Informatyka inżynierska

dla szkół ponadpodstawowych

Tabela wymagań na poszczególne oceny

Spis treści

Klasa trzecia. Moduł C. Wokół algorytmiki i programowania

Uwaga: Zgodnie z podstawa programową do informatyki dla szkół ponadpodstawowych**: „*Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:*”**

Prezentacja algorytmu liniowego w wybranej notacji				
2	**3**	**4**	**5**	**6**
Uczeń:	**Uczeń:**	**Uczeń:**	**Uczeń:**	**Uczeń:**
zna sposoby prezentacji algorytmów; testuje działanie algorytmu liniowego zapisanego w postaci listy kroków i przedstawionego w postaci schematu blokowego; potrafi narysować (odręcznie) schemat blokowy algorytmu liniowego; zapisuje prosty algorytm liniowy w wybranym języku programowania	zna pojęcie specyfikacji zadania i potrafi zapisać specyfikację zadania; zna i stosuje zasady tworzenia listy kroków; przedstawia algorytm liniowy w postaci listy kroków; zna podstawowe zasady graficznego prezentowania algorytmów: podstawowe rodzaje bloków, ich przeznaczenie i sposoby umieszczania w schemacie blokowym; podczas rysowania schematów blokowych potrafi wykorzystać **Kształty** z edytora tekstu; pisze programy komputerowe realizujące dane algorytmy na podstawie ich list kroków i/lub schematów blokowych	przedstawia dokładną specyfikację dowolnego zadania. analizuje poprawność budowy schematu blokowego; objaśnia dobrany algorytm, uzasadnia poprawność rozwiązania na wybranych przykładach danych; realizuje przykładowy algorytm liniowy w wybranym języku programowania na podstawie ich list kroków i schematów blokowych; testuje program dla różnych danych	potrafi samodzielnie zapoznać się z nowym programem edukacyjnym przeznaczonym do konstrukcji schematów blokowych; potrafi przeprowadzić szczegółową analizę poprawności konstrukcji schematu blokowego; analizuje działanie algorytmu dla przykładowych danych; potrafi wskazać i poprawić błędy w liście kroków czy w schemacie blokowym; Samodzielnie pisze program realizujący algorytm liniowy.	zapisuje specyfikację zadania i przedstawia w postaci listy kroków i schematu blokowego rozwiązanie trudniejszego zadania; w wybranym języku programowania (C++ lub Python) pisze trudniejsze programy realizujące algorytmy przedstawione w postaci list kroków i schematów blokowych; uczestniczy w konkursach/olimpiadach informatycznych

Prezentacja algorytmu z warunkami w wybranej notacji				
2	**3**	**4**	**5**	**6**
Uczeń:	**Uczeń:**	**Uczeń:**	**Uczeń:**	**Uczeń:**
określa sytuacje warunkowe; podaje przykłady zadań, w których występują sytuacje warunkowe; wie, jaka figura reprezentuje sytuacje warunkową; potrafi narysować (odręcznie) schemat blokowy nietrudnego algorytmu z warunkiem prostym	analizuje działanie algorytmu z warunkami zapisanego w postaci graficznej. zna i stosuje zasady tworzenia listy kroków algorytmu z warunkami; potrafi narysować schemat blokowy algorytmu z warunkami, korzystając z wybranego narzędzia; pisze program w wybranym języku programowania realizujący algorytm z warunkami prostymi; zna i stosuje instrukcję warunkową	przedstawia algorytm z warunkami w postaci listy kroków; analizuje listę kroków i schemat blokowy algorytmu z warunkami, testując go dla wybranych danych; potrafi zapisać warunek złożony; korzystając z przykładu, zapisuje w postaci programu algorytm z warunkami złożonymi; zna i omawia warunek istnienia trójkąta	Zapisuje algorytmy z pętlą zagnieżdżoną. testuje działanie algorytmu z warunkami zagnieżdżonymi zapisanego w postaci listy kroków na wybranych przykładach danych; buduje schemat blokowy algorytmu sprawdzania warunku trójkąta na podstawie listy kroków; zapisuje w postaci programu algorytm z warunkami zagnieżdżonymi	wymyśla samodzielnie problem z warunkami zagnieżdżonymi, zapisuje jego specyfikację, listę kroków, tworzy schemat blokowy i program w wybranym języku programowania korzystając z dodatkowych źródeł, znajduje inny, niż podany w podręczniku, sposób sprawdzenia, czy z danych trzech odcinków można zbudować trójkąt i zapisuje ten algorytm w postaci programu komputerowego; w wybranym języku programowania pisze trudniejsze programy realizujące algorytmy z warunkami (w tym zagnieżdżonymi) przedstawione w postaci list kroków i schematów blokowych

