Jak przeżyć kurs Podstaw Programowania / Programowania Obiektowego

mgr inż. Łukasz Janiec

Politechnika Wrocławska
Wydział Elektroniki, Fotoniki i Mikrosystemów, Katedra Cybernetyki i Robotyki

rok akademicki 2021/2022

Spis treści

Motywacja

Przygotowanie tej prezentacji jest motywowane Państwa prawdopodobnie jeszcze niewielkim doświadczeniem z programowaniem, bądź jego zupełnym brakiem.
Słowem wstępu - wspomniane w tytule kursy nie uczą Państwa tylko programowania w C i C++, ale całkiem szerokiego zestawu umiejętności i wiedzy potrzebnej inżynierowi pracującemu z wszelkiego rodzaju sprzętem elektronicznym i komputerami, jak też innymi ludźmi.

Chciałem, aby zostały tutaj zebrane ogólne uwagi i porady, jak łatwo i z korzyścią dla siebie przeżyć te zajęcia, co będzie dla Państwa dobrą bazą do tego semestru, kolejnych kursów i Państwa kariery zawodowej w przyszłości.

Jak zwykle wszelkie porady - proszę je przyjmować ze szczyptą soli i za każdym razem myśleć też samodzielnie. Na brak tego nie ma usprawiedliwienia.

Oznajmienie

Osoby, które ze względu na stan zdrowia, niepełnosprawność lub inne obiektywne przesłanki mogą mieć szczególne potrzeby związane ze sposobem realizacji zajęć, zaliczenia bądź przygotowaniem materiałów proszone są o zgłoszenie się na konsultacje lub po zajęciach, napisanie takiej informacji na prywatnym czacie, bądź napisanie e-maila w tej sprawie. Będę starał się, aby na moich zajęciach każdy miał **równe prawo** do zdobycia wiedzy i rozliczenia się z niej.

Umiejętność programowania i wiedza jest zdobywana, nie wrodzona, metodą małych kroków, najlepiej każdego dnia. To trwa (długo), praktycznie nigdy się nie kończy, ale zaczyna być interesująco już po opanowaniu podstaw. Zostanie w tym mistrzem to zadanie na całe życie.

Nauka programowania to także świetna inwestycja czasu, ponieważ programowanie może być bardzo ciekawe, kreatywne (ang. *logic based creativity*), ułatwiające i usprawniające życie, a także po prostu bardzo opłacalne w większości zawodów.

Nie ominie to też Państwa aż do końca tych studiów w żaden sposób, więc warto wejść w to gładko jak najwcześniej. Konkretnie - 4 duże kursy o samym programowaniu (**PProg**, **PO**, PAMSI, SCR cz. 1 i cz. 2), wiele kursów z Matlabem (MUD, TR, SPC...), projekty na różnych zajęciach, gdzie się programuje (od sterowników PLC, przez mikrokontrolery, do aplikacji bazodanowych i algorytmów sterowania robotami), praca inżynierska i potem magisterska.
Sposób myślenia inżyniera nabywany teraz będzie wszędzie podobny.

Garść porad, tylko minimalnie uporządkowana

- *Środowisko do pracy* - zadbaj o skonfigurowanie (najlepiej własnego) komputera z Linuxem (samodzielnym, obok Windowsa/Maca, bądź jako maszyna wirtualna), praca na X2Go i sesji użytkownika na serwerze inyo KCiR jest rozwiązaniem raczej doraźnym do kończenia pracy z zajęć.
- Zainstaluj wybrane *środowisko do pisania kodu* (np. **dobrze skonfigurowany** Emacs, Visual Studio Code, Atom, CLion, Netbeans, Eclipse, CodeBlocks...).
- Sprawdź, *czy działa* - pełne IDE z wbudowanymi kompilatorami tworzą problemy, najlepiej mieć wg. mnie rozbudowany edytor (z drzewem plików, dzielonymi oknami na kod, z kolorowaniem i podpowiedzią składni, snippetami do ułatwień od pisania dokumentacji do obsługi gita) i kompilować wszystko z użyciem Makefile'a/cmake w terminalu.

