Wojciech Olech (2017)

Część #0: Hello, world

Ta część kursu jest poświęcona opisowi języka, oraz ustawieniu środowiska do jego nauki oraz programowania. Zawiera również krótkie FAQ dotyczące zarówno języka, jak i samego kursu.

Krótkie FAQ na start

Dla kogo przeznaczony jest kurs

Kurs jest przeznaczony dla osób, które zaczynają zabawę z C++em, lub też ją zaczęli i chcą się doszkolić oraz powtórzyć informacje, ewentualnie - być może - nauczyć czegoś nowego. Nie daję jednak gwarancji, że osoba nie mająca wcześniej styczności z programowaniem zrozumie ten kurs za pierwszym razem w stu procentach.

Dla kogo jest przeznaczony C++

Zacznijmy od tego, że C++ jest jednym z trudniejszych języków, z jakimi można się spotkać - głównie ze względu na jego filozofię, polegającą na tym, że język daje nam na bardzo dużo kontroli - co wiąże się z tym, że nie jest trudno strzelić sobie w stopę. Jest to język dość ogólnego zastosowania - możemy w nim pisać wiele rzeczy na wiele różnych platform. Jeśli sami z siebie nie mamy powodu żeby uczyć się C++'a, lub też zaczynamy dopiero programowanie, sugeruję przemyśleć opcję nauki innego, prostszego języka - na przykład Pythona. Jeśli jednak miałeś już styczność z programowaniem, lub chcesz rzucić się na głębokie wody, potrzebujesz szybkiego i wydajnego języka dającego ci dużą kontrolę nad kodem, jednocześnie pozwalającego pisać wieloparadygmatowo oraz jesteś gotowy spędzić duuużo czasu na jego naukę - ten język może być dla ciebie.

Słowem wstępu

Tworzę ten kurs głównie ze względu na to, że brakuje w internecie porządnych kursów nowoczesnego języka C++, które byłyby przeznaczone dla szerokiej grupy odbiorców. Oczywistą alternatywą są kursy angielskie, strony z referencjami oraz książki, lecz nie każdy chce od razu wydawać pieniądze (ze względu na to, że język może mu po prostu nie przypaść do gustu), a kurs w ojczystym języku zawsze będzie trochę bardziej zrozumiały. Uprzedzam, że mimo tego że kurs jest stricte polski, mogę używać w nim terminów angielskich ponieważ są one niekiedy ładniejsze oraz bardziej zrozumiałe od polskich translacji. Jeśli nie znasz w tej chwili angielskiego na poziomie, który pozwoli ci czytać bez większych problemów – sugeruję zacząć się go uczyć (albo praktykować skillsy w używaniu Google Translate), bo większość materiałów, dokumentacji i poradników jest anglojęzyczna.

Będę starał się tłumaczyć wszystko tak łopatologicznie jak to tylko możliwe, używając dużej ilości przykładów i kodu. Jeśli uważasz że coś pominąłem, niejasno wytłumaczyłem, lub czegoś nie rozumiesz – skontaktuj się ze mną, chętnie pomogę i ewentualnie poprawię kurs.

O formie kursu

Kurs postaram się prowadzić w dość dowolnej i luźnej formie – taka wydaje mi się najbardziej stosowna dla większości osób. Będę starał się tłumaczyć zagadnienia tak szeroko, jak to możliwe – oczywiście w granicach

rozsądku, używając terminologii programistycznej, lecz dla rzeczy które mogą nie być do końca zrozumiałe na pierwszy rzut oka będę starał się również dopisywać uproszczenia. Mogę też nawiązywać do rzeczy, które niekoniecznie są ściśle związane z tematem danej części kursu, niemniej jednak warto wiedzieć w momencie czytania jej, o co z nimi chodzi – przynajmniej pobieżnie. Takiego typu informacje będę umieszczał w osobnych mini-akapitach, lub spoilerach, a w przypadku rzeczy których opisy w momencie czytania mogą nie być w pełni zrozumiałe, będę wstawiał opis *tl;dr*, który w jednym zdaniu powie o sensie istnienia lub użycia danego elementu języka. Do każdej części kursu postaram się dodawać ćwiczenia, które pomogą utrwalić wiedzę z danej części, oraz części poprzednich. Przykłady kodu oraz ćwiczenia będą umieszczanie (ze względu na wygodę i walory wizualne) jako Gisty.