Prezentacja algorytmu iteracyjnego w wybranej notacji				
2	**3**	**4**	**5**	**6**
Uczeń:	**Uczeń:**	**Uczeń:**	**Uczeń:**	**Uczeń:**
analizuje listę kroków algorytmu iteracyjnego, testując go dla wybranych danych; analizuje schemat algorytmu iteracyjnego, testując go dla wybranych danych;	zna pojęcie iteracji i rozumie pojęcie algorytmu iteracyjnego; podaje przykłady algorytmów iteracyjnych; tworzy schemat blokowy algorytmu z warunkiem prostym i pętlą; testuje rozwiązanie dla wybranych danych; zna i stosuje instrukcję iteracyjną **`for`** w wybranym języku programowania; zapisuje prosty algorytm iteracyjny w postaci programu w wybranym języku programowania	analizuje algorytmy, w których występują powtórzenia (iteracje); ocenia zgodność algorytmu ze specyfikacją; analizuje listę kroków i schemat blokowy algorytmu z pętlą zagnieżdżoną, testując go dla wybranych danych; zapisuje algorytm iteracyjny w postaci programu w wybranym języku programowania	zapisuje w postaci listy kroków algorytmów z warunkami i iteracyjne; pisze listę kroków i tworzy schemat blokowy algorytmu z pętlą zagnieżdżoną; ocenia zgodność algorytmu ze specyfikacją problemu; wie, kiedy należy zastosować pętlę zagnieżdżoną; zapisuje w postaci programu wybrany algorytm z pętlą zagnieżdżoną; testuje program dla wybranych danych	przestrzega zasad zapisu algorytmów w zadanej postaci (notacji); stosuje listy kroków i schematy blokowe w opisie zadań (problemów) z innych przedmiotów szkolnych oraz różnych dziedzin życia; wskazuje podobieństwa i różnice dotyczące działania instrukcji warunkowych i iteracyjnej **`for`** w różnych językach programowania; zapisuje trudniejsze algorytmy iteracyjne w wybranym języku programowania; zapisuje programy w czytelnej postaci – stosuje wcięcia, komentarze; uczestniczy w konkursach/olimpiadach informatycznych

Funkcje w wybranym języku programowania				
2	**3**	**4**	**5**	**6**
Uczeń:	**Uczeń:**	**Uczeń:**	**Uczeń:**	**Uczeń:**
Wie, czym są podprogramy; W wybranym języku programowania definiuje prostą funkcję niezwracająca wartości bez parametrów. Wie, jak wywołać funkcję niezwracająca wartości bez parametrów w programie. Wywołuje taka funkcję w programie.	Wie, na czym polega programowanie strukturalne; Pisze programy, stosując funkcje. Definiuje funkcje niezwracające wartości bez parametrów i z parametrami. Stosuje te funkcje w programach. Wyjaśnia pojęcia zmienna lokalna i zmienna globalna.	Wymienia modele programowania; Rozumie i stosuje zasady programowania strukturalnego. Wyjaśnia, czym są parametry funkcji. Wyjaśnia, czym jest funkcja zwracająca wartość. Porównuje do funkcji niezwracającej wartości. Wskazuje różnice. Wie, na czym polega wywołanie funkcji z parametrami i wywołuje taką funkcję w programach. Definiuje funkcje zwracające wartości bez parametrów i z parametrami oraz stosuje je w programach. Wie, co to jest zasięg zmiennej i jakie ma znaczenie. Deklaruje odpowiednio zmienne lokalne i globalne w programach.	Omawia modele programowania; Rozumie zasady postępowania przy rozwiązywaniu problemu metodą zstępującą. Wyjaśnia, czym różni się programowanie zstępujące od wstępującego. Definiuje funkcje niezwracające wartości i zwracające wartość bez parametrów i z parametrami oraz stosuje je w programach. Wyjaśnia, na czym polega przesłanianie zmiennych globalnych. Wyjaśnia, na czym polega przesłanianie parametrów funkcji. Modyfikuje programy, stosuje zmienne globalne i lokalne, objaśnia otrzymane wyniki.	Porównuje modele programowania, wskazując różnice. Sprawnie definiuje i stosuje funkcje niezwracające wartości i zwracające wartość bez parametrów i z parametrami w programach. Rozwiązuje przykładowe zadania z matury i olimpiady informatycznej. Potrafi, na przykładzie programu utworzonego według własnego pomysłu, wyjaśnić różnice w stosowaniu zmiennych lokalnych i globalnych, omówić zasięg zmiennych i przesłanianie zmiennych. Potrafi, na przykładzie programu utworzonego według własnego pomysłu, wyjaśnić przesłanianie parametrów funkcji.