- Dobrze skonfigurowane i opanowane środowisko pomaga w programowaniu! Nie warto walczyć ze środowiskiem, ale wykorzystać je jako sojusznika w walce z zadaniem.
- Programuj *przyrostowo*: napisz fragment kodu $\rightleftarrows$ od razu go przetestuj i ewentualnie poprawiaj (zbieraj tu także bazę testów poprawności działania programu) - to umożliwi ominięcie wielu problemów podczas debugowania niedziałającego kodu w całości.
- Debuguj metodą *gumowej kaczuszki* - tłumacz działanie programu komuś/sam sobie, aby łatwiej zauważać braki w logice. Przydatne są także logi programu, chociaż lepszy będzie dobry debugger.
- Staraj się rozumieć instrukcje, które piszesz - jeśli czegoś nie rozumiesz, przerwij na chwilę programowanie, doczytaj o tym i dopiero po zrozumieniu wróć do pisania programu, może to wydłużyć proces pisania, ale zaprocentuje w przyszłości.
- Dobrą praktyką jest pisanie kodu i programowanie w języku angielskim. Dokumentacja może być po polsku.

- Szczerość wobec siebie to podstawa - jeśli masz problem nie do rozwiązania samemu, szukaj pomocy bez żadnego wstydu - to ludzie mądrzy pytają o wątpliwości, głupcy milczą.
- Powiązane z powyższym - brak plagiatów itp. (inaczej `2.0`), to oszukiwanie głównie samego siebie, nie tylko mnie.
- Pytaj o pomoc kolejno: Google lub mnie (zajęcia, e-mail), potem kolegów lub na konsultacjach (nie tylko moich, ale dowolnego prowadzącego kurs, zadania są te same w innych grupach).
- Testuj funkcjonalność programu tak, jakbyś chciał go zepsuć, mimo używania zgodnie z przeznaczeniem.
- Programuj *defensywnie* - myśl o testach do spełnienia przez dany fragment kodu zawczasu, przed pisaniem, szczególnie o przypadkach granicznych i krytycznych (także sprzętowych, np. ograniczenia pamięciowe przy braku zwalniania pamięci).
- "Programuj i pisz kod taki, jaki powinien być, a nie taki, jaki umiesz (dop. obecnie) napisać.", ale jednocześnie...
- "Póki się nie nauczysz, pisz prosty kod. Potem tym bardziej."

- Zawsze czytaj z uwagą błędy, ostrzeżenia i komunikaty przy kompilacji z flagą `-Wall -pedantic`, także po `make clean` (czysta kompilacja od zera), potem analizuj możliwe ich źródła.
- Zawsze miej choćby minimalne przygotowanie przed laboratorium - wstępne programy, diagramy działania i schematy algorytmów dużo pomagają w zrozumieniu zadania i zadawaniu pytań co do niejasności. To także punkty zbierane na lepszą ocenę.
- "3 dni debugowania programu mogą zaoszczędzić godzinę planowania implementacji."
- "Program [dop. *dobrze* zaplanowany] na początku ma się kompilować, potem działać, na końcu [dop. ewentualnie] działać szybko" oraz "Premature optimization is the root of all evil" (Donald Knuth).
- `pl.wikipedia.org/wiki/KISS_(regu\OT4\la)` - *Keep It Simple, Stupid* - dąż do utrzymania eleganckiej i przejrzystej struktury programu i projektu, bez dodawania niepotrzebnych elementów.
- Kod powinien być podzielony na możliwie małe fragmenty (w ramach zdrowego rozsądku), tak, aby każda osobna funkcja miała tylko i wyłącznie jedno zadanie.

- Jeśli istnieje gotowe rozwiązanie (np. moduł sprzętowy do liczenia transformaty Fouriera względem implementacji tego samodzielnie), to warto je wykorzystać, jeśli nie jest to zabronione wprost ("Nie wynajdujmy koła na nowo").
- Wykorzystuj zdalne repozytoria z rozproszonych systemów kontroli wersji, jak `git`, do przechowywania kodu.
- Powtórz/zapisz w README podstawowe komendy w terminalu do uruchamiania programów przed pisaniem sprawozdania i PRZED oddaniem/prezentacją programu na żywo.
- Nie zaczynaj robienia zadania dzień lub dwa przed terminem jego oddania - odcinasz się od źródeł pomocy (konsultacje, koledzy) i pracujesz w stresie. Wiem, że wielu z Państwa na tym punkcie polegnie, ale jest to poświęcenie, na które jestem gotów.

