# Znajdowanie określonego elementu w zbiorze w języku C++


Źródło: Alex Block, domena publiczna.

> W tym e-materiale powtarzamy wiadomości ze szkoły podstawowej.

Współczesne komputery przetwarzają ogromne ilości danych – kursy walut, ceny akcji na giełdzie, dane meteorologiczne itd. Informacje te byłyby jednak bezużyteczne, gdybyśmy nie mogli zweryfikować wystąpienia pewnych danych, np. sprawdzić, przez ile dni akcje spółki kosztowały mniej niż 50 zł lub czy w maju wystąpił jakikolwiek dzień, w którym temperatura w Polsce spadła poniżej zera. Aby uzyskać odpowiedzi na tego typu pytania, możemy wykorzystać algorytmy wyszukiwania.

W tym e-materiale poznasz implementację algorytmu wyszukiwania określonego elementu w zbiorze w języku C++. W e-materiale Znajdowanie określonego elementu w zbiorze znajdziesz ogólne informacje dotyczące szukania idola oraz lidera w zbiorze.

Implementacja w innych językach programowania:

- Znajdowanie określonego elementu w zbiorze w języku Java,
- Znajdowanie określonego elementu w zbiorze w języku Python.

Więcej zadań? Znajdowanie określonego elementu w zbiorze – zadania maturalne

> **Twoje cele**

- Wyjaśnisz, czym jest lider zbioru.
- Przeanalizujesz działanie algorytmu wyszukiwania binarnego.
- Rozwiążesz zadanie związane z wyszukiwaniem danych.

# Przeczytaj

## Implementacja algorytmu znajdowania lidera w C++

Liderem nazywamy liczbę, która występuje w zbiorze o długości `n` więcej niż $\frac{n}{2}$ razy. Ponieważ inna liczba nie może występować w tym samym zbiorze liczbowym więcej niż $\frac{n}{2}$ razy, lider jest tylko jeden. W zbiorze `{1, 4, 6, 1, 1}` lider to zatem liczba 1, ponieważ występuje w nim aż 3 razy.

W implementacji algorytmu znajdowania lidera użyjemy dodatkowej zmiennej `dlugoscZbioru`, która będzie zawierać liczbę elementów tablicy.

```
int zbior[3] = {1, 3, 1};
int dlugoscZbioru = 3;
int lider = zbior[0];
int n = 1;
```

Następnie zaimplementujemy pętlę `for`, wraz ze wszystkimi instrukcjami warunkowymi, które są potrzebne, aby algorytm zadziałał.

```
int zbior[3] = {1, 3, 1};
int dlugoscZbioru = 3;
int lider = zbior[0];
int n = 1;

for (int i = 1; i < dlugoscZbioru; i++) {
    if (n > 0) {
        if (lider == zbior[i]) {
            n = n + 1;
        } else {
            n = n - 1;
        }
    } else {
        lider = zbior[i];
        n = 1;
    }
}
```

Kolejnym krokiem jest sprawdzenie, czy wartość zmiennej `n` wynosi 0. Jeżeli tak, możemy zakończyć działanie programu, ponieważ zbiór na pewno nie ma lidera. Jeżeli nie, wartość znajdująca się w zmiennej `lider` jest potencjalnym liderem.

```
int zbior[3] = {1, 3, 1};
int dlugoscZbioru = 3;
int lider = zbior[0];
int n = 1;

for (int i = 1; i < dlugoscZbioru; i++) {
    if (n > 0) {
        if (lider == zbior[i]) {
            n = n + 1;
        } else {
            n = n - 1;
        }
    } else {
        lider = zbior[i];
        n = 1;
    }
}

if (n == 0) {
    cout << "Zbiór nie ma lidera" << endl;
    return 0;
}
```