Tablice w wybranym języku programowania				
2	**3**	**4**	**5**	**6**
Uczeń:	**Uczeń:**	**Uczeń:**	**Uczeń:**	**Uczeń:**
Analizuje i omawia gotowe przykłady programów, w których są użyte tablice lub listy. Potrafi wyjaśnić, dlaczego do rozwiązania niektórych zadań należy użyć tablic (list).	Zna pojęcia: *tablica*, *lista*, *zmienna indeksowana*. Na bazie przykładów z podręcznika, deklaruje tablicę i/lub listę, wczytuje i wyprowadza elementy tablicy i/lub listy, definiując odpowiednie funkcje w wybranym języku programowania.	Rozróżnia struktury danych: proste i złożone. Podaje przykłady. Wczytuje i wyprowadza elementy tablicy i/lub listy. Definiuje odpowiednie funkcje. Wie, jak odwołać się do elementu tablicy i/lub listy. Potrafi zastosować tablicę (listę) w zadaniach oraz modyfikować program, znaleźć błędy i je poprawić.	Rozumie, na czym polega dobór struktur danych do algorytmu i tworzy programy, dobierając odpowiednie struktury danych do programu. Deklaruje tablicę dwuwymiarową w języku C++. Definiuje odpowiednie funkcje. Definiuje różne listy (w tym dwuwymiarowe) w języku Python. Tworzy programy, dobierając odpowiednie struktury danych do programu.	Omawia podobieństwa i różnice w definiowaniu tablic w języku C++ i list w języku Python Stosuje w programach listy jednowymiarowe i dwuwymiarowe, odpowiednio dobierając określoną strukturę danych (tu: rodzaj listy) do algorytmu. Rozwiązuje przykładowe zadania z konkursów i olimpiady informatycznej, w których należy zastosować tablice i/lub listy i funkcje. Dobiera najlepszy algorytm i odpowiednie struktury danych do rozwiązania postawionego problemu.

Stosowanie instrukcji iteracyjnych				
2	**3**	**4**	**5**	**6**
Uczeń:	**Uczeń:**	**Uczeń:**	**Uczeń:**	**Uczeń:**
Analizuje i omawia działanie gotowych programów zapisanych w wybranym języku programowania, zawierających instrukcję pętli **`for`**. Analizuje i omawia działanie gotowych programów zapisanych w wybranym języku programowania, zawierających instrukcję pętli **`while`**.	Zna pojęcie iteracji i rozumie pojęcie algorytmu iteracyjnego. Podaje ich przykłady. Zna postać i działanie instrukcji iteracyjnej **`while`** w językach C++ i/lub Python i stosuje ją w tworzonych programach komputerowych.	Zna sposoby zakończenia iteracji. Określa kroki iteracji. Stosuje instrukcję **`while`** w programach komputerowych. W języku C++ stosuje instrukcję **`do … while`** w programach komputerowych. Zna różne sposoby określania liczby iteracji w języku Python, np. poprzez powtarzanie poleceń zawartych wewnątrz pętli dla konkretnych wartości lub poprzez podanie liczby powtórzeń z zastosowaniem funkcji **`range()`**.	Rozumie różnicę pomiędzy instrukcją **`for`** a instrukcją **`while`** w wybranym języku programowania. Modyfikuje programy, zamieniając pętlę **`for`** na **`while`** i odwrotnie. Ocenia program po zmianie (styl, czytelność). Pisze programy, stosując poznane instrukcje iteracyjne. Dobiera odpowiednią technikę algorytmiczną i odpowiednie struktury danych do rozwiązywanego problemu.	Potrafi samodzielnie zastosować odpowiedni rodzaj instrukcji pętli w tworzonym programie. Potrafi samodzielnie dobrać odpowiednią instrukcję **`while`** lub **`do… while`**. Omawia podobieństwa i różnice w działaniu wszystkich omówionych instrukcji pętli w języku C++. Sprawnie pisze trudniejsze programy, stosując poznane instrukcje iteracyjne. Wymyśla samodzielnie problem iteracyjny, formułuje zadanie, pisze jego specyfikację i listę kroków oraz zapisuje w wybranym języku programowania.

