

KARTA KURSU

Nazwa	Programowanie niskopoziomowe
Nazwa w j. ang.	Low-level programming

Koordinator	mgr inż. Mateusz Rus	Zespół dydaktyczny
		mgr inż. Mateusz Rus
Punktacja ECTS*	3	

Opis kursu (cele kształcenia)

Celem kursu jest zapoznanie studentów z podstawami programowania niskopoziomowego, ze szczególnym uwzględnieniem dobrych praktyk związanych z bezpieczeństwem i podatnością w kodzie.

Studenci poznają narzędzia i środowiska pracy, techniki optymalizacji kompilatora pod kątem kodu niskopoziomowego, mechanizmy niskopoziomowe w kodzie wysokopoziomowym, podejścia do kontroli nad sprzętem i zasobami komputera poprzez umożliwienie tworzenia kodu na bardziej szczegółowym poziomie. Kurs realizowany jest w języku polskim.

Warunki wstępne

Wiedza	Warunkiem przystąpienia do zajęć jest znajomość podstawowych zagadnień z programowania, języków i narzędzi programowania obiektowego.
Umiejętności	Potrafi implementować podstawowe algorytmy i struktury danych w języku programowania obiektowego w oparciu o zasady bezpieczeństwa.
Kursy	Programowanie. Języki i narzędzia programowania obiektowego.

Efekty uczenia się

	Efekt uczenia się dla kursu	Odniesienie do efektów kierunkowych
Wiedza	Po zakończeniu kursu student:	
	W01: ma wiedzę o cechach programowania niskopoziomowego ze szczególnym uwzględnieniem dobrych praktyk związanych z bezpieczeństwem i podatnością w kodzie:	K_W03
	W02: definiuje narzędzia i środowiska pracy, mianowicie omawia kompilatory i narzędzia do generowania kodu asemblera, debuggery i narzędzia do śledzenia wykonania kodu, deasembler i analizatory binarne. podstawowe pojęcia związane z organizacją i architekturą komputera; opisuje cechy charakterystyczne i parametry urządzeń techniki komputerowej.	K_W06
	W03: opisuje mechanizmy niskopoziomowe w kodzie wysokopoziomowym.	K_W03, K_W06
	W04: rozumie bezpieczeństwo i podatność w kodzie, wyjaśnia bezpieczne praktyki programistyczne w kontekście niskopoziomowym.	K_W07

Umiejętności	Efekt uczenia się dla kursu	Odniesienie do efektów kierunkowych
	Po zakończeniu kursu student:	
	U01: potrafi implementować podstawowe algorytmy w języku programowania niskiego poziomu z uwzględnieniem zasad bezpieczeństwa.	K_U04
	U02: umie przeprowadzić analizę kodu wysokopoziomowego oraz zoptymalizować kompilator w kontekście niskopoziomowym niskopoziomowego.	K_U04, K_U06
	U03: umie przeprowadzić optymalizację krytycznych fragmentów kodu i analizę bezpieczeństwa oraz podatności w kodzie.	K_U04, K_U09, K_U11
	U04: potrafi wykonać projekt praktyczny do dedykowanych zagadnień programowania niskopoziomowego.	K_U05, K_U13

Kompetencje społeczne	Efekt uczenia się dla kursu	Odniesienie do efektów kierunkowych
	Po zakończeniu kursu student:	
	K01: rozumie potrzebę i zna możliwości ciągłego dokształcania się – podnoszenia kompetencji zawodowych.	K_K01, K_K02
	K02: potrafi pracować zespołowo przyjmując w zespole różne role.	K_K01, K_K03, K_K04

Studia stacjonarne

Organizacja							
Forma zajęć	Wykład (W)	Ćwiczenia w grupach					
		A	K	L	S	P	Z
Liczba godzin	15			15			

Studia niestacjonarne

Organizacja							
Forma zajęć	Wykład (W)	Ćwiczenia w grupach					
		A	K	L	S	P	Z
Liczba godzin	10			10			

Opis metod prowadzenia zajęć

Kurs składa się z ćwiczeń prowadzonych w formie ćwiczeń laboratoryjnych. W ramach zajęć studenci projektują i tworzą zadane programy w języku C++, które następnie są omawiane.
Metody oceny: Kolokwia i testy teoretyczne, Zadania programistyczne na zajęciach.
Ćwiczenia laboratoryjne – wspólne lub samodzielne (przygotowanie do laboratorium lub praca w grupach laboratoryjnych) rozwiązywanie zadań zadawanych przez prowadzącego, zakończone praktyczną weryfikacją wyników, symulacja laboratoryjna, odpowiednią analizą bezpieczeństwa i podatności w kodzie.