- Sztywne zasady co do formy pisania programów na zajęcia uczą wymagań stawianych także w pracy (ustalona forma projektu, standardy formatowania kodu, praca w grupie, debugowanie i pair programming, code review działającego już kodu, dobre praktyki programowania, wzorce projektowe, metodyki...), to jest konieczne.
- Styl pisania kodu powinien być czytelny nie tylko dla autora w chwili jego pisania (a później niekoniecznie nawet dla niego...), musi mieć dokumentację (nazwy funkcji, argumentów + komentarze - kod powinien się także sam dokumentować dobrymi nazwami zmiennych i funkcji), projekt ma mieć jasną strukturę oraz zautomatyzowane w sobie testy (w ramach rozsądku).
- Standardy formatowania GNU kodu w C są tutaj: `www.gnu.org/prep/standards/html_node/Writing-C.html`, zalecam od siebie też tzw. camelCase. Analogiczne poradniki znajdą Państwo np. od Google i Nokii.
- Proszę zawsze przemyśleć strukturę programu, kiedy postawione będzie zadanie bez wymaganego z góry układu.

- Nawyk regularnej, samodzielnej nauki z różnych źródeł i pracy zadanej i we własnych projektach powinien być wykształcony jak najwcześniej (metoda małych kroków, *Slight Edge*).
- Inicjatywa wobec wyzwań (także ogółem, względem życia), uparte utrzymanie wybranej, przemyślanej drogi mimo nieuchronnych porażek oraz skupienie na samym procesie bardzo ułatwiają pokonywanie przeszkód, to naturalny proces nauki.
- Studia są jak szwedzki stół, korzystają z nich Państwo tyle, ile Państwo sami chcą, nie tyle, ile Państwu podadzą. Dotyczy to od opanowanej wiedzy i umiejętności, do poznanych osób, projektów, kół naukowych i studenckich. Trzeba działać i być aktywnym także bez zewnętrznych bodźców.
- Bycie naprawdę dobrym w tym, co się wybrało jako pracę (jak na przykład tutaj, umiejętność programowania jako przyszły inżynier), zwiększa samą radość życia (źródło: `www.80000hours.org`).

Źródła, hasła, słowa kluczowe...

Nie wszystko o języku C i działaniu programów w komputerze w teorii i praktyce zostanie powiedziane na zajęciach, to niemożliwe ze względu na ograniczony czas i długą historię oraz szerokie wykorzystanie tego języka. Kolejny język programowania, jaki Państwo na pewno poznają, C++ , wprowadza wiele nowych elementów w porównaniu z językiem C. Jeśli chcemy ich używać, musimy najpierw zrozumieć ich działanie.
Głównym rozwiązaniem tutaj jest samodzielna nauka.
Aby nauczyć się dobrze języka C lub C++, konieczne jest więc czytanie o nich i praca z nimi na własną rękę. Trzeba na to poświęcić trochę czasu, tym więcej, im większe mamy problemy na początku.
Nie ma innej drogi. Na szczęście istnieje wiele wspaniałych źródeł do nauki tego języka. Korzystanie z odpowiednich źródeł znacznie ułatwia nam każde zadanie. Kilka przykładów podam na kolejnych stronach.

Do C i ogólnie (1)

- wykłady dra Muszyńskiego
- literatura zebrana przez dra Muszyńskiego na jego stronie:

Literatura podstawowa:

1. Kernighan, Ritchie: **„Język ANSI C"** (w serii Klasyka Informatyki), WNT, (1987), 1994, 2007, 2010
2. Glass, Ables: **„Linux dla programistów i użytkowników"**, Helion, 2007

Standard języka C ISO/IEC 9899:1999 (C99):

- Oficjalna wersja z ANSI (płatna)
- Dostępna bezpłatnie wersja z września 2007 z uzupełnieniami TC1, TC2 i TC3 (Kopia lokalna)

Inne normy dotyczące języka C:

- Informacje o aktualnie obowiązujących normach
- Norma ISO/IEC 9899:2011 (C11):
  - Oficjalna wersja z ANSI (płatna)
  - Dostępna bezpłatnie wersja z kwietnia 2011 (Kopia lokalna)

Rysunek: Literatura podstawowa i standard C

Do C i ogólnie (2)

