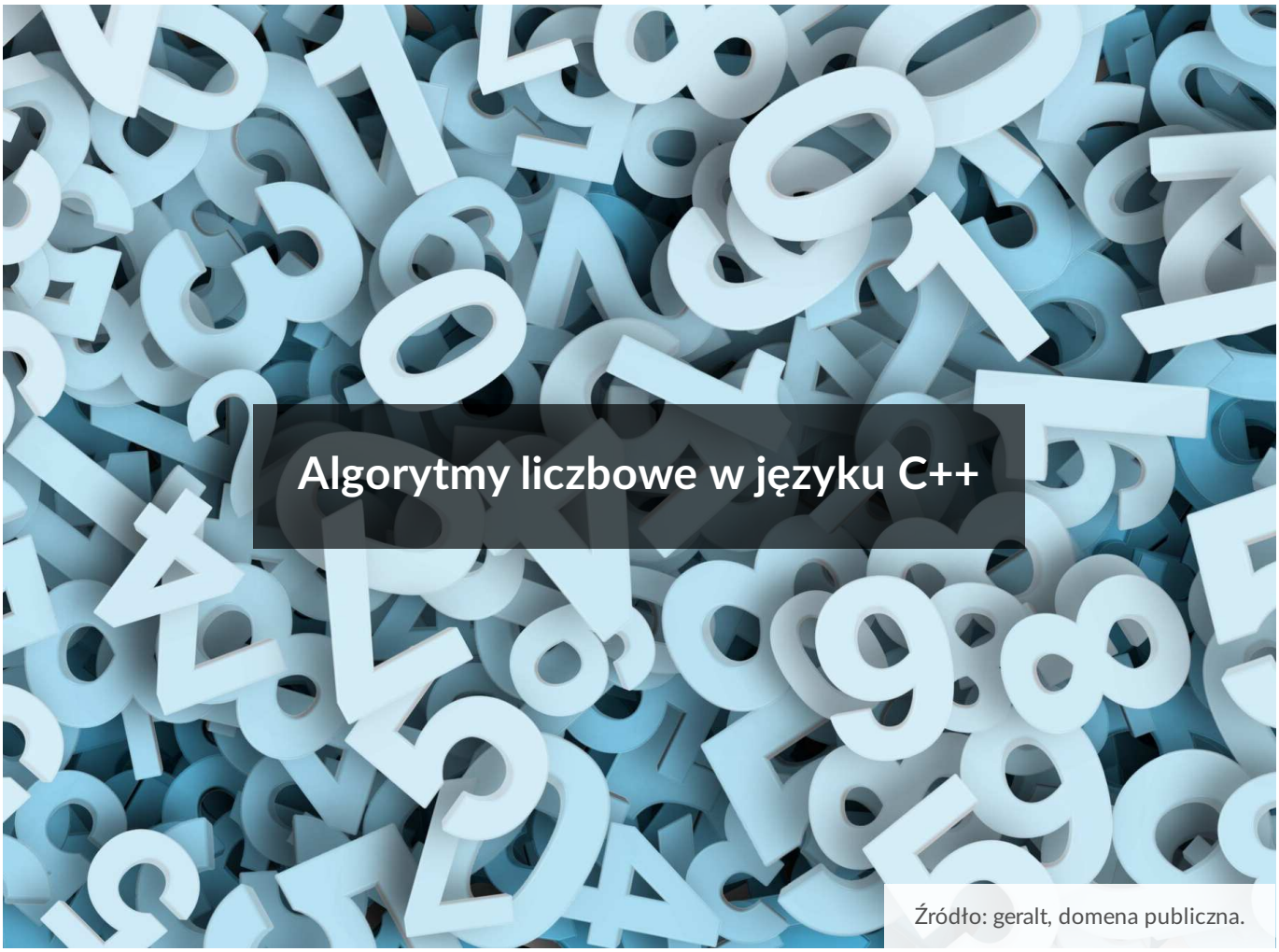
# Algorytmy liczbowe w języku C++




Źródło: geralt, domena publiczna.

Liczby zaprzyjaźnione, bliźniacze, doskonałe oraz pierwsze to liczby spełniające określone warunki. To, czy dana liczba należy do wybranego zbioru, weryfