
POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

# Programowanie w językach C i C++ (PJC) – sylabus

Marzec 2021

## 1. Wprowadzenie

Przedmiot poświęcony jest powtórzeniu ogólnych podstaw programowania przy pomocy języka C++. Studenci poznają odmienny sposób manipulacji tworzonymi przez siebie abstrakcjami i ich implementacjami w porównaniu do tego, jak programowali dotychczas. Ich uwaga zostanie zwrócona ku ważnym korelacjom pomiędzy kosztami korzystania z danych abstrakcji a korzyściami, które za sobą niosą. Poznają też odmienne podejścia do rozwiązywania tych samych problemów w kontekście innego języka programowania. Zwrócona zostanie ich uwaga nie tylko na funkcjonalność, ale również i na jakość pisanego kodu – w sposób możliwie uniwersalny w kontekście języków programowania. Przedmiot opiera się na systematycznej pracy przez cały semestr, podczas której student zdobywa nową wiedzę oraz utrwala już poznane elementy.

Przedmiot zakłada wykorzystanie standardu C++17, ponieważ standard C++20 nie jest w wystarczającym stopniu wspierany przez jakikolwiek, dostępny publicznie kompilator.

## 2. Plan zajęć

| Nr ćwiczeń | Omawiany materiał |
|---|---|
| **1** | Krótkie omówienie filozofii i założeń przedmiotu, przedstawienie i omówienie zasad zaliczenia oraz projektu.<br>Powtórzenie podstawowych operacji takich jak:<br>• Wyświetlanie danych |

| | |
|---|---|
| | • Sczytywanie / pobieranie danych<br>• Instrukcje warunkowe<br>• Pętle<br>• Podstawowe typy danych, takie jak:<br>  ○ Typy wbudowane (`char`, `int`, `double`, itp.)<br>  ○ Typ tekstowy (`std::string`)<br>  ○ Kontener (`std::vector`)<br>• Wskazanie znaczących różnic pomiędzy Javą a C++ |
| **2** | Dokładniejsze omówienie operacji na `std::vector`.<br>Wprowadzenie do funkcji. Argumenty, argumenty domyślne, typy zwracane. Przeciążanie funkcji.<br>Zwrócenie uwagi na semantykę wartości i na intuicyjność operatorów, których funkcjonalność w języku C++ jest rozszerzona również dla typów zdefiniowanych przez użytkownika.<br>Wprowadzenie do słowa kluczowego `auto`. |
| **3** | "*Almost always auto*".<br>Standardowe algorytmy:<br>• <algorithm><br>• <numeric><br>Wprowadzenie do koncepcji i abstrakcji iteratorów.<br>Wykorzystanie iteratorów w standardowych algorytmach.<br>Przykłady, gdzie użycie pętli bezpośrednio zwiększa trudność implementacji, więc lepiej użyć jest standardowych algorytmów. |
| **4** | Wprowadzenie do lambd.<br>Zobrazowanie działania algorytmów przyjmujących predykat, funkcję oraz consumera.<br>Powtórzenie działania rekurencji z przykładami rekurencyjnych algorytmów w bibliotece standardowej. |
| **5** | Wprowadzenie do typów zdefiniowanych przez użytkownika w C++. Zastosowanie ich przy `std::vector`, przy standardowych algorytmach oraz przy funkcjach.<br>`enum class` i jego wykorzystanie w praktyce. |
| **6** | Wprowadzenie do obiektowości w C++.<br>*Special member functions*:<br>• Konstruktor domyślny<br>• Konstruktor<br>• Konstruktor kopiujący<br>• Operator nadpisania<br>• Destruktor |

| | |
|---|---|
| | Metoda a funkcja. Metody statyczne. |
| **7** | Omówienie kontenerów sekwencyjnych i ich celów:<br>• `std::vector`<br>• `std::deque`<br>• `std::array`<br>• `std::list`<br>Omówienie kontenerów asocjacyjnych i ich celów:<br>• `std::set`<br>• `std::map`<br>Omówienie kontenerów adaptacyjnych i ich celów:<br>• `std::stack`<br>• `std::queue`<br>• `std::priority_queue`<br>• `std::deque` |
| **8** | Dziedziczenie oraz enkapsulacja w C++.<br>Wprowadzenie referencji.<br>Dynamiczny polimorfizm w C++. Znaczenie `const`. |
| **9** | Powrót do standardowych algorytmów i ich wykorzystanie z własnymi klasami oraz kontenerami.<br>Powrót do enkapsulacji i wprowadzenie `friend`.<br>Przestrzenie nazw. Wersjonowanie implementacji i grupowanie podobnych funkcjonalności.<br>**Udostępnienie i omówienie projektu semestralnego.** |
| **10** | Powtórzenie destruktorów na przykładach RAII (ang. *Resource Acquisition Is Initialisation*).<br>Operacje na plikach z wykorzystaniem:<br>• `std::fstream`<br>• `std::stringstream`<br>Zwrócenie uwagi na brak konieczności jawnego stosowania instrukcji strażniczych (takich jak *try-with-resources* (Java), *with* (Python) czy *using* (C#)) dzięki destruktorom. |
| **11** | Wprowadzenie do wyjątków oraz do *try-catch*.<br>Wprowadzenie do szablonów i ich dokładne omówienie:<br>• Rozwiązanie problemu zbyt dużej liczby przeciążeń za pomocą szablonu funkcji.<br>• Rozwiązanie problemu obsługi generycznych typów danych przechowujących inny typ danych.<br>• Współpraca z unikalnymi typami (lambdy) i przyjmowaniu ich jako argumenty. |