Iteracyjna realizacja wybranych algorytmów, w tym algorytmów wyszukiwania				
2	**3**	**4**	**5**	**6**
Uczeń:	**Uczeń:**	**Uczeń:**	**Uczeń:**	**Uczeń:**
Potrafi odróżnić algorytm liniowy od algorytmu iteracyjnego. Omawia algorytm znajdowania elementu najmniejszego. Analizuje listę kroków i rysuje schemat blokowy na jej podstawie. Analizuje i omawia gotową listę kroków i schemat blokowy algorytmu Euklidesa w jednej z wersji. Testuje algorytm na podstawie listy lub schematu.	Zna przykładowe algorytmy na liczbach naturalnych; Wie, na czym polega metoda wyszukiwania liniowego i przez połowienie. Zapisuje algorytm znajdowania najmniejszego (największego) elementu w postaci programu. Zna iteracyjną postać algorytmu Euklidesa w wersji z odejmowaniem. Potrafi napisać listy kroków algorytmu i narysować schemat blokowy Euklidesa w wersji z odejmowaniem. Zna metodę „dziel i zwyciężaj”. Zapisuje algorytm Euklidesa w wersji z odejmowaniem w postaci programu w wybranym języku programowania.	Potrafi omówić algorytm naiwny i optymalny jednoczesnego znajdowania największego i najmniejszego elementu w zbiorze. Zna iteracyjną postać algorytmu Euklidesa z resztą z dzielenia. Pisze listę kroków tego algorytmu. Potrafi narysować schemat blokowy algorytmu Euklidesa w wersji z resztą z dzielenia. Zapisuje algorytm Euklidesa w wersji z resztą z dzielenia w postaci programu w wybranym języku programowania. Wyjaśnia na przykładzie różnicę między wersją algorytmu Euklidesa z odejmowaniem a wersją z resztą z dzielenia.	Określa liczbę porównań w algorytmie naiwnym i optymalnym znajdowania największego i najmniejszego elementu w zbiorze. Porównuje otrzymane wyniki. Pisze listę kroków algorytmu jednoczesnego znajdowania minimum i maksimum z wykorzystaniem metody dziel i zwyciężaj. Zapisuje ten algorytm w postaci programu w języku C++ i/lub Python. Omawia zastosowanie schematu Hornera do obliczania wartości wielomianu; pisze listę kroków, rysuje schemat blokowy tego algorytmu i pisze program realizujący algorytm obliczania wartości wielomianu według schematu Hornera. Programując w/w algorytmy, definiuje odpowiednie funkcje, dobiera struktury danych. Dba o stosowanie podstawowych zasad programowania.	Podaje przykłady problemów, w których można zastosować wyszukiwanie liniowe lub przez połowienie. Pisze trudniejsze programy komputerowe, w których wykorzystuje poznane algorytmy. Korzystając z dodatkowych źródeł, wyszukuje informacje o zastosowaniu metody „dziel i zwyciężaj” oraz pisze program według własnego pomysłu pokazujący zastosowanie tej metody. Samodzielnie zapoznaje się ze schematem Hornera i zapisuje go w postaci programu, dopierając poprawne struktury danych. Korzystając z dodatkowych źródeł, omawia przykłady zastosowań algorytmu Euklidesa i poznaje trudniejsze algorytmy, np. trwałego małżeństwa, problem ośmiu hetmanów, algorytm znajdowania liczb bliźniaczych. Potrafi zapisać je w języku programowania.

Elementy analizy algorytmów				
2	**3**	**4**	**5**	**6**
Uczeń:	**Uczeń:**	**Uczeń:**	**Uczeń:**	**Uczeń:**
Wymienia własności algorytmów. Potrafi przeanalizować przebieg prostego algorytmu zapisanego w postaci listy kroków lub w postaci schematu blokowego dla przykładowych danych i ocenić w ten sposób jego poprawność.	Zna i omawia własności algorytmów. Wie, kiedy algorytm jest poprawny. Potrafi przeanalizować przebieg algorytmu (np. obliczania silni) zapisanego w postaci listy kroków lub w postaci schematu blokowego dla przykładowych danych i ocenić w ten sposób jego poprawność. Analizuje program wybranego algorytmu (np. obliczania silni) i ocenia jego poprawność.	Wie, jak sprawdzić, czy algorytm jest skończony. Potrafi ocenić poprawność działania algorytmu i jego zgodność ze specyfikacją. Określa liczbę prostych działań zawartych w algorytmie. Określa liczbę prostych działań zawartych w algorytmie. Potrafi poprawić program, który jest niepoprawny.	Wie, kiedy algorytm jest skończony. Potrafi przeanalizować przebieg algorytmu zapisanego w postaci listy kroków lub w postaci schematu blokowego dla przykładowych danych i ocenić w ten sposób jego skończoność. Analizuje program realizujący wybrany algorytm i ocenia jego skończoność. Modyfikuje program, aby działał poprawnie. Sprawdza poznane własności algorytmów, rozwiązując zadania, m.in. uzasadnia skończoność algorytmu znajdowania największego wspólnego dzielnika (NWD) dwóch liczb naturalnych. Oblicz liczbę operacji porównania w algorytmie wyboru minimum z tablicy zawierającej *n* losowo uporządkowanych liczb.	Potrafi samodzielnie ocenić poprawność i skończoność wybranych algorytmów. Potrafi samodzielnie wymyśleć zadanie (problem), napisać specyfikację zadania, listę kroków i program realizujący to zadanie. Korzysta samodzielnie z dodatkowej literatury fachowej. Poznane dodatkowe możliwości wybranego języka programowania (np. standardowe funkcje) i stosuje w programach. Rozwiązuje przykładowe zadania z olimpiady informatycznej.