Wymagania edukacyjne niezbędne do otrzymania przez ucznia śródrocznej i rocznej oceny klasyfikacyjnej z Programowania strukturalnego i obiektowego

Klasa 3 technikum

Wymagania edukacyjne niezbędne do otrzymania przez ucznia śródrocznej oceny klasyfikacyjnej

temat lekcji	zagadnienia	Wymagania edukacyjne na poszczególne oceny				
		ocena dopuszczająca uczeń	ocena dostateczna uczeń	ocena dobra uczeń	ocena bardzo dobra uczeń	ocena celująca uczeń
Rozdział 1. Wprowadzenie do programowania						
Podstawowe pojęcia	Algorytm, kodowanie, program.	Rozpoznawanie podstawowych pojęć, ale brak umiejętności ich opisania.	Zrozumienie i umiejętność opisania podstawowych pojęć.	Zrozumienie, umiejętność opisania oraz zastosowania podstawowych pojęć w prostych zadaniach.	Analiza i interpretacja podstawowych pojęć w kontekście różnych problemów.	Samodzielne poszerzanie wiedzy o podstawowe pojęcia i ich zastosowanie w zaawansowanych problemach.
Podstawy algorytmiki	Blok algorytmiczny, sekwencja, decyzja.	Rozpoznawanie podstawowych algorytmów bez zdolności ich implementacji.	Implementacja prostych algorytmów oraz zrozumienie ich działania.	Analiza, modyfikacja i implementacja różnych algorytmów.	Optymalizacja algorytmów oraz analiza ich złożoności.	Samodzielne tworzenie skomplikowanych algorytmów oraz ich analiza.
Rozdział 2. Podstawy programowania						
Podstawowe elementy języka C++	Zmienne, stałe, operatory.	Zastosowanie podstawowej składni w gotowych przykładach.	Samodzielne tworzenie prostych programów w C++.	Zastosowanie bardziej zaawansowanej składni, takiej jak wskaźniki czy referencje.	Zastosowanie zaawansowanych technik programowania w C++.	Samodzielne tworzenie i optymalizacja skomplikowanych programów.
Podejmowanie decyzji	If, else, switch.	Zastosowanie	Korzystanie z	Optymalizacja i	Analiza i	Samodzielne

w programie		prostych instrukcji warunkowych (if, else).	bardziej skomplikowanych struktur warunkowych (switch, zagnieżdżone ify).	refaktoryzacja kodu z instrukcjami warunkowymi.	interpretacja bardziej skomplikowanych struktur warunkowych.	projektowanie skomplikowanych struktur decyzyjnych.
Pętle programowe	For, while, do-while.	Zastosowanie prostych pętli (for, while).	Korzystanie z bardziej skomplikowanych pętli i zagnieżdżonych pętli.	Optymalizacja kodu z wykorzystaniem pętli.	Zastosowanie pętli w skomplikowanych zadaniach programistycznych.	Analiza, refaktoryzacja i optymalizacja kodu zawierającego złożone struktury pętlowe.
Typ wyliczeniowy	Enum.	Uczeń zna podstawową definicję typu wyliczeniowego. Potrafi stworzyć podstawowe wyliczenie, ale ma trudności z jego zastosowaniem w praktyce.	Uczeń zna i rozumie definicję i zastosowanie typu wyliczeniowego. Potrafi stworzyć i użyć typ wyliczeniowy w prostym programie.	Uczeń potrafi skutecznie wykorzystać typ wyliczeniowy w różnych kontekstach programistycznych. Zna różne techniki związane z wyliczeniami, takie jak konwersja do typu int czy porównywanie wartości.	Uczeń wykazuje głębokie zrozumienie typu wyliczeniowego i potrafi wykorzystać go w zaawansowany sposób. Potrafi stosować wyliczenia w skomplikowanych problemach programistycznych, takich jak interakcje z innymi typami danych czy zastosowanie w klasach.	Uczeń doskonale opanował temat typu wyliczeniowego i potrafi wykorzystać go w najbardziej zaawansowanych scenariuszach. Zna wszystkie subtelności związane z typem wyliczeniowym i potrafi je efektywnie wykorzystać w optymalizacji czy tworzeniu bardziej zaawansowanych struktur danych.
Rozdział 3. Programowanie z użyciem wskaźników						