1. Pyszczuk: **,,Programowanie w języku C''**, CC 2010 (Kopia lokalna, Kody) *(napisana nieformalnym językiem (niestety z licznymi błędami, ale mimo to wciąż warta polecenia) książka o programowaniu w C pod linuksem, trochę podobna do pozycji ,,Język ANSI C'', z wieloma dobrze opisanymi przykładami – warto zajrzeć także na pozostałe podstrony przytoczonej powyżej strony (dostępne z umieszczonego na niej menu), traktujące między innymi o instalacji linuksa i pracy w tym systemie)*
2. Kernighan, Pike: **,,Lekcja programowania''** (w serii Inżynieria Oprogramowania), WNT 2002 *(bardziej zaawansowana pozycja z przykładami w większości w języku C – zagadnienia stylu, algorytmów i struktur danych, ich implementacji, interfejsów, wykrywania błędów, wydajności i przenośności oprogramowania)*
3. Love: **,,Linux. Programowanie systemowe''** , Helion 2008 *(systematycznie o programowaniu w C pod linuksem, o plikowych operacjach wejścia/wyjścia, o buforowanych operacjach wejścia/wyjścia, o zarządzaniu plikami, procesami, pamięcią, sygnałach i czasie)*
4. Martin: **,,Czysty kod. Podręcznik dobrego programisty''**, Helion 2010 *(obowiązkowa pozycja dla każdego początkującego programisty, który uważa, że już umie programować – jak pisać dobry kod, jak go formatować, jak implementować pełną obsługę błędów bez zaśmiecania logiki kodu, niestety z przykładami w Javie)*
5. Wróblewski: **,,Algorytmy, struktury danych i techniki programowania''**, Helion 2000, 2003, 2009 *(dla zaawansowanych z przykładami w C++ – algorytmy i struktury danych, analiza sprawności, optymalizacja, zaawansowane techniki programowania, algorytmika grafów, kodowanie i kompresja danych)*

Do C i ogólnie (3)

6. 🕮 Prata: **„Język C. Szkoła programowania"**, Helion 2006, 2016, albo King: **„Język C. Nowoczesne programowanie"**, Helion 2011 *(dla chcących wiedzieć wiele – biblie w zakresie języka C w standardzie C99, niestety drogie. Dlaczego albo? O tym tutaj)*
7. 🕮 Harel, Feldman: **„Rzecz o istocie informatyki. Algorytmika"** (w serii Klasyka Informatyki), WNT 1992, 2008 *(trochę filozoficzna, aczkolwiek zawiera elementy z zakresu algorytmiki – świetna książka dla tych, którym wydaje się, że wiedzą czym jest informatyka)*
8. 🕮 Cormen, Leiserson, Rivest: **„Wprowadzenie do algorytmów"** (w serii Klasyka Informatyki), WNT 1998, 2007 *(dla chcących wiedzieć wiele – biblia w zakresie algorytmów i struktur danych, niestety bardzo droga)*
9. 🕮 Matthew, Stones: **„LINUX: Programowanie"**, Wydawnictwo RM 1999 *(dla zaawansowanych z przykładami w C – programowanie w środowisku linux, współpraca między programami z wykorzystaniem potoków i mechanizmów komunikacji międzyprocesowej, programowanie aplikacji sieciowych za pomocą gniazd, biblioteka curses, języki powłoki i Tcl, TK)*
10. 🕮 Granneman: **„Linux. Rozmówki"**, Helion 2006 *(elementarna – praca z systemem, zarządzanie, administrowanie, wszystko przede wszystkim z poziomu konsoli tekstowej, do tego tania)*
11. 🕮 Czarny: **„Linux. Kurs"**, Helion 2007 *(elementarna w obrazkach (lamerska:) – j.w. z poziomu konsoli graficznej, przeglądanie stron www, korzystanie z poczty, nagrywanie płyt, tworzenie grafiki, redagowanie tekstów, arkuszy kalkulacyjnych)*
12. 🕮 Cameron: **„GNU Emacs"** (w serii Leksykon kieszonkowy), Helion 2002 *(ściąga dotycząca pracy z edytorem emacs)*
13. 🕮 Wirth: **„Algorytmy + struktury danych = programy"**, WNT 1980, 2004 *(ze względów historycznych – struktury danych, algorytmy ilustrowane przykładami w języku Pascal:)*