Ostatnim krokiem jest sprawdzenie, ile razy potencjalny kandydat wytypowany przez algorytm występuje w zbiorze. Jeżeli liczba ta jest większa niż długość zbioru podzielona przez 2, liczba ta jest liderem tego zbioru.

```
using namespace std;

int zbior[3] = {1, 3, 1};
int dlugoscZbioru = 3;
int lider = zbior[0];
int n = 1;

for (int i = 1; i < dlugoscZbioru; i++) {
    if (n > 0) {
```

```
        if (lider == zbior[i]) {
            n = n + 1;
        } else {
            n = n - 1;
        }
    } else {
        lider = zbior[i];
        n = 1;
    }
}

if (n == 0) {
    cout << "Zbiór nie ma lidera" << endl;
    return 0;
}

int liczbaWystapienLidera = 0;
for (int i = 0; i < dlugoscZbioru; i++) {
    if (zbior[i] == lider) {
    liczbaWystapienLidera += 1;
    }
}

if (liczbaWystapienLidera > dlugoscZbioru / 2) {
    cout << "Liderem zbioru jest " << lider << endl;
} else {
    cout << "Zbiór nie ma lidera" << endl;
}
```

## Słownik

**instrukcja warunkowa**

element algorytmu pozwalający na sprawdzenie jednego lub kilku warunków, a następnie zdefiniowanie, jakie czynności mają być wykonane, jeśli dane warunki są spełnione lub niespełnione (służy do sterowania programem)

**inicjalizacja**

nadanie wartości początkowej

# Symulacja interaktywna

**Problem 1**

Napisz program, który sprawdzi, czy uporządkowany zbiór *n*-elementowy zawiera element o wartości *x*.

**Specyfikacja:**

*Dane:*

*zbiór* – zbiór liczb całkowitych = {2, 4, 6, 8, 10}

*x* – szukana liczba = 4

*Wynik:*

Na standardowym wyjściu wyświetlany jest indeks szukanego elementu lub komunikat o jego braku.

```
#include <iostream>
using namespace std;

int main () {

    // Tutaj dodaj kod. Żeby coś wypisać, użyj polecenia: cout
}
```

```
```

**Polecenie 1**

Przeanalizuj prezentację.

1

Czym jest wyszukiwanie binarne?

Jest to algorytm, który pozwala na znacznie szybsze, w porównaniu do iteracji przez całą tablicę, znajdowanie elementów w zbiorze. Aby wyszukiwanie binarne mogło zadziałać, zbiór elementów musi być posortowany. Tu będzie posortowany rosnąco, ale z kilkoma małymi zmianami. Wyszukiwanie zadziała też z danymi posortowanymi malejąco.

2

Jak działa wyszukiwanie binarne?

Wyszukiwanie binarne używa dwóch indeksów, które oznaczają początek oraz koniec badanego obszaru tablicy. Początkowo obszarem tym jest cała tablica. Gdy algorytm rozpoczyna swoje działanie, znajduje indeks w środku badanego zakresu i sprawdza, czy szukana liczba jest większa czy mniejsza od liczby będącej w miejscu tego indeksu. Jeżeli szukana liczba jest mniejsza, algorytm zmniejsza badany zakres o wszystkie indeksy większe od indeksu środkowego. Jeżeli szukana liczba jest większa, wszystkie indeksy mniejsze niż indeks środkowy zostają usunięte z badanego zakresu. Algorytm kontynuuje zmniejszanie zakresu aż do momentu, w którym w badanym zakresie znajdzie się tylko jedna liczba: to poszukiwana wartość.

Zobaczmy, jak wyszukiwanie binarne wygląda w pseudokodzie:

3

```
zbior[] = tablica liczb o rozmiarze n
lewaGranica = 0
prawaGranica = n - 1

szukanaWartosc = 38

dopóki lewaGranica < prawaGranica wykonuj:
    srodekZakresu = (lewaGranica + prawaGranica) / 2 (dzielenie całkowite)

    jeżeli zbior[srodekZakresu] < szukanaWartosc wykonaj:
        lewaGranica = srodekZakresu + 1
    w przeciwnym wypadku wykonaj:
        prawaGranica = srodekZakresu

jeżeli zbior[lewaGranica] = szukanaWartosc wykonaj:
    wypisz "Szukana wartość jest elementem tablicy o indeksie "
    wypisz lewaGranica
w przeciwnym wypadku wykonaj:
    wypisz "Nie znaleziono szukanej wartości"