Operator adresu	& (ampersand).	Rozpoznawanie operatora adresu w kodzie, ale brak umiejętności jego używania.	Poprawne używanie operatora adresu w prostych sytuacjach.	Efektywne wykorzystanie operatora adresu w bardziej złożonych przypadkach.	Analiza i interpretacja kodu z zaawansowanym użyciem operatora adresu.	Samodzielne tworzenie i optymalizacja kodu z wykorzystaniem operatora adresu.
Wskaźniki	Wskaźnik, dereferencja.	Zrozumienie teorii wskaźników, ale trudności w praktycznym ich użyciu.	Poprawne korzystanie z wskaźników w prostych programach.	Efektywne korzystanie z wskaźników, w tym wskaźników do wskaźników, w bardziej skomplikowanych strukturach.	Analiza i interpretacja kodu zawierającego wskaźniki, identyfikacja potencjalnych problemów (np. wycieki pamięci).	Zaawansowane zastosowania wskaźników, w tym optymalizacja kodu i zarządzanie pamięcią.
Dynamiczna alokacja pamięci	Malloc, new, delete.	Zrozumienie teorii alokacji, ale brak praktycznych umiejętności.	Proste zastosowania dynamicznej alokacji i dealokacji pamięci.	Stosowanie dynamicznej alokacji w bardziej złożonych strukturach danych.	Identyfikacja problemów związanych z alokacją, jak wycieki pamięci czy "dzikie" wskaźniki.	Zaawansowane zarządzanie pamięcią i optymalizacja kodu z wykorzystaniem dynamicznej alokacji.
Rozdział 4. Tablice i wektory						
Tablice statyczne	Tablica jednowymiarowa, wielowymiarowa.	Proste operacje na tablicach: przypisanie wartości, odczyt.	Wykorzystanie tablic w prostych algorytmach.	Implementacja bardziej zaawansowanych algorytmów z użyciem tablic.	Optymalizacja kodu z wykorzystaniem tablic, analiza różnych technik dostępu.	Zaawansowane techniki i triki związane z tablicami statycznymi.
Tablice i wskaźniki	Indeks, adres.	Rozpoznawanie wskaźników jako adresów	Operacje arytmetyki wskaźnikowej	Implementacja zaawansowanych algorytmów	Analiza i interpretacja kodu zawierającego	Zaawansowane techniki korzystania z

		pierwszych elementów tablic.	związane z tablicami.	łączących wskaźniki i tablice.	tablice i wskaźniki.	tablic przez wskaźniki, optymalizacja kodu.
Tablice dynamiczne i wektory	Rezerwacja, zwalnianie.	Uczeń zna podstawową definicję tablicy dynamicznej i wektora. Może inicjować wektor, ale ma trudności z dodawaniem lub usuwaniem elementów.	Uczeń potrafi inicjować, dodawać i usuwać elementy z wektora. Zna podstawowe metody wektora, takie jak push_back, size czy empty.	Uczeń potrafi skutecznie korzystać z wektorów w różnych kontekstach programistycznych. Zna i rozumie metody dostępu, iteratory oraz inne funkcje członkowskie wektora.	Uczeń potrafi optymalizować działanie z wektorami, zna zagadnienia związane z pojemnością (capacity) i rozmiarem (size).	Uczeń ma głębokie zrozumienie działania tablic dynamicznych i wektorów, w tym ich wewnętrznej implementacji.
Pętla foreach	Iteracja, kolekcja.	Uczeń zna podstawową składnię pętli foreach, ale ma trudności z jej praktycznym zastosowaniem.	Uczeń potrafi używać pętli foreach do iterowania przez elementy kontenerów. Zna podstawową składnię i potrafi czytać kod wykorzystujący tę pętlę.	Uczeń skutecznie używa pętli foreach w różnych sytuacjach, takich jak przetwarzanie elementów wektora czy mapy.	Uczeń potrafi korzystać z pętli foreach w zaawansowanych sytuacjach, takich jak iterowanie przez elementy kontenerów z biblioteki standardowej czy własnych typów.	Uczeń ma głębokie zrozumienie działania pętli foreach i potrafi wykorzystywać ją w skomplikowanych scenariuszach.
Rozdział 5. Funkcje						
Deklarowanie i definiowanie funkcji	Prototyp, definicja.	Zrozumienie teoretyczne deklaracji i definicji, ale	Umiejętność deklarowania i definiowania prostych funkcji.	Zastosowanie bardziej skomplikowanych funkcji z	Analiza i interpretacja kodu zawierającego różne funkcje;	Zaawansowane techniki deklaracji i definicji, w tym szablony funkcji.