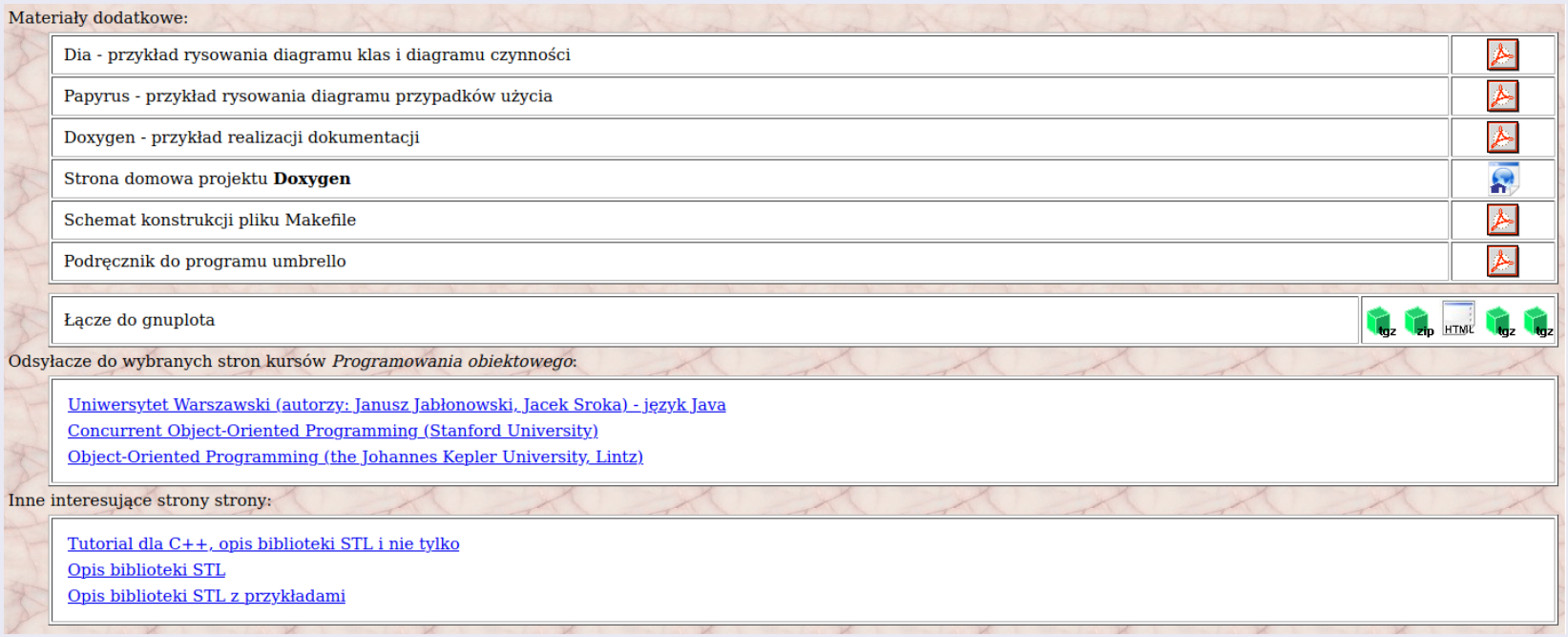
Do C i ogólnie (4):

Materiały internetowe:

- Kurs programowania w C z WikiBooks – biblioteki wolnych podręczników *(nie zawsze prawdziwe i nie zawsze ANSI, ale chyba dobre dla tych, co kumają mniej:)*
- Kurs języka C autorstwa Jacka Dondu *(systematyczny przegląd elementów składowych języka C kierowany do przyszłych programistów mikrokontrolerów)*
- Język C autorstwa Roberta Chwastka *(systematyczny przegląd elementów składowych języka C i jego funckji bibliotecznych)*
- Strona kursu Wstęp do programowania ze Studiów informatycznych *(dla tych, co wiedzieć chcą więcej)*
- Wgłąb języka C wydawnictwa Helion *(tytuł w oryginalnej pisowni Wydawnictwa!!! Trochę odmienne od prezentowanego na wykładzie spojrzenie na niektóre zagadnienia)*
- Język C – ściąga autorstwa Piotra Różnickiego *(trochę C, trochę C++, trochę ANSI, trochę Borland – świadomym nie powinno zaszkodzić)*
- Język C w pytaniach i odpowiedziach *(tytuł mówi praktycznie wszystko, może tylko nie to, że strona jest po angielsku – mnóstwo praktycznych uwag i wyjaśnień rozwiewających wątpliwości)*
- Programowanie w języku C w 5. krokach *(też po angielsku, ale za to jak elementarnie)*
- Jeszcze jedna dobra strona o programowaniu w języku C *(;-)*
- Algorytmy dla każdego autorstwa Łukasza Żarczyńskiego (w większości) *(przede wszystkim algorytmy sortowania, ale też kilka innych)*
- Środowisko programisty autorstwa Grzegorza M. Wójcika i Sławomira Kotyry (Kopia lokalna) *(jak wygodnie urządzić się w świecie GNU/Linux – zestarzał się jedynie rozdział 2 :)*
- Podstawy programowania powłoki Bash z Wiki Linux *(ciągle niekompletne, ale za to systematyczne)*
- Przewodnik po edytorze GNU Emacs – wydanie oficjalne *(warto zapoznać się, by zdawać sobie sprawę z możliwości edytora)*