```

4

Wspierając się podanym pseudokodem, zacznijmy implementować rozwiązanie

w języku C++. Na początek inicjalizujemy zmienne. Oczywiście będziemy potrzebować posortowanej tablicy i wartości szukanej. Zmienna `lewaGranica` przechowuje indeks elementu, który stanowi lewą granicę obszaru szukania. W miarę zmniejszania się obszaru szukania, wartość ta będzie się zwiększać. Jej odpowiednikiem po prawej stronie jest `prawaGranica` – ta w miarę zmiejszania się obszaru również będzie się zmiejszać. Ponieważ na początku cała tablica jest obszarem wyszukiwania, zmienna `lewaGranica` inicjalizowana jest pierwszym indeksem tablicy, czyli 0, natomiast `prawaGranica` inicjalizowana jest ostatnim indeksem, czyli długością tablicy - 1.

```
1 int zbior[17] = {1, 3, 5,
  8, 9, 10, 12, 15, 18, 20,
  23, 28, 30, 32, 53, 89,
  111};
2 int dlugoscZbioru = 17;
3 int szukanaWartosc = 29;
4 int lewaGranica = 0;
5 int prawaGranica =
  dlugoscZbioru - 1;
```

Dla ułatwienia w implementacji w C++ dodaliśmy zmienną `dlugoscZbioru`, która przechowuje rozmiar tablicy `zbior`.

5

Algorytm ma za zadanie zmniejszać obszar wyszukiwania tak długo, aż jedynym elementem, jaki pozostanie, będzie szukana liczba. Gdy tak się stanie `lewaGranica`, będzie miała taką samą wartość jak `prawaGranica`, wtedy też pętla `while` powinna zakończyć swoje działanie.

```
1 while (lewaGranica <
  prawaGranica) {
```

```
    // Zmiejszanie obszaru wyszukiwania
}
```

6

Zanim zmniejszymy obszar wyszukiwania, musimy wyznaczyć środkowy indeks i sprawdzić, czy liczba, która jest elementem tablicy o danym indeksie, jest większa, czy mniejsza od szukanej wartości. Informacja ta pozwoli nam stwierdzić, która granica powinna zostać przesunięta.

```
while (lewaGranica < prawaGranica) {
    int srodekZakresu = (lewaGranica + prawaGranica) / 2;

    if (zbior[srodekZakresu] < szukanaWartosc) {
        lewaGranica = srodekZakresu + 1;
    } else {
        prawaGranica = srodekZakresu;
    }
}
```

Jeżeli liczba w miejscu środkowego indeksu jest mniejsza niż szukana wartość, oznacza to, że nie może ona być szukaną wartością, dlatego do lewej granicy dodajemy 1.

7

Po zakończeniu działania pętli zmienne `lewaGranica` oraz `prawaGranica` mają tę samą wartość. To indeks szukanej wartości.

Jeżeli jednak szukana wartość nie istnieje w tablicy, będzie to indeks ostatniego elementu przeszukanego przez algorytm. Aby sprawdzić, czy szukana wartość została odnaleziona, użyjemy instrukcji warunkowej.

8

```
if (zbior[lewaGranica] == szukanaWartosc) {
    cout << "Szukana wartość jest elementem tablicy o indeksie " << lewaGranica << endl;
}
```

Jeżeli element tablicy o indeksie `lewaGranica` (lub też `prawaGranica`, ponieważ są to identyczne wartości) równa się szukanej wartości, informujemy użytkownika o znalezieniu szukanej wartości.