		trudności w praktyce.		wieloma parametrami i różnymi typami zwracanymi.	modyfikacja i poprawa funkcji napisanych przez innych.	
Wywołanie funkcji	Wywołanie, zwracanie wartości.	Wywoływanie prostych funkcji bez parametrów.	Umiejętność wywołania funkcji z różnymi argumentami.	Zastosowanie funkcji w bardziej złożonych konstrukcjach i algorytmach.	Analiza wywołań funkcji w kodzie, identyfikacja i poprawa problemów.	Zaawansowane techniki wywoływania funkcji, w tym przekazywanie funkcji jako argumentów.
Parametry funkcji	Argument, wartość domyślna.	Używanie podstawowych typów parametrów w funkcjach.	Stosowanie wskaźników i referencji jako parametrów.	Zastosowanie różnych metod przekazywania argumentów, takich jak lista inicjalizacyjna.	Optymalizacja kodu poprzez odpowiedni wybór metody przekazania parametrów.	Zaawansowane techniki zarządzania parametrami, w tym wykorzystanie "perfect forwarding".
Zmienne globalne i lokalne	Zakres, dostęp.	Uczeń rozpoznaje składnię zmiennych globalnych i lokalnych, ale ma trudności z ich właściwym zastosowaniem.	Uczeń rozumie podstawową różnicę między zmiennymi globalnymi a lokalnymi oraz zna ich zakresy.	Uczeń rozumie i stosuje w praktyce zarówno zmienne globalne, jak i lokalne. Zna wady i zalety korzystania z zmiennych globalnych oraz rozumie potencjalne problemy	Uczeń skutecznie korzysta ze zmiennych lokalnych w różnych kontekstach, minimalizując użycie zmiennych globalnych.	Uczeń ma głębokie zrozumienie zakresu, żywotności i widoczności zmiennych w różnych kontekstach w języku C++.

Funkcje przeciążone	Przeciążenie.	Uczeń rozpoznaje składnię przeciążonych funkcji, ale ma trudności z ich właściwym definiowaniem i wywoływaniem.	Uczeń potrafi poprawnie zdefiniować i wywołać proste przeciążone funkcje. Rozumie podstawowe zastosowania przeciążenia funkcji.	Uczeń potrafi stosować przeciążenie funkcji w praktycznych sytuacjach, w tym różnić funkcje na podstawie ich listy parametrów.	Uczeń ma głębokie zrozumienie mechanizmu przeciążania funkcji w C++. Potrafi analizować potencjalne problemy związane z przeciążeniem, takie jak niejednoznaczności w wywołaniach.	Uczeń stosuje zaawansowane techniki przeciążania funkcji, integrując je z innymi konceptami C++, takimi jak przeciążanie operatorów.
Funkcje rekurencyjne	Rekursja.	Uczeń rozpoznaje składnię funkcji rekurencyjnej, ale ma trudności z jej implementacją lub rozumieniem mechanizmu działania.	Uczeń potrafi napisać proste funkcje rekurencyjne, takie jak obliczanie silni lub ciągu Fibonacciego.	Uczeń potrafi stosować rekursję w bardziej złożonych problemach, takich jak przeszukiwanie drzew lub rozwiązania problemów typu "podziel i zwyciężaj".	Uczeń ma głębokie zrozumienie mechanizmu rekursji i potrafi optymalizować funkcje rekurencyjne.	Uczeń stosuje zaawansowane techniki rekursji, integrując je z innymi konceptami C++, takimi jak rekursja ogonowa.
Wymagania edukacyjne niezbędne do otrzymania przez ucznia rocznej oceny klasyfikacyjnej (obejmują wymagania edukacyjne niezbędne do otrzymania przez ucznia śródrocznej oceny klasyfikacyjnej).						
Rozdział 6. Preprocesor						
Dyrektywa #include	Biblioteki, nagłówki.	Podstawowe zastosowanie #include do załączania	Załączanie własnych plików nagłówkowych.	Zrozumienie i zarządzanie zależnościami między różnymi	Optymalizacja struktury kodu poprzez efektywne	Zaawansowane techniki zarządzania plikami