| | |
|---|---|
| **12** | Przeciążanie operatorów:<br>• Przypomnienie o operatorze nadpisania.<br>• Przypomnienie o intuicyjności operatorów nawet z typami zdefiniowanymi przez użytkownika (`==` czy `<` dla `std::vector`).<br>• Przypomnienie o `friend` w kontekście przeciążania operatora `<<`. |
| **13** | Wprowadzenie do sterty i stosu.<br>Wprowadzenie do wskaźników. Użycie wskaźników jako widoku na obiekty. Porównanie zachowania wskaźników z zachowaniem "obiektów" w Javie.<br>Implementacja ekstensji klasy w języku z semantyką wartości – wskazanie koniecznego użycia (surowych) wskaźników.<br>Omówienie dynamicznej alokacji pamięci.<br>Wykorzystanie technik RAII do automatyzacji (i zwiększenia bezpieczeństwa) zwalniania zasobów – w tym pamięci, czyli inteligentne wskaźniki. |
| **14** | Omówienie wielowątkowości tylko na przykładach praktyki:<br>• `std::async`<br>• `std::future` (oraz `std::promise`)<br>• `std::thread`<br>• `std::mutex`<br>• `std::shared_ptr` |
| **15** | **Obrona projektu semestralnego.** |

## 3. Prace domowe

Po wybranych ćwiczeniach udostępniane będą krótkie zadania do samodzielnego wykonania i przesłania na platformę Microsoft Teams na zasadach modułu Assignments / Zadania. Zadania te będą oceniane przez prowadzącego na zasadzie małych punktów – wynik za każde zadanie będzie zawierał się w przedziale ⟨0,1⟩ i może być ułamkiem. Przewiduje się, że liczba zadań będzie wynosiła 13.

Terminem oddania zadań będą najczęściej najbliższe zajęcia po ich udostępnieniu. W celu weryfikacji i upewnienia się, należy zawsze wpierw sprawdzać datę dostarczenia podaną w module Assignments / Zadania na platformie Microsoft Teams przy konkretnym zadaniu.

## 4. Projekt

Podczas ćwiczeń nr **9** zostanie udostępniony **semestralny projekt programistyczny**, którego szczegóły zostaną opisane w osobnym, dedykowanym dokumencie.

## 5. Zaliczenie ćwiczeń

Punkty wymagane do uzyskania odpowiedniej oceny są przedstawione w tabeli poniżej (Tabela 1):

| Uzyskana liczba punktów z ćwiczeń: | Ocena z ćwiczeń: |
|---|---|
| 0-50 | 2 |
| 51-65 | 3 |
| 66-75 | 3.5 |
| 76-85 | 4 |
| 86-91 | 4.5 |
| 92+ | 5 |

*Tabela 1: Mapowane przedziałów punktowych na odpowiadające im oceny*

Zaliczenie ćwiczeń wymagania uzyskania co najmniej oceny **3**. Uzyskana liczba punktów jest wyliczana ze wzoru podanego poniżej (Równanie 1):

$$Suma\ punktów = \frac{x}{10} * y$$

*Równanie 1: Wzór na obliczenie wynikowej liczby punktów z przedmiotu*

Gdzie $x$ to **suma uzyskanych punktów za wszystkie prace domowe**, a $y$ to **uzyskana liczba punktów za projekt semestralny**.

Można to rozumieć na dwa sposoby: Oceną z przedmiotu jest ocena z projektu, która jest modyfikowana przez wyznacznik systematycznej pracy przez cały semestr; Na ocenę końcową składają się dwa czynniki – ocena z projektu oraz ocena z prac domowych, gdzie należy zwrócić uwagę, że perfekcyjny wynik z jednego z tych składowych nie jest wystarczający do zaliczenia całego przedmiotu.

Liczba możliwych punktów do zdobycia za prace domowe to **13**, a liczba możliwych punktów do uzyskania za projekt to **100**. Efektywnie, rozpatrując przedziały liczb rzeczywistych:

$$x \in \langle 0,13 \rangle$$

*Równanie 2: Przedział punktów możliwych do zdobycia z **zadań domowych***

$$y \in \langle 0,100 \rangle$$

*Równanie 3: Przedział punktów możliwych do zdobycia z **projektu semestralnego***

Warto zwrócić uwagę, że możliwa do zdobycia liczba punktów z zadań domowych jest większa niż dzielnik we wzorze na sumę punktów (Równanie 1). Jest to intencjonalne i ma na celu osiągnięcie następujących efektów:

1. Zakłada się, że błędy lub sytuacje wyjątkowe się zdarzają i nieoddanie jednego czy dwóch zadań nie skreśla szansy na otrzymanie najwyższej oceny z przedmiotu.
2. Dzięki możliwości uzyskania wynikowego mnożnika większego niż 1 (z prac domowych), poprzedni punkt również tyczy się potknięć w przypadku implementacji projektu semestralnego. Systematyczna praca przez cały semestr może efektywnie nadrobić stracone punkty z projektu.
3. Studenci, którzy uzyskają już punkty za zdecydowaną większość zadań, dalej mogą czerpać motywację do ukończenia ostatnich zadań, ponieważ ich poprawne wykonanie tworzy większą "poduszkę punktową", dzięki czemu mogą uniknąć zmniejszenia oceny wynikającego z mniejszych przeoczeń w projekcie.