9

W przypadku, w którym wartość nie została odnaleziona, informujemy użytkownika, że szukana wartość nie istnieje w danej tablicy.

```
if (zbior[lewaGranica] == szukanaWartosc) {
    cout << "Szukana wartość jest elementem tablicy o indeksie " << lewaGranica << endl;
} else {
    cout << "Nie znaleziono szukanej wartości" << endl;
}
```

10

Cała implementacja w C++ wygląda następująco:

```cpp
#include <iostream>
using namespace std;

int main() {
    int zbior[17] = {1, 3, 5, 8, 9, 10, 12, 15, 18, 20, 23, 28, 30, 32, 53, 89, 111};
    int dlugoscZbioru = 17;
    int szukanaWartosc = 29;
    int lewaGranica = 0;
    int prawaGranica = dlugoscZbioru - 1;

    while (lewaGranica < prawaGranica) {
        int srodekZakresu = (lewaGranica + prawaGranica) / 2;

        if (zbior[srodekZakresu] < szukanaWartosc) {
            lewaGranica = srodekZakresu + 1;
        } else {
            prawaGranica = srodekZakresu;
        }
    }

    if (zbior[lewaGranica] == szukanaWartosc) {
        cout << "Szukana wartość jest elementem tablicy o indeksie " << lewaGranica << endl;
    } else {
```

```
        cout << "Nie znaleziono szukanej wartości" << endl;
    }

    return 0;
}
```



**Polecenie 2**

Zapoznaj się z symulacją interaktywną ilustrującą proces wyszukiwania binarnego. Punkty przedstawione na schemacie reprezentują liczby zawarte w tablicy uporządkowanej rosnąco.

Zasób interaktywny dostępny pod adresem https://zpe.gov.pl/a/D19B8M691

# Sprawdź się

Pokaż ćwiczenia: 

**Ćwiczenie 1**

Napisz program, który wypisze największą liczbę z tablicy `zbior`.

Przetestuj program dla następujących danych:

```
zbior[10] = {5, 78, 64, 56, 12, 49, 98, 63, 23, 76}
rozmiarZbioru = 10
```

**Specyfikacja:**

*Dane:*

- *zbior* – tablica liczb naturalnych
- *rozmiarZbioru* – rozmiar tablicy *zbior*; liczba naturalna

*Wynik:*

Na standardowym wyjściu wyświetlana jest liczba naturalna: największa wartość z tablicy *zbior*.

## Twoje zadania

1. Program wypisuje największą liczbę.

**Ćwiczenie 2**

Napisz program, który wyszuka wartość `szukanaWartosc` w tablicy `zbior` i wypisze indeks tego elementu w tablicy (nie wykorzystując wyszukiwania binarnego).

Przetestuj program dla następujących danych:

```
zbior[20] = {2, 6, 8, 13, 16, 19, 23, 28, 31, 36, 39, 40, 42, 4
rozmiarZbioru = 20
szukanaWartosc = 61
```