		bibliotek standardowych.		plikami nagłówkowymi.	wykorzystanie #include.	nagłówkowymi, minimalizowanie redundancji.
Dyrektywy kompilacji warunkowej	Warunki, kompilacja.	Uczeń rozumie podstawowe pojęcie dyrektyw kompilacji warunkowej, ale ma trudności z ich prawidłowym zastosowaniem.	Uczeń potrafi stosować podstawowe dyrektywy, takie jak #ifdef i #ifndef, do kontroli kompilacji fragmentów kodu.	Uczeń potrafi skutecznie korzystać z dyrektyw #if, #elif oraz zdefiniowanych makr, aby kontrolować bardziej złożone warunki kompilacji.	Uczeń posiada głębokie zrozumienie zastosowania dyrektyw kompilacji warunkowej w złożonych projektach i potrafi zoptymalizować kod za ich pomocą.	Uczeń demonstruje zaawansowane zastosowanie dyrektyw kompilacji warunkowej w skomplikowanych scenariuszach, integrując je z innymi zaawansowanymi funkcjami języka C++.
Rozdział 7. Funkcje biblioteczne						
Funkcje matematyczne	Matematyka, sinus, cosinus.	Podstawowe użycie wybranych funkcji matematycznych, takich jak sqrt czy sin.	Zastosowanie większości podstawowych funkcji matematycznych w prostych programach.	Zastosowanie bardziej zaawansowanych funkcji matematycznych oraz kombinowanie różnych funkcji.	Analiza i interpretacja kodu zawierającego różne funkcje matematyczne; optymalizacja wykorzystania tych funkcji.	Zaawansowane techniki matematyczne, tworzenie własnych rozszerzeń funkcji bibliotecznych.
Funkcje znakowe	ASCII, znaki.	Proste manipulacje na znakach za pomocą funkcji bibliotecznych.	Użycie funkcji do przetwarzania ciągów znaków.	Połączenie różnych funkcji znakowych do skomplikowanego przetwarzania tekstu.	Analiza kodu zawierającego manipulacje znakowe, optymalizacja.	Zaawansowane techniki przetwarzania tekstu z wykorzystaniem funkcji bibliotecznych.

Konwersja typu danych	Cast, konwersja.	Uczeń rozumie podstawową ideę konwersji typów i potrafi zastosować prostą konwersję C-stylową, ale może nie rozumieć konsekwencji niejawnych konwersji.	Uczeń potrafi wykonywać podstawowe konwersje typów, zarówno jawne, jak i niejawne. Rozpoznaje i potrafi zastosować rzutowanie w stylu C++ przy użyciu static_cast.	Uczeń dobrze rozumie różnice między różnymi metodami rzutowania w C++ i potrafi je stosować w odpowiednich sytuacjach.	Uczeń potrafi stosować zaawansowane metody rzutowania, takie jak dynamic_cast, const_cast i reinterpret_cast, i rozumie, kiedy i dlaczego je stosować.	Uczeń posiada dogłębną wiedzę na temat konwersji typów w C++ i demonstruje umiejętność stosowania tej wiedzy w złożonych i nietypowych sytuacjach.
Funkcje wejścia/wyjścia	Cin, cout, printf.	Uczeń potrafi korzystać z podstawowych funkcji cout i cin do prostych operacji wydruku i wczytywania danych z konsoli.	Uczeń efektywnie korzysta z funkcji cout i cin, stosując różne modyfikatory formatowania (np. setw, setprecision).	Uczeń potrafi korzystać z zaawansowanych funkcji I/O, takich jak fstream do czytania i zapisu do plików.	Uczeń potrafi diagnozować i rozwiązywać problemy związane z funkcjami I/O, takie jak błędy odczytu/zapisu.	Uczeń posiada dogłębną wiedzę na temat mechaniki I/O w C++ i potrafi optymalizować operacje wejścia/wyjścia dla różnych zastosowań.
Rozdział 8. Klasy i obiekty						
Wprowadzenie do programowania obiektowego	Obiektowość, kapsułkowanie.	Teoretyczne zrozumienie pojęć klasy i obiektu.	Deklaracja prostych klas i tworzenie obiektów.	Projektowanie klas z różnymi metodami i atrybutami, tworzenie skomplikowanych obiektów.	Analiza kodu obiektowego, optymalizacja i refaktoryzacja.	Zaawansowane techniki programowania obiektowego, tworzenie zaawansowanych hierarchii klas.
Definiowanie klas	Członkowie, metody.	Uczeń potrafi zdefiniować prostą klasę z kilkoma	Uczeń umie definiować klasy z różnymi poziomami dostępu (public,	Uczeń zna i potrafi stosować mechanizmy dziedziczenia i	Uczeń zna i potrafi definiować metody statyczne, atrybuty klas i inne	Uczeń ma dogłębne zrozumienie mechanizmów