Do C++ i ogólnie (1)

- dokumentacja C++
- wykłady dra Kreczmera
- materiały zebrane przez dra Kreczmera na jego stronie:



Materiały dodatkowe:

Dia - przykład rysowania diagramu klas i diagramu czynności
Papyrus - przykład rysowania diagramu przypadków użycia
Doxygen - przykład realizacji dokumentacji
Strona domowa projektu **Doxygen**
Schemat konstrukcji pliku Makefile
Podręcznik do programu umbrello
Łącze do gnuplota

Odsyłacze do wybranych stron kursów *Programowania obiektowego*:

Uniwersytet Warszawski (autorzy: Janusz Jabłonowski, Jacek Sroka) - język Java
Concurrent Object-Oriented Programming (Stanford University)
Object-Oriented Programming (the Johannes Kepler University, Lintz)

Inne interesujące strony strony:

Tutorial dla C++, opis biblioteki STL i nie tylko
Opis biblioteki STL
Opis biblioteki STL z przykładami

Rysunek: Materiały do C++

Do C++ i ogólnie (2)

- książki: *Symfonia C++*, *Pasja C++*, ***Opus Magnum C++*** (najnowsza), Grębosz J. [łatwy, obszerny podręcznik w 3 tomach]; *Język C++ - kompedium wiedzy*, Stroustrup B.; *Skuteczny i nowoczesny C++*, *Effective C++ 3rd Edition*, Meyers S.; *C++ Primer*; Leppman S.B.
- *C++ Notes for Professionals* (Stackoverflow) [zaawansowane!]
- *Google C++ Style Guide* (Google)
- *Praktyczny wstęp do wytwarzania programowania* (Nokia) [zaawansowane!]
- `https://www.freecodecamp.org/news/search?query=c++`

Do C/C++ i inne dodatkowe wg. mnie

- *Google to mój przyjaciel*, używaj mądrze `stackoverflow.com`
- `devdocs.io/c/` i en.cppreference.com/w/- dobre interaktywne dokumentacje do C/C++
- płytka STM32F4 od ST i własna elektronika, dokumentacja mikrokontrolera (dalej skrót `uC`), `Forbot.pl` i poniższe - systemy wbudowane to interesujący element studiów AiR, który pojawia się na nich zdecydowanie za późno. (*Uwaga - programowanie uC w C jest czymś zgoła innym niż programowanie na Linuxie w C! Trzeba być tego świadomym, to inne zastosowanie C.*)
- Arduino - proste, ciekawe, bardzo tanie jako sprzęt do wejścia w elektronikę, dużo przykładów w Internecie na szybki start
- Atmega32 od AVR - starszy, ale nawet fajny uC, pojawi się na Państwa studiach, książki/strony o nim:
  - książki autorstwa Mirosława Kardasia (podstawy podstaw dla uC),
  - `mikrokontrolery.blogspot.com` (jw.)

Inne materiały, dla samodzielnej nauki dodatkowych, ciekawych dla inżyniera rzeczy, kolejność bliższa losowej

- `spoj.com` - zbiór typowych zadań dla programistów, od bardzo podstawowego poziomu trudności, podstrona `FRAKTAL` zawiera informacje o ciekawym konkursie dla programistów z algorytmów
- `projecteuler.net` - zbiór zadań trudniejszych, bliższych matematyce komputacyjnej, pozwalącymi na zderzenie z rzeczywistymi ograniczeniami sprzętowymi (pamięć, długość obliczeń, złożoność i trudność algorytmu)
- Codility - platforma używana do testowania umiejętności programowania kandydata w rozmowach kwalifikacyjnych, dużo zadań
- GeeksForGeeks - przydatna strona z dużą ilością zadań z rozwiązaniami i artykułami o tematyce powiązanej z programowaniem
- `en.wikibooks.org/wiki/C_Programming/Common_practices` - zbiór dobrych praktyk programowania w C i ogółem
- `pl.wikibooks.org/wiki/Programowanie_w_systemie_UNIX/Valgrind` - ułatwi m.in. badanie wycieków pamięci w C i C++