**Specyfikacja:**

*Dane:*

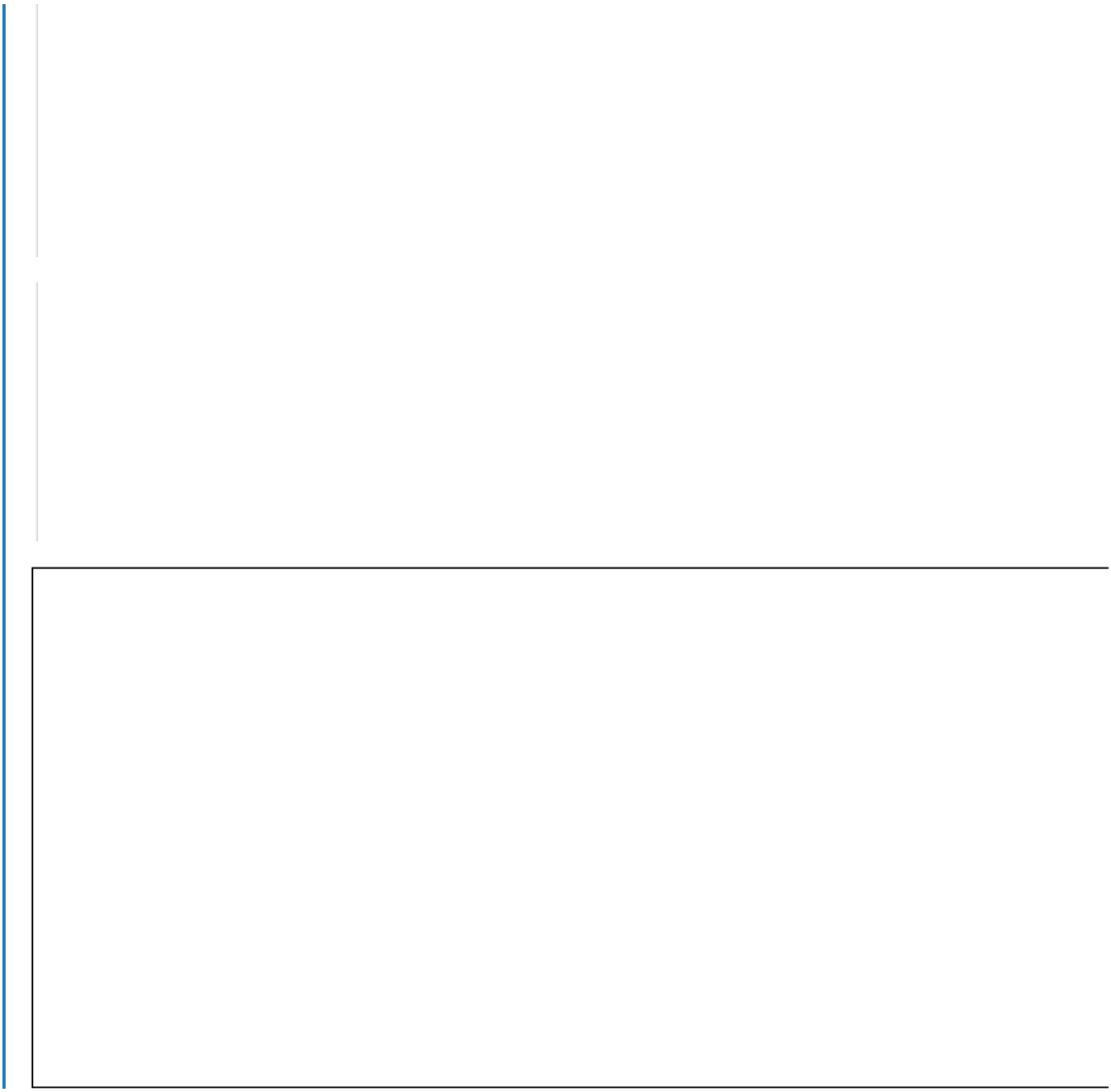
- *zbior* – tablica liczb naturalnych
- *rozmiarZbioru* – rozmiar tablicy *zbior*; liczba naturalna
- *szukanaWartosc* – liczba naturalna

*Wynik:*

Na standardowym wyjściu wyświetlany jest indeks liczby *szukanaWartosc* w tablicy *zbior*.

## Twoje zadania

1. Program znajduje szukaną wartość w tablicy i wypisuje jej indeks (nie wykorzystując wyszukiwania binarnego).



**Ćwiczenie 3**

Napisz program, który używając wyszukiwania binarnego, znajdzie `szukanaWartosc` w tablicy `zbior` i wypisze indeks tego elementu. Następnie porównaj rozwiązanie z rozwiązaniem zadania 2.

Przetestuj program dla następujących danych:

```
zbior[10] = {2, 6, 8, 13, 16, 19, 23, 28, 31, 36, 39, 40, 42, 4
rozmiarZbioru = 20
szukanaWartosc = 61
```