Formy sprawdzania efektów uczenia się

	E – learning	Gry dydaktyczne	Ćwiczenia w szkole	Zajęcia terenowe	Praca laboratoryjna	Projekt indywidualny	Projekt grupowy	Udział w dyskusji	Referat	Praca pisemna (esej)	Egzamin ustny	Egzamin pisemny	Zadania problemowe
W01					X		X	X					
W02					X		X	X					
W03					X		X	X					
W04					X		X	X					
U01					X		X	X					
U02					X		X	X					
U03					X		X	X					
U04					X		X	X					
K01					X			X					
K02					X		X	X					

Kryteria oceny	Dopuszcza się przeprowadzenie zaliczenia z zastosowaniem metod i technik kształcenia na odległość. Obecność na zajęciach, aktywność (zadawanie pytań/udzielanie odpowiedzi ustnej w trakcie ćwiczeń laboratoryjnych i podczas dyskusji dydaktycznej kierowanej na omawiany temat, systematyczność w wykonaniu ćwiczeń laboratoryjnych udokumentowana sprawozdaniem oraz uzyskanie pozytywnej oceny końcowej – średniej ocen formujących zaliczenia z ceną.
----------------	--

Uwagi	
-------	--

Treści merytoryczne (wykaz tematów)

- Wstęp do programowania niskopoziomowego:
 - podstawowe różnice między programowaniem wysokopoziomowym a niskopoziomowym;
 - wprowadzenie do architektury procesora (rejstry, pamięć, instrukcje, zegar procesora);
 - rola języków niskopoziomowych w analizie kodu wysokopoziomowego.
- Narzędzia i środowiska pracy:
 - omówienie kompilatorów (GCC, Clang) i narzędzi do generowania kodu asemblera (opcje kompilatora: np. `-S` (w GCC) do generowania kodu asemblera);
 - debuggery i narzędzia do śledzenia wykonania kodu (GDB (GNU Debugger), LLDB, objdump, radare2);
 - deasemblerzy i analizatory binarne (IDA Free/Pro, Ghidra, obserwowanie zmian w kodzie maszynowym po optymalizacjach).
- Analiza kodu wysokopoziomowego w kontekście niskopoziomowym:
 - translacja kodu w języku C++ do kodu asemblera (przykłady prostych programów (pętle, instrukcje warunkowe, wskaźniki), zmiany w kodzie źródłowym a wygenerowany kod asemblera);
 - ABI (Application Binary Interface) (przekazywanie argumentów do funkcji (np. przez stos/rejstry), zasady zwracania wartości.);
 - mechanizmy zarządzania pamięcią (analiza dynamicznej alokacji (malloc, new) i dealokacji; konwersja wskaźników wysokopoziomowych na adresy fizyczne w pamięci).

4. Optymalizacje kompilatora a kod niskopoziomowy:
 - wpływ flag optymalizacyjnych kompilatora na wygenerowany kod assemblera (analiza różnych poziomów optymalizacji, usuwanie martwego kodu, inlining, unrolling pętli);
 - ręczne zmiany w kodzie C++ dla lepszej wydajności.
5. Przerwania i obsługa sprzętu:
 - podstawy pracy z przerwaniami (np. IRQ) w kontekście języka C i assemblera;
 - analiza kodu obsługującego niskopoziomowe operacje sprzętowe (obsługa portów I/O, współpraca z pamięcią mapowaną do urządzeń).
6. Mechanizmy niskopoziomowe w kodzie wysokopoziomowym:
 - implementacja stosu i serty w kodzie C++ a ich odwzorowanie w assemblerze;
 - analiza kodu generowanego dla mechanizmów C++:
7. Optymalizacja krytycznych fragmentów kodu:
 - detekcja wąskich gardeł w kodzie przy użyciu profilera (perf, Valgrind);
 - ręczna optymalizacja wybranych fragmentów kodu przy użyciu assemblera;
 - tworzenie "inline assembly" w kodzie C++.
8. Analiza bezpieczeństwa i podatności w kodzie:
 - wyszukiwanie podatności na przepełnienia bufora;
 - analiza exploitów na poziomie assemblera;
 - bezpieczne praktyki programistyczne w kontekście niskopoziomowym.
9. Propozycja projektów praktycznych:
 - projekt 1: analiza kodu wysokopoziomowego i jego optymalizacji pod kątem wydajności;
 - projekt 2: śledzenie błędów w kodzie (np. wycieków pamięci, niepoprawnego dostępu do pamięci) z wykorzystaniem narzędzi takich jak Valgrind, GDB;
 - projekt 3: przeprowadzenie analizy kodu generowanego dla konkretnego urządzenia lub systemu operacyjnego (np. sterowniki wbudowane).