Opis języka

Zakładam, że jeśli czytasz ten kurs, chcesz się nauczyć nie tylko samego języka, lecz również programowania. Jednak, żeby programować trzeba znać język w jakim piszemy kod. C++ jest językiem stworzonym w latach 80 przez Bjarne'a Strostrupa, jako obiektowy następca języka C. Język żyje po dziś dzień, nadal rozwijany – wprowadzane są coraz to nowsze standardy, dające więcej możliwości i aktualizujące język. Najnowszym standardem jest C++17, aktualnie jest on w stanie finalizacji standardu (od marca 2017 roku, został zatwierdzony około września). Z racji tego, że kompilatorów języka jest kilka, standaryzacja pozwala na zachowanie spójności między nimi – jeśli piszemy kod na jeden kompilator, to na 99% będziemy w stanie go skompilować na innym, chyba ze korzystamy z featuresów dostępnych wyłącznie na danym kompilatorze.

Z głównych cech C++'a chciałbym wyróżnić:

- **Wieloparadygmatowość**: paradygmat to pewien wzorzec programowania jakim się posługujemy pisząc kod. C++ obsługuje kilka paradygmatów, takich jak na przykład programowanie strukturalne, obiektowe czy też generyczne (o czym powiem później), co pozwala nam uzyskać swobodę w pisaniu kodu.

- **Kompilowalność bezpośrednio do kodu maszynowego**: Innymi słowy, żeby uruchomić program, musimy go przepuścić przez kompilator który nam przetłumaczy kod z języka C++ na język komputera, co pozwala na uruchomienie go. Wiąże się to też z tym, że C++ jest dość wydajny – nie mamy tutaj żadnych pośredników, pokroju interpreterów *(co występuje w językach interpretowanych)*, ani wirtualnych maszyn *(które występują w językach typu C# i Java)*, z drugiej strony jednak mamy troszkę ostrzejsze zasady pisania kodu i nieco mniejszą elastyczność.

- **Statyczne typowanie**: Temat ten rozwinę w następnej części kursu, ale teraz warto o tym wspomnieć – statyczne typowanie oznacza że wszystko w C++ (zmienne, obiekty) musi mieć swój określony z góry typ – czyli podczas pisania kodu musimy określić że ta zmienna trzyma w sobie liczby, ta tekst, a ta wartości logiczne i nie można tego potem zmienić. Innym rodzajem typowania, występującym na przykład w Pythonie jest typowanie dynamiczne – tworzymy zmienną, ustawiamy jej wartość, i automatycznie jest ustawiany jej typ. Przy zmianie wartości typ może się zmienić. Z jednej strony to wygodne, a z drugiej może prowadzić do nieprzewidzianych konsekwencji i troszkę zmniejsza wydajność. Co prawda, od C++11 można nieco nagiąć zasady statycznego typowania, ale nadal jest ono statyczne na poziomie kompilacji. Ale – o tym później.

- **Wysokopoziomowość**: C++ to język wysokiego poziomu – oznacza to, że jego składnia jest stosunkowo prosta dla programisty, posługujemy się głównie słówkami w języku angielskim i nie musimy się martwić o bezpośrednią obsługę sprzętu (chyba że koniecznie chcemy, architektura C++'a daje nam możliwość ingerencji w fizyczny sprzęt do pewnego stopnia!) kompilator robi niskopoziomową robotę za nas.

- **Wieloplatformowość**: Sam język *(włącznie z biblioteką standardową)* jest wieloplatformowy. Oznacza to, że dopóki nie używamy bibliotek, które są zależne od systemu, nasz kod powinien bezproblemowo wykonać się na dowolnej platformie *(Windowsie, Linuxie, iOSie, Androidzie, Arduino, itd.)* oraz architekturze *(i386, amd64, AVR, i686, armv6/7/8, itd.)*, o ile istnieje pod nią kompilator naszego języka. Można śmiało stwierdzić że C++ jest jednym z najbardziej wieloplatformowych języków jaki istnieją (razem z C).

Wybór środowiska

Okej, znamy zarys języka, teraz potrzebujemy miejsca w którym będziemy mogli wygodnie tworzyć nas kod, oraz go uruchamiać. Najwygodniejszym narzędziem będzie **IDE** *(Integrated Development Environment)* – narzędzie, które typowo zawiera w sobie edytor kodu, wsparcie dla kompilatora i systemu budowania *(build system)*, oraz debugera i ewentualnych wtyczek, ułatwiających integrację środowiska z zewnętrznymi narzędziami. Poniżej przedstawię i opiszę krótko kilka darmowych środowisk, które mogą cię zainteresować

Code::Blocks

Code::Blocks jest sympatycznym, multiplatformowym IDE, dobrym dla początkujących. Wygląda nieco topornie, ale jest prosty w obsłudze i działa out-of-the-box. Bardzo popularny wybór wśród początkujących.