**Specyfikacja:**

*Dane:*

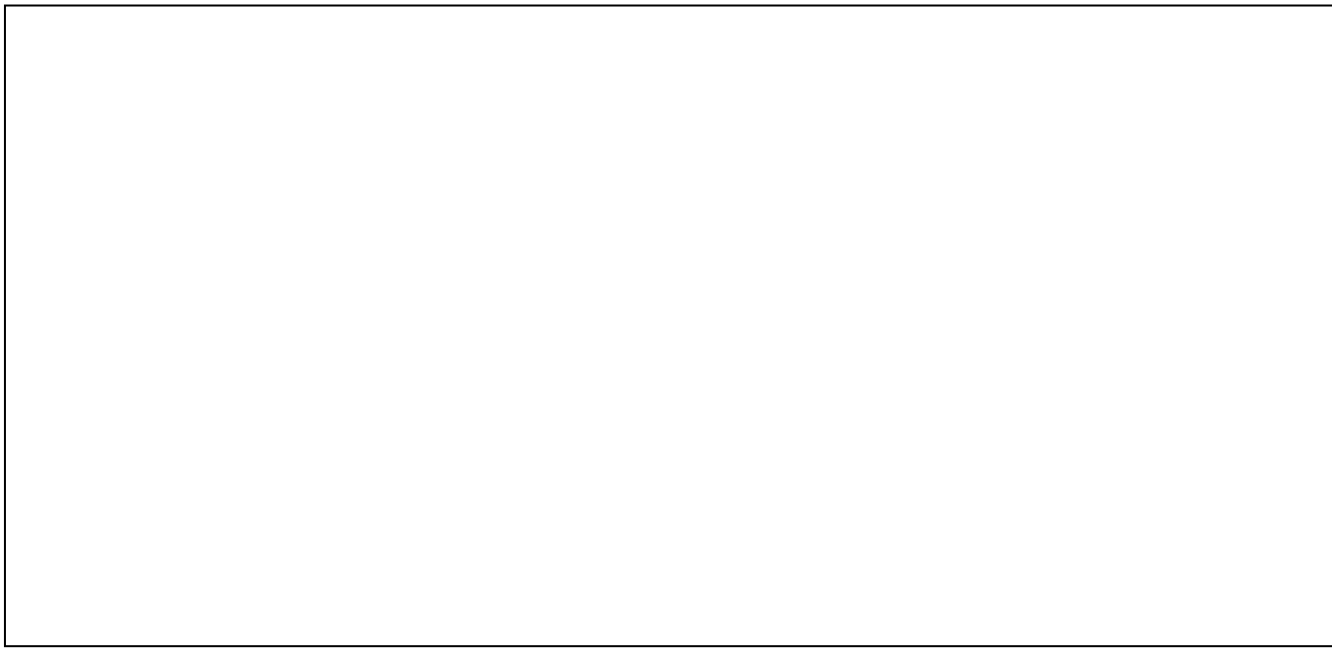
- *zbior* – tablica liczb naturalnych
- *rozmiarZbioru* – rozmiar tablicy *zbior*; liczba naturalna
- *szukanaWartosc* – liczba naturalna

*Wynik:*

Na standardowym wyjściu wyświetlany jest indeks liczby *szukanaWartosc* w tablicy *zbior* lub napis „Nie znaleziono szukanej wartości” w przeciwnym wypadku

## Twoje zadania

1. Program wyszukuje wartość w tablicy i wypisuje jej indeks.



# Dla nauczyciela

**Autor:** Maurycy Gast
**Przedmiot:** Informatyka

**Temat: Znajdowanie określonego elementu w zbiorze w języku C++**

**Grupa docelowa:**

Liceum ogólnokształcące i technikum, liceum ogólnokształcące, technikum, zakres rozszerzony

**Podstawa programowa:**

Cele kształcenia – wymagania ogólne

I. Rozumienie, analizowanie i rozwiązywanie problemów na bazie logicznego i abstrakcyjnego myślenia, myślenia algorytmicznego i sposobów reprezentowania informacji.

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera oraz innych urządzeń cyfrowych: układanie i programowanie algorytmów, organizowanie, wyszukiwanie i udostępnianie informacji, posługiwanie się aplikacjami komputerowymi.

III. Posługiwanie się komputerem, urządzeniami cyfrowymi i sieciami komputerowymi, w tym: znajomość zasad działania urządzeń cyfrowych i sieci komputerowych oraz wykonywania obliczeń i programów.

Treści nauczania – wymagania szczegółowe

I + II. Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

1) zapisuje za pomocą listy kroków, schematu blokowego lub pseudokodu, i implementuje w wybranym języku programowania, algorytmy poznane na wcześniejszych etapach oraz algorytmy:

b) znajdowania określonego elementu w zbiorze: lidera, idola, elementu w zbiorze uporządkowanym metodą binarnego wyszukiwania,

**Kształtowane kompetencje kluczowe:**

- kompetencje cyfrowe;
- kompetencje osobiste, społeczne i w zakresie umiejętności uczenia się;

- kompetencje matematyczne oraz kompetencje w zakresie nauk przyrodniczych, technologii i inżynierii.

