos i sterta
- 1.2. Typy proste - wzmianka pierwsza
  - 1.2.1. Zbiór podstawowych typów prostych i ich poprawne wykorzystanie z uwzględnieniem problematyki signed/unsigned
  - 1.2.2. Wsoko-poziomowe spojrzenie na pamięć
    - 1.2.2.1. Rozróżnienie deklaracji i definicji
    - 1.2.2.2. Pojęcie obiektu i instancji już na etapie typów prostych
    - 1.2.2.3. Kopia, wskaźnik i referencja
    - 1.2.2.4. Problem zarządzania pamięcią z uwzględnieniem aspektu “legacy code”
- 1.3. OOP - Object Oriented Programming - programowanie obiektowe
  - 1.3.1. Paradygmat programowania obiektowego[a][b] w praktyce
    - 1.3.1.1. Analiza paradygmatu programowania obiektowego i jego poprawna interpretacja
    - 1.3.1.2. GRASP - General Responsibility Assignment Software Patterns (Principles).
    - 1.3.1.3. 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).
  - 1.3.2. Typy złożone po raz pierwszy

BO·TT·EGA IT minds

1.3.2.1. Klasa a struktura

1.3.2.2. Klasa/struktura a obiekt

2. Chemia w C++ - wszystko na temat tworzenia i związków

2.1. Punkt absolutnie podstawowy - organizacja pracy

2.1.1. Wersjonowanie kodu - absolutna konieczność

2.1.1.1. Systemy kontroli wersji (SVN, Git)

2.1.1.2. Struktura repozytorium: podsystemy, komponenty, moduły...

2.1.2. Budowanie kodu

2.1.2.1. Single point of fire - budowanie i uruchamianie z jednego miejsca

2.1.2.2. Narzędzie wspomagające - Makefile

2.1.2.3. Zależności małe i duże

2.1.2.4. Problem zależności między komponentami i jak go przełamać

2.1.2.5. Problem zależności "inkludów" i jak go pokonać

2.1.3. Myślenie i kodowanie w czasie przyszłym

2.1.3.1. TDD - Test Driven Development

2.1.3.2. Kod reużywalny

2.2. Typy proste - czy naprawdę takie proste?

2.2.1. Problematyka jawnego i niejawnego przekształcania typów prostych - jak sobie z tym radzić

2.2.2. Referencje i wskaźniki po raz drugi

2.2.2.1. Ujecie zmiennych lokalnych i globalnych

2.2.2.2. Wskaźnik i tablica – czy to się da rozróżnić?

2.2.2.3. Argumenty funkcji – API wymuszające, optymalne i bezpieczne

2.3. Typy złożone wzmianka druga

2.3.1. Składowe

2.3.1.1. Member - o tym co obiekt ma i jak to wygląda w pamięci

BO·TT·EGA IT minds

2.3.1.2. Funkcje - czyli co obiekt robi i nawiązanie do sekcji kodu

2.3.2. Konstrukcja podstawą obiektu i jego poprawności

2.3.2.1. Mechanika budowy obiektu

2.3.2.2. Konstruktor jednoargumentowy i niebezpieczeństwo niejawnego przekształcenia typów (wstęp do kopiowania)

2.3.2.3. Poprawność obiektu po konstrukcji

2.3.2.4. Metody konstrukcji obiektów w świetle wzorców i możliwości składniowych

2.4. Kopiowanie - zmora wymagań wydajnościowych

2.4.1. Mechanika kopiowania typów prostych, wskaźników i referencji

2.4.2. Kopiowanie typów złożonych od strony składni

2.4.3. Problemy z kopiowaniem klas i struktur

2.5. Lawina destrukcji

2.5.1. Zakres obiektów - nawiązanie do stosu i o sterty

2.5.2. Problem delete

2.5.3. Panowanie nad poprawną destrukcją i wzorce wspomagające

3. Więcej o obiektach

3.1. Zasięg

3.1.1. Rodzaje i znaczenie w rozumieniu zasad OOP (wstęp do dziedziczenia)

3.1.2. Przyjaźń – piękno a zarazem pułapki

3.1.3. Zasięg kluczowych składowych

3.2. Osobliwości inne

3.2.1. const vs mutable – pułapki

3.2.2. volatile – w czym (nie) pomaga

4. Dżungla dziedziczenia i polimorfizm

4.1. Po co jest dziedziczenie?

BOTTEGA IT minds

BOTTEGA IT minds

5.2.4.1. Value object

5.2.4.2. RAII

5.2.4.3. Zarządzanie pamięcią

5.2.4.4. Coś o szablonach

6. Wyjątki i obsługa błędów

6.1. Konieczność destrukcji

6.2. Poprawna konstrukcja raz jeszcze

6.3. Wyjątki w destruktorach

6.4. Różnica pomiędzy wyjątkiem a przekazaniem argumentu lub wywołaniem funkcji wirtualnej

6.5. Chwytanie wyjątków i rozważne stosowanie specyfikacji sytuacji wyjątkowych

6.6. Koszty obsługi wyjątków

7. Podsumowanie: problemy i rozwiązania

7.1. Jak radzić sobie ze wskaźnikami wykorzystując podejście imperatywne

7.1.1. RAII

7.1.2. Zliczanie referencji

7.1.3. Obiekty podobne do natywnych wskaźników (klasy pełnomocników)

7.2. STL – to co już jest

7.2.1. std::auto_ptr

7.2.2. std::string

7.2.3. Kontenery i algorytmy

7.2.4. C++11

7.3. Katalog bibliotek wspomagających