		atrybutami i metodami.	private, protected). Potrafi korzystać z konstruktorów do inicjalizacji obiektów klasy.	kompozycji w definicjach klas. Potrafi korzystać z różnych rodzajów konstruktorów, w tym konstruktorów kopiujących.	zaawansowane funkcje klas.	działania klas w C++, w tym mechanizmów takich jak przeciążanie operatorów w kontekście klas.
Deklarowanie zmiennych obiektowych	Instancja.	Uczeń potrafi zadeklarować prostą zmienną obiektową danej klasy, ale może popełniać błędy podczas inicjalizacji.	Uczeń umie poprawnie deklarować i inicjować zmienne obiektowe różnych klas. Rozumie, jak używać konstruktorów podczas deklaracji obiektów.	Uczeń potrafi deklarować zmienne obiektowe, które używają dziedziczenia i kompozycji. Zna różnicę między deklaracją a inicjalizacją zmiennej obiektowej.	Uczeń potrafi efektywnie korzystać z różnych konstruktorów podczas deklaracji zmiennych obiektowych.	Uczeń ma dogłębne zrozumienie wszystkich aspektów deklarowania zmiennych obiektowych, w tym zaawansowanych technik i optymalizacji.
Odwołania do elementów członkowskich obiektów	Dostęp, modyfikacja.	Uczeń potrafi podstawowo odwołać się do publicznych atrybutów i metod obiektów, ale może mieć trudności z rozróżnieniem pomiędzy atrybutami a metodami.	Uczeń prawidłowo odwołuje się do atrybutów i metod obiektów za pomocą operatora .. Rozumie różnicę między odwołaniem się do atrybutu a wywołaniem metody obiektu.	Uczeń umie korzystać z operatora -> do odwoływania się do elementów członkowskich obiektów poprzez wskaźniki.	Uczeń demonstruje zaawansowane zrozumienie odwołań do elementów członkowskich, w tym do dziedziczonych atrybutów i metod w klasach pochodnych.	Uczeń ma głęboką wiedzę na temat odwoływania się do elementów członkowskich w różnych kontekstach, w tym w przypadku zaawansowanych technik programowania obiektowego,