Wykaz literatury podstawowej

1. Assembler: sztuka programowania : HLA 4/Randall Hyde ; [tł. Przemysław Szeremiota]. Wyd. 2. Gliwice: Helion, 2011.
2. Mikroprocesory jednoukładowe PIC / Stanisław Pietraszek. Gliwice: Wydaw. Helion, 2002.
3. C++ : zadania z programowania z przykładowymi rozwiązaniami / Mirosław J. Kubiak. Gliwice : Helion, 2011.
4. Programowanie niskopoziomowe:
https://wazniak.mimuw.edu.pl/index.php?title=Programowanie_niskopoziomowe (Dostęp 07.03.2025).

Wykaz literatury uzupełniającej

1. Od C do ASSEMBLERA czyli Jak skutecznie programować interface użytkownika / Piotr Wróblewski. Gliwice : Helion, 1992.
2. Programowanie w języku Assembler / Stanisław Kruk. Warszawa: Wydawnictwo PLJ, 1992.
3. Asemblery i programy ładujące / D. W. Barron ; [z jęz. ang. tł. Henryk Stelmasik, Włodzimierz Zuberek]. Warszawa: PWN, 1982.
4. Programowanie niskopoziomowe: co to jest i jak się zabrać.
<https://www.interaktywna.pl/spoleczenstwo/kariera/programowanie-niskopoziomowe-co-to-jest-i-jak-sie-zabrac/> (dostęp 07.03.2025).
5. An introduction to low-level programming. <https://codilime.com/blog/introduction-to-low-level-programming/> (dostęp 07.03.2025)

Bilans godzinowy zgodny z CNPS (Całkowity Nakład Pracy Studenta) – **studia stacjonarne**

Liczba godzin w kontakcie z prowadzącymi	Wykład	15
	Konwersatorium (ćwiczenia, laboratorium itd.)	15
	Pozostałe godziny kontaktu studenta z prowadzącym	10
Liczba godzin pracy studenta bez kontaktu z prowadzącymi	Lektura w ramach przygotowania do zajęć	10
	Realizacja zadań domowych (problemowych) po zapoznaniu się z niezbędną literaturą przedmiotu	
	Przygotowanie projektu lub prezentacji na podany temat (praca indywidualna lub w grupie)	15
	Przygotowanie do egzaminu/zaliczenia	10
Ogółem bilans czasu pracy		75
Liczba punktów ECTS w zależności od przyjętego przelicznika		3

Bilans godzinowy zgodny z CNPS (Całkowity Nakład Pracy Studenta) – **studia niestacjonarne**

Liczba godzin w kontakcie z prowadzącymi	Wykład	10
	Konwersatorium (ćwiczenia, laboratorium itd.)	10
	Pozostałe godziny kontaktu studenta z prowadzącym	10
Liczba godzin pracy studenta bez kontaktu z prowadzącymi	Lektura w ramach przygotowania do zajęć	15
	Realizacja zadań domowych (problemowych) po zapoznaniu się z niezbędną literaturą przedmiotu	
	Przygotowanie projektu lub prezentacji na podany temat (praca indywidualna lub w grupie)	20
	Przygotowanie do egzaminu/zaliczenia	10
Ogółem bilans czasu pracy		75
Liczba punktów ECTS w zależności od przyjętego przelicznika		3