**Cele operacyjne (językiem ucznia):**

- Wyjaśnisz, czym jest lider zbioru.
- Przeanalizujesz działanie algorytmu wyszukiwania binarnego.
- Rozwiążesz zadanie związane z wyszukiwaniem danych.

**Strategie nauczania:**

- konstruktywizm;
- konektywizm.

**Metody i techniki nauczania:**

- dyskusja;
- rozmowa nauczająca z wykorzystaniem multimedium i ćwiczeń interaktywnych;
- ćwiczenia praktyczne.

**Formy pracy:**

- praca indywidualna;
- praca w parach;
- praca w grupach;
- praca całego zespołu klasowego.

**Środki dydaktyczne:**

- komputery z głośnikami, słuchawkami i dostępem do internetu;
- zasoby multimedialne zawarte w e-materiale;
- tablica interaktywna/tablica, pisak/kreda;
- oprogramowanie dla języka C++, w tym kompilator GCC/G++ 4.5 (lub nowszej wersji) i Code::Blocks 16.01 (lub nowszej wersji), Orwell Dev-C++ 5.11 (lub nowszej wersji) lub Microsoft Visual Studio.

**Przebieg lekcji**

**Przed lekcją:**

1. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Znajdowanie określonego elementu w zbiorze w języku C++". Uczniowie zapoznają się z treściami w sekcji „Przeczytaj" w kontekście programowania.

**Faza wstępna:**

1. Nauczyciel wyświetla i odczytuje temat lekcji oraz cele zajęć. Prosi uczniów o sformułowanie kryteriów sukcesu.
2. **Rozpoznanie wiedzy uczniów.** Nauczyciel prosi wybranego ucznia lub uczniów o przedstawienie sytuacji problemowej związanej z tematem lekcji.

**Faza realizacyjna:**

1. **Praca z tekstem.** Uczniowie przystępują do cichego czytania tekstu e-materiału. Indywidualnie zapoznają się z treścią w sekcji „Przeczytaj".
2. **Praca z multimedium.** Nauczyciel wyświetla zawartość sekcji „Film samouczek". Uczniowie wspólnie analizują prezentację, a następnie próbują zaimplementować wyszukiwanie binarne w języku C++.
3. **Ćwiczenie umiejętności.** Uczniowie wykonują indywidualnie ćwiczenie nr 1, a następnie porównują swoje odpowiedzi z kolegą lub koleżanką.
4. Uczniowie dobierają się w pary i wykonują ćwiczenia nr 2 z sekcji „Sprawdź się". Następnie konsultują swoje rozwiązania z inną parą uczniów.

**Faza podsumowująca:**

1. Nauczyciel wyświetla na tablicy temat lekcji i cele zawarte w sekcji „Wprowadzenie". W kontekście ich realizacji podsumowuje przebieg zajęć, a także wskazuje mocne i słabe strony pracy uczniów.

**Praca domowa:**

1. Uczniowie zapoznają się z symulacją interaktywną z sekcji „Symulacja interaktywna", która ilustruje proces wyszukiwania binarnego. Punkty przedstawione na schemacie reprezentują liczby zawarte w tablicy uporządkowanej rosnąco.
2. Uczniowie wykonują ćwiczenie 3 z sekcji „Sprawdź się".

**Materiały pomocnicze:**

- Oficjalna dokumentacja techniczna dla języka C++.
- Oficjalna dokumentacja techniczna dla kompilatora GCC/G++ 4.5 (lub nowszej wersji).
- Oficjalna dokumentacja techniczna dla oprogramowania Code::Blocks 16.01 (lub nowszej wersji), Orwell Dev-C++ 5.11 (lub nowszej wersji) lub Microsoft Visual Studio.

**Wskazówki metodyczne:**

- Treści w sekcji „Przeczytaj" można wykorzystać jako podsumowanie i utrwalenie wiedzy uczniów.