						takich jak przeciążanie operatorów, dziedziczenie wielokrotne i metaprogramowanie.
Rozdział 9. Tworzenie i inicjowanie obiektów						
Konstruktory	Tworzenie, inicjalizacja.	Zastosowanie podstawowych konstruktorów bez parametrów.	Definiowanie konstruktorów z parametrami i ich wykorzystanie.	Użycie konstruktorów kopiujących i list inicjalizacyjnych.	Analiza i modyfikacja konstruktorów w skomplikowanym kodzie.	Zaawansowane techniki związane z konstruktorami, w tym konstruktory przenoszące.
Inicjalizacja obiektów	Konstruktor domyślny, parametryczny.	Uczeń potrafi inicjalizować podstawowe typy danych za pomocą bezpośredniego przypisania wartości.	Uczeń rozumie i stosuje konstruktory do inicjalizacji obiektów klasy. Potrafi używać list inicjalizacyjnych do przypisania wartości do atrybutów klasy.	Uczeń potrafi inicjalizować obiekty z różnych klas, w tym klasy z dziedziczeniem. Zna różne metody inicjalizacji, takie jak jednoargumentowe konstruktory i konstruktory kopiujące.	Uczeń zna i stosuje zaawansowane techniki inicjalizacji, takie jak inicjalizacja delegowana, inicjalizacja za pomocą list inicjalizacyjnych oraz inicjalizacja domyślna.	Uczeń ma głęboką wiedzę na temat wszystkich aspektów inicjalizacji obiektów w C++ i potrafi skutecznie wykorzystać tę wiedzę w praktyce, stosując najlepsze praktyki branżowe.
Konstruktor kopiujący	Kopiowanie, klony.	Uczeń ma świadomość istnienia konstruktora kopiującego, ale nie potrafi	Uczeń potrafi zaimplementować podstawowy konstruktor kopiujący dla prostych klas, które	Uczeń umiejętnie implementuje konstruktor kopiujący dla klas, które	Uczeń potrafi zidentyfikować sytuacje, w których domyślny konstruktor kopiujący jest	Uczeń ma zaawansowaną wiedzę na temat konstruktorów kopiujących w kontekście

		poprawnie go zaimplementować dla większości klas.	nie zawierają wskaźników lub dynamicznie alokowanej pamięci.	zawierają wskaźniki i dynamicznie alokowaną pamięć, dbając o odpowiednie zarządzanie pamięcią.	niewystarczający i wymaga własnej implementacji.	zaawansowanych konceptów języka C++, takich jak przeciążanie operatorów i dziedziczenie.
Delegowanie konstruktorów	Delegowanie, wielość.	Uczeń zna ideę delegowania konstruktorów, ale ma problemy z jej poprawnym stosowaniem.	Uczeń potrafi zaimplementować prosty przypadek delegowania konstruktorów dla klas, w których jest to odpowiednie.	Uczeń skutecznie stosuje delegowanie konstruktorów w bardziej złożonych klasach, minimalizując duplikację kodu.	Uczeń potrafi analizować i oceniać, kiedy delegowanie konstruktorów jest właściwe, a kiedy może prowadzić do problemów.	Uczeń posiada dogłębną wiedzę na temat delegowania konstruktorów, w tym zaawansowane przypadki użycia i interakcje z innymi aspektami języka C++.
Destruktory	Czyszczenie, zwalnianie.	Uczeń rozumie podstawową koncepcję destruktora, ale ma trudności z określeniem, kiedy i jak go stosować.	Uczeń potrafi zaimplementować prosty destruktor w klasie, by zwolnić przydzielone dynamicznie zasoby.	Uczeń potrafi zaimplementować destruktory w bardziej złożonych klasach i hierarchiach dziedziczenia.	Uczeń ma głębokie zrozumienie dla roli destruktorów w języku C++ i potrafi zaimplementować je w zaawansowanych scenariuszach, takich jak smart wskaźniki czy własne klasy zarządzające zasobami.	Uczeń posiada dogłębną wiedzę na temat destruktorów, ich interakcji z innymi elementami języka (np. konstruktory kopiujące, operatory przypisania) oraz potrafi radzić sobie z trudnymi problemami związanymi z zarządzaniem pamięcią i