Inne materiały, dla samodzielnej nauki dodatkowych, ciekawych dla inżyniera rzeczy, kolejność bliższa losowej

- Pasja informatyki (strona, YT) - bardzo wartościowa witryna i kanał autorstwa Mirosława Zelenta i Damiana Stelmacha o programowaniu (tutoriale C++ [dop. mocno zdezaktualizowane, niestety], PHP, kurs Linuxa itd.) i życiu (ww. Slight Edge)
- Computerphile (YT) - brat Numberphile, wiele interesujących problemów CS, jak analiza danych, algorytmika, ataki hakerskie, działanie różnych technologii
- CodeBullet (YT) - programowanie sztucznej inteligencji, która umie grać w różne proste gry oraz inne ciekawe projekty programistyczne
- CodeParade (YT) - uczenie maszynowe, gry, algorytmy, fraktale, "inteligentne"generatory danych
- sentdex (YT) - tutoriale Pythona od podstaw do wprowadzenia PyTorch i Django, w tym projekty z robotyki i wiele innych
- The Coding Train (YT) - programowanie na żywo kodu do różnych wyzwań i zadań, jak zbiór Mandelbrota, wizualizacje sortowań itd.

Inne materiały, dla samodzielnej nauki dodatkowych, ciekawych dla inżyniera rzeczy, kolejność bliższa losowej

- *live coding* (Twitch i inne) - ciekawe źródła wiedzy od doświadczonych programistów na żywo na streamach
- `handmadehero.org` - gra pisana na streamie ZUPEŁNIE od podstaw w C (nawet sterowniki grafiki, dźwięku, wielowątkowości itd.)
- elektroda.pl (strona, YT) - dobre źródło wiedzy o wielu projektach z elektroniki i uC, trzeba umieć używać wyszukiwarki na forum(!)
- ElektroPrzewodnik (YT) - kanał dla wszystkich początkujących elektroników, opis teorii, używanie modułów sprzętowych, projekty
- Learn Engineering (YT) - opis wielu zagadnień interesujących dla inżyniera elektronika
- MIT OpenCourseWare (YT) - wykłady i materiały z kursów MIT, które pozwalają na samodzielną naukę nawet złożonych przedmiotów z i spoza zakresu studiów na PWr na AiR

Inne materiały, dla samodzielnej nauki dodatkowych, ciekawych dla inżyniera rzeczy, kolejność bliższa losowej

- ROBOTIS OpenSourceTeam (YT) - interesujący i wartościowy kanał dotyczący projektów robotycznych (modelowanie, programowanie, mechanika)
- RobotStudio (YT) - kanał dotyczący oprogramowania RobotStudio od ABB, praktycznie co roku organizują konkurs na najlepszy studencki projekt w nim
- iAutomatyka (strona, YT) - portal z branży automatyki przemysłowej, z kursami, które rozwiną Państwa suchą wiedzę z wykładów
- RS Elektronika (YT) - kanał dla osób zainteresowanych szeroko pojętą elektroniką
- KoNaR (koło naukowe i ich strona) - znane w całej Polsce i na świecie Koło Naukowe Robotyków, z masą świetnych materiałów na swojej stronie do startu przygody z budowaniem robotów

Inne materiały, dla samodzielnej nauki dodatkowych, ciekawych dla inżyniera rzeczy, kolejność bliższa losowej

- 3Blue1Brown (YT) - wizualizacje wielu zagadnień z matematyki (przydatnej w robotyce) i inżynierii w przystępny sposób
- Primer (YT) - ciekawe symulacje złożonych systemów (teoria gier, rywalizacja, agresja, ewolucja)
- MajorPrep (YT) - interesujące filmiki z pogranicza inżynierii
- Datacamp - strona z kursami do programowania i Data Science, niestety płatna (ale prowadzący odpowiednie kursy może zapewnić dostęp do 6 miesięcy)
- Geogebra - przydatne narzędzie do tworzenia rysunków geometrycznych, w tym interaktywnych z użytkownikiem
- Adams, AutodeskInventor, IronCAD - do projektowania robotów (trzeba nauczyć się tego samodzielnie...)
- CircuitMaker, Altium, Eagle, KiCAD, Multisim - do projektowania/ symulacji elektroniki nie w programie z lat 90. na Windowsa XP