Uwaga: Jeśli chcesz z niego skorzystać, upewnij się że pobierasz wersję z MinGW, ponieważ jeśli pobierzesz czystą i nie masz na komputerze żadnego kompilatora, nie będziesz mógł niczego skompilować

Zalety C::B

- **Wieloplatformowe oraz open-source** – działa na Windowsie, Linuxie oraz iOSie, jeśli nie ma go na nasz system/dystrybucję można go bez problemu skompilować.

- **Proste w użyciu** – nie ma zbędnych fajerwerków, działa od razu po instalacji, konfiguracja projektów i kompilatora jest uproszczona do granic możliwości.

- **Posiada szablony projektów dla wielu bibliotek** – jeśli kiedyś zechcemy tworzyć coś, wykorzystując

zewnętrzne biblioteki to Code::Blocks ułatwia konfigurację projektu pod daną bibliotekę, poprzez szablony. Niestety, niektóre są już przestarzałe, dlatego warto tak czy inaczej zajrzeć do dokumentacji danej biblioteki przed rozpoczęciem pracy z nią.

- **Wsparcie dla wtyczek** – jeśli nie mamy jakiejś funkcjonalności, zawsze możemy poszukać wtyczki która nam tą funkcjonalność zaoferuje.

Wady C::B

- **Toporny wygląd** – Wygląd, oraz standardowe ułożenie toolbarów, zakładek i okienek pozostawia trochę do życzenia i nie daje nam maksymalnego pola pracy, ale zawsze można to przekonfigurować.
- **Brak wbudowanego ciemnego motywu** – ciemny motyw to dar z niebios, który powoduje że oczy męczą się znacznie mniej podczas programowania. Niestety, Code::Blocks o nim nie słyszał, i żeby mieć ciemny motyw trzeba trochę pobawić się z konfiguracją.

Visual Studio 2017 (Community)

Visual Studio 2017 w wersji Community przychodzi z kompilatorem Visual C++, oraz ogromnym zestawem narzędzi pomagających testować, optymalizować, debugować i pisać nasz kod. Jako początkujący programista prawdopodobnie nie użyjesz 90% z nich, ale jeśli chcesz mieć trochę większe i bardziej zaawansowane IDE, oraz nie przeszkadza ci brak wieloplatformowości – to może być dobry wybór.

Zalety VS17

- **Duża ilość wbudowanych narzędzi** – Visual przychodzi z bardzo przyjemnym debugerem, obsługą wtyczek, szerokimi możliwościami konfiguracyjnymi projektu, generalnie – czego dusza zapragnie. Każdy znajdzie coś dla siebie, a jeśli nie, to wystarczy poszukać w menadżerze wtyczek.
- **Wysoka konfigurowalność** – Wersja 2017 wprowadziła trochę zmian do procesu instalacji, powodując to, że mamy teraz szerszy wybór tego, co chcemy tak na prawdę zainstalować. A Visual wspiera kilka technologii, które w przyszłym programowaniu mogą nam się przydać, więc jeśli myślimy przyszłościowo to możemy zacząć się nim interesować.
- **Ciemny motyw** – Visual ma bardzo przyjemny dla oka ciemny motyw, dzięki któremu nasze oczy się tak nie męczą, jak podczas używania jasnego motywu.

Wady VS17

- **Waga** – Niestety, ilość rzeczy jakie są w Visuala wbudowane powoduje to, że jest on dość ciężki i nieco powolny. Nic za darmo.
- **Brak wieloplatformowości, poniekąd** – Visual jest niestety przeznaczony tylko i wyłącznie na Windowsa. Co prawda, oferuje pisanie kodu na Linuxa, ale osobiście tego jeszcze nie testowałem, więc nie mogę się wypowiedzieć na temat skuteczności tego rozwiązania.

Qt Creator (Community)

Qt to framework przeznaczony głównie do aplikacji okienkowych w języku C++, ale oprócz tego posiada mnóstwo bibliotek, które bardzo ułatwiają życie. Nas jednak framework sam z siebie na razie nie interesuje, bardziej interesuje nas IDE, które jest częścią tego frameworka – Qt Creator. Jest to bardzo fajne i przyjemne środowisko w którym możemy bez problemu tworzyć aplikacje zarówno w Qt, jak i w czystym C/C++. Prosty debuger, przyjemny edytor, wsparcie dla wtyczek, ładny wygląd, niemałe możliwości konfiguracji, wieloplatformowość i sympatyczny build system – qmake – powodują że jest to mój osobisty numer jeden, jeśli chodzi o środowiska.