						zasobami.
Rozdział 10. Mechanizm dziedziczenia						
Definicja relacji dziedziczenia	Hierarchia, bazowy, pochodny.	Teoretyczna wiedza o dziedziczeniu bez umiejętności praktycznego zastosowania.	Proste zastosowania dziedziczenia w kodzie.	Projektowanie hierarchii klas z użyciem dziedziczenia.	Analiza, optymalizacja i refaktoryzacja kodu wykorzystującego dziedziczenie.	Zaawansowane techniki dziedziczenia, wielodziedziczenie, wirtualne klasy bazowe.
Rodzaje dziedziczenia	Pojedynczy, wielokrotny.	Uczeń rozumie podstawowy koncept dziedziczenia, ale ma trudności z określeniem różnic między poszczególnymi typami dziedziczenia.	Uczeń zna i rozumie trzy podstawowe rodzaje dziedziczenia w C++: publiczne, chronione i prywatne.	Uczeń potrafi prawidłowo zastosować wszystkie trzy rodzaje dziedziczenia w praktyce. Rozumie implikacje zastosowania każdego z typów dziedziczenia i potrafi je uzasadnić.	Uczeń ma głębokie zrozumienie dla dziedziczenia wielobazowego i potencjalnych problemów z nim związanych, takich jak diamentowy problem dziedziczenia.	Uczeń posiada dogłębną wiedzę na temat dziedziczenia w języku C++, w tym złożonych interakcji między dziedziczeniem a innymi mechanizmami języka.
Rozdział 11. Mechanizm abstrakcji						
Klasy abstrakcyjne	Czystość, koncept.	Teoretyczna wiedza na temat klas abstrakcyjnych bez zdolności do ich implementacji.	Definiowanie prostych klas abstrakcyjnych i dziedziczenie po nich.	Tworzenie skomplikowanych hierarchii klas z użyciem klas abstrakcyjnych.	Analiza, optymalizacja i refaktoryzacja kodu z klasami abstrakcyjnymi.	Zaawansowane techniki związane z klasami abstrakcyjnymi, w tym wirtualne funkcje czysto abstrakcyjne.

Interfejsy	Polimorfizm, kontrakt.	Zrozumienie idei interfejsu, ale brak umiejętności jego zastosowania w praktyce.	Implementacja i użycie prostych interfejsów.	Projektowanie bardziej skomplikowanych interfejsów i ich zaawansowane zastosowanie.	Analiza i optymalizacja kodu z wykorzystaniem interfejsów.	Zaawansowane techniki i wzorce projektowe z wykorzystaniem interfejsów.
Abstrakcja danych	Kapsułkowanie, ukrywanie.	Uczeń ma ogólną świadomość, czym jest abstrakcja, ale nie potrafi jej zastosować w praktyce.	Uczeń rozumie ideę ukrywania szczegółów implementacji i potrafi zdefiniować prostą klasę z enkapsulacją danych.	Rozumie różnicę między interfejsem a implementacją klasy i potrafi wyjaśnić, jak abstrakcja ułatwia pisanie czytelnego i elastycznego kodu.	Uczeń potrafi projektować zaawansowane systemy z wykorzystaniem abstrakcji, takie jak hierarchie klas czy wzorce projektowe.	Jest w stanie analizować, projektować i wdrażać zaawansowane systemy, które skutecznie wykorzystują abstrakcję do osiągnięcia większej modularności, czytelności i elastyczności kodu.
Rozdział 12. Funkcje i klasy zaprzyjaźnione						
Funkcje zaprzyjaźnione	Dostęp, przyjaźń.	Teoretyczne zrozumienie funkcji zaprzyjaźnionych bez zdolności praktycznej implementacji.	Użycie funkcji zaprzyjaźnionych w prostych scenariuszach.	Zaawansowane zastosowania funkcji zaprzyjaźnionych w kodzie.	Analiza i refaktoryzacja kodu zawierającego funkcje zaprzyjaźnione.	Optymalizacja kodu i zaawansowane techniki wykorzystujące funkcje zaprzyjaźnione.
Klasy zaprzyjaźnione	Relacje, współpraca.	Uczeń wie, że istnieje coś takiego jak klasy zaprzyjaźnione,	Uczeń rozumie podstawową koncepcję klas zaprzyjaźnionych i	Zna wady i zalety korzystania z klas	Uczeń ma zaawansowaną wiedzę na temat klas	Uczeń posiada głęboką wiedzę na temat klas zaprzyjaźnionych i