Inne materiały, dla samodzielnej nauki dodatkowych, ciekawych dla inżyniera rzeczy, kolejność bliższa losowej

- `gist.github.com/aras-p/6224951` i `github.com/Droogans/unmaintainable-code` - (mniej poważne) jak nie pisać kodu
- `learngitbranching.js.org` - interaktywny poradnik do nauki gita
- `linuxjourney.com` - interaktywny poradnik do terminala Linuxa
- Atnel - mirekk36 (YT) - kanał Mirosława Kardasia, z dużą ilością wideoporadników do AVR
- Brian Douglas (YT) - wykłady dotyczące teorii sterowania i kontroli systemów (czyli materiał z MUD-ów dla Państwa za jakiś czas)
- Bartek Kurosz (YT) - kilka filmów, które ułatwią Państwu zdać SAiC
- `biblioteka.pwr.edu.pl/e-zasoby/zdalny-dostep-proxy` - PWr ma dostęp do wielu zasobów elektronicznych (książki Springera, artykuły itd.), można mieć je przez proxy w domu, do tego też Safari Books

Inne materiały, dla samodzielnej nauki dodatkowych, ciekawych dla inżyniera rzeczy, kolejność bliższa losowej

- `cut-the-knot.org/` - interaktywne puzzle matematyczne
- `deltami.edu.pl` - strona czasopisma Delta, z ciekawymi artykułami z matematyki, fizyki, informatyki i astronomii
- `khanacademy.org` - duża ilość kursów z przekroju całych studiów
- `regexone.com` - wyrażenia regularne (*lepsza* notacja MBNF)
- `pythontutor.com/c.html` - wizualizacja działania kodu w C
- `cp-algorithms.com/` - opis wielu algorytmów i struktur danych
- `algonotes.com/pl/` - zapiski Tomasza Idziaszka dot. algorytmów
- książki: *Algorytmy, struktury danych i techniki programowania*, *Czysty kod* i podobne pozycje, *Wzorce projektowe* w uznanej książce, seria *Perełki programowania*, seria *Head First...*, *Język C. Szkoła programowania* Stephena Praty, ogółem wszystkie książki autorstwa Donalda Knutha

Inne materiały, dla samodzielnej nauki dodatkowych, ciekawych dla inżyniera rzeczy, kolejność bliższa losowej

- `https://matthieu-moy.fr/c/c_collection/` - zbiór programów, pokazujący szczególne przypadki zachowania programów w C i C++

Zbiór źródeł do nauki elektroniki i robotyki we własnym zakresie, od podstaw do poziomu uniwersyteckiego

- `forbot.pl/blog/jak-zaczac-budowac-roboty-sprawdzony-poradnik`
- `forbot.pl/blog/kurs-budowy-robotow-arduino-wstep-spis-tresci-id18935`
- `forbot.pl/blog/kurs-elektroniki-dla-poczatkujacych-id5151`
- Polecam inwestycję w gotowe zestawy do nauki elektroniki, robotyki i Arduino/OpenCV/STM32 z Forbota do znalezienia na Botlandzie - ilość włożonych PLN w swój rozwój do wpływu na Państwa przyszłość jest potencjalnie ogromna. Polecam także Koło Naukowe Robotyków na naszym wydziale - to świetne miejsce do startu nauki i poznania starszych kolegów (czego mocno brakowało przez nauczanie zdalne). Za realizację własnego projektu robotycznego można otrzymać wyższą ocenę.
  `https://konar.pwr.edu.pl/index.php/projekty/`

Inne materiały, dla samodzielnej nauki dodatkowych, ciekawych dla inżyniera rzeczy, kolejność bliższa losowej

Podstawowe sklepy z elektroniką do robotyki:

- `electropark.pl`
- `botland.com.pl`

Bardziej zaawansowane materiały do nauki robotyki samodzielnie:

- `https://github.com/kiloreux/awesome-robotics`
- `https://laconicml.com/computer-science-curriculum-youtube-videos/`
- `https://www.reddit.com/r/robotics/comments/kpkne5/how_to_get_started_in_robotics_education_career/`
- `https://www.toptal.com/developers/blog`

Co może się przydać później na Państwa studiach, w skrócie:

- cierpliwość i wytrwałość w pracy i w nauce
- C/C++
- Python
- Matlab + Simulink
- LaTeX
- git
  - GitHub
  - Gitlab
  - BitBucket
- ROS 1&2 - przydatny dla robotyków

I na koniec

Powodzenia z programowaniem!