Zalety Qt Creatora

- **Wieloplatformowość** – Qt Creator, tak samo jak framework Qt, jest dostępny na Windowsa, Linuxa i Maca.
- **Wygląd** – Qt Creator jest fajny, przejrzysty, i minimalistyczny, co pozwala skupić się na pisaniu kodu.
- **Wsparcie dla GCC oraz VC++, oraz duża ilość wbudowanych wtyczek** – Qt Creator pozwala nam na kompilację zarówno używając Visual C++, jak i GCC *(MinGW, pod Windowsem)*. Oprócz tego ma wbudowane wsparcie dla systemów kontroli wersji (GitHub), formatowania kodu używając Clanga, oraz wielu innych rzeczy – mimo swojego „minimalizmu", jest to dość rozbudowane narzędzie.
- **Własny build system** – qmake jest jednym z najprzyjemniejszych build systemów z jakim się spotkałem. Jest też dobrze udokumentowany. W tej chwili nie będzie cię to za mocno interesować, ponieważ nie będziesz tworzyć żadnych większych projektów, ale w przyszłości wybór build systemu będzie miał znaczenie, a prostota qmake zdecydowanie przemawia za nim.

Wady Qt Creatora

- **Wymaga instalacji frameworka Qt** - Nie jest on wymagany bezpośrednio do działania Qt Creatora, ale bez frameworka Qt nie mamy qmake, czyli musimy używać CMake jako build systemu, który jest o wiele bardziej skomplikowany.

Inne środowiska

Niżej wymienionych środowisk używałem przez krótki okres czasu, dlatego nie jestem w stanie rozwinąć się na ich temat, mimo to zasługują jednak na wzmiankę

- **CodeLite** - nieduże środowisko przystosowane do pracy z PHP, JavaScriptem, C i C++. Jest dość lekkie i przyjemne, wymaga ręcznego zainstalowania kompilatora. Powiedział bym że jest to udoskonalony Code::Blocks. Niestety, miałem lekkie problemy z build systemem tego środowiska, lecz było to tak dawno temu że jest duża szansa na to, że te problemy już nie występują.
- **CLion** - środowisko JetBrains, darmowe w wersji dla studentów i uczniów. Używa CMake jako build systemu, co przekłada się na to że konfiguracja projektów w nim niekoniecznie będzie najprzyjemniejszą rzeczą jaką będziemy musieli robić, ale z drugiej strony renoma środowisk programistycznych od JetBrains (PyCharm, IntelliJ) podpowiada mi że może to być dość dobre IDE.

- **Dev-C++** - z tego IDE nie polecam korzystać. Przestarzałe, niewspierane, oraz jest w stanie przysporzyć BARDZO dużo problemów.

Hello, world

Jeśli już wybrałeś swoje IDE, oraz odpowiednio je skonfigurowałeś – wypada sprawdzić czy działa. Stwórz nowy, pusty projekt, a następnie wrzuć do niego ten kod i spróbuj go skompilować i uruchomić:

```
#include <iostream>

int main() {
    std::cout << "Hello, world!" << std::endl;
}
```

Jeśli widzisz w terminalu napis Hello, world! – gratulacje, skompilowałeś swój pierwszy program i możesz kontynuować naukę!

Natomiast, jeśli coś poszło nie tak, to...

- **Upewnij się że pobrałeś IDE z kompilatorem** – typowo taka sytuacja zdarza się przy zabawie z Code::Blocksem, dlatego opisałem w spoilerze jaką wersję należy wybrać.
- **Upewnij się że skopiowałeś powyższy kod, i nie zgubiłeś niczego po drodze** – każdy znak ma znaczenie
- **Upewnij się że stworzyłeś odpowiedni projekt, oraz że kompilator jest skonfigurowany** – może wybrałeś zły typ projektu? Może IDE nie wykryło kompilatora, i musisz pogrzebać w ustawieniach żeby ustawić odpowiednie ścieżki? Najprościej będzie wygooglować problem (najlepiej po angielsku).
- **Upewnij się że masz prawa zapisu do folderu z projektem** – rzadko się to zdarza, ale czasami problem bierze się stąd, że nie mamy praw do zapisu plików w folderze w którym chcemy stworzyć nasz projekt. Upewnij się że możesz tworzyć w nim pliki.
- **Nie używaj polskich ani specjalnych znaków w nazwach projektów ani plików!** – generalnie, w kursie wszystkie komunikaty będę pisał w języku angielskim i tak samo będę nazywał pliki, ponieważ kompilatory, konsole i IDE mogą nie lubić się z polskimi znakami. Tobie też zalecam takie podejście.
- **Jeśli sprawdziłeś wszystkie powyższe możliwości, i nadal ci coś nie działa – użyj Google**, jest bardzo duże prawdopodobieństwo że ktoś miał już taki problem przed tobą. Szansa na znalezienie odpowiedzi wzrasta kilkukrotnie przy wyszukiwaniu informacji w języku angielskim!

I tym miłym akcentem kończę tą część kursu.

Następna część: Zmienne