		ale nie rozumie ich działania ani zastosowania.	wie, jak zadeklarować jedną klasę jako zaprzyjaźnioną wobec drugiej.	zaprzyjaźnionych oraz potrafi je kontrastować z alternatywnymi podejściami do dostępu do członków klasy.	zaprzyjaźnionych i potrafi stosować je w złożonych projektach.	jest w stanie analizować oraz projektować zaawansowane systemy z ich wykorzystaniem.
Rozdział 13. Obsługa błędów i wyjątków						
System komunikatów i kodów zwrotnych	Błędy, komunikaty.	Podstawowe wykorzystanie kodów zwrotnych w programie.	Implementacja prostych mechanizmów obsługi błędów w kodzie.	Zaawansowane techniki obsługi błędów i komunikacji o błędach.	Analiza, interpretacja i poprawa mechanizmów obsługi błędów w zaawansowanym kodzie.	Optymalizacja i tworzenie zaawansowanych systemów obsługi błędów.
Mechanizm obsługi wyjątków	Try, catch, throw.	Uczeń ma podstawową świadomość, co to jest obsługa wyjątków, ale nie potrafi jej zastosować w praktyce.	Uczeń rozumie podstawowe pojęcia związane z obsługą wyjątków, takie jak bloki try i catch.	Uczeń zna różne rodzaje wyjątków i potrafi stosować je w praktyce. Rozumie, jak przechwytywać różne typy wyjątków i jak korzystać z informacji zawartych w wyjątku.	Uczeń potrafi korzystać z mechanizmu obsługi wyjątków w złożonych aplikacjach, w tym stosować wielopoziomową obsługę wyjątków.	Uczeń posiada głęboką wiedzę na temat mechanizmu obsługi wyjątków i potrafi stosować go w zaawansowanych sytuacjach oraz projektować własne rozwiązania bazujące na tym mechanizmie.
Rozdział 14. Przykłady implementacji wybranych algorytmów						
Wyznaczenie największego wspólnego dzielnika	Euklides, GCD.	Teoretyczne zrozumienie algorytmu bez	Prosta implementacja algorytmu	Optymalizacja i modyfikacja algorytmu NWD.	Analiza i interpretacja różnych wersji	Zaawansowane zastosowania i modyfikacje

		zdolności do jego implementacji.	wyznaczania NWD.		algorytmu NWD.	algorytmu NWD.
Sortowanie tablic	Sortowanie, porządkowanie.	Uczeń rozumie podstawowy koncept sortowania. Może manualnie posortować małą tablicę liczb w kolejności rosnącej, ale ma trudności z implementacją tego w kodzie.	Uczeń potrafi zaimplementować i zastosować podstawowe algorytmy sortowania, takie jak sortowanie bąbelkowe.	Uczeń potrafi zaimplementować bardziej zaawansowane algorytmy sortowania, takie jak sortowanie przez wstawianie lub sortowanie szybkie.	Uczeń ma zaawansowaną wiedzę na temat różnych algorytmów sortowania i jest w stanie wybrać odpowiedni algorytm w zależności od sytuacji.	Uczeń posiada dogłębną wiedzę na temat algorytmów sortowania, w tym bardziej zaawansowanych metod, takich jak sortowanie kubełkowe czy sortowanie przez zliczanie.
Wyszukiwanie binarne	Bisekcja, efektywność.	Zna różnicę między wyszukiwaniem liniowym a binarnym, ale ma trudności z implementacją wyszukiwania binarnego w kodzie.	Uczeń potrafi zaimplementować podstawowy algorytm wyszukiwania binarnego.	Uczeń potrafi zaimplementować wyszukiwanie binarne w różnych strukturach danych, takich jak tablice czy listy.	Uczeń potrafi optymalizować algorytm wyszukiwania binarnego dla różnych scenariuszy. Rozumie i potrafi analizować złożoność czasową wyszukiwania binarnego.	Uczeń posiada dogłębną wiedzę na temat wyszukiwania binarnego, w tym zaawansowanych wariantów i technik.

UWAGI:

1. Ocenę wyższą otrzymuje uczeń spełniający łącznie wymagania edukacyjne określone dla ocen niższych np. ocenę dobrą otrzymuje uczeń spełniający wymagania edukacyjne na oceną dopuszczającą, dostateczną oraz dobrą. W ramach przedmiotu: “Programowania strukturalnego i

obiektowego” uczeń powinien wykazać się umiejętnością wykorzystania wiedzy teoretycznej w praktyce.

2. Ocenę niedostateczną otrzymuje uczeń, który nie spełnia wymagań na poszczególne pozytywne oceny.

3. W przypadku nie zrealizowania tematów lekcji (zagadnień) w I okresie będą one realizowane po klasyfikacji śródrocznej. W tym przypadku obowiązują również wymagania edukacyjne dla tych tematów (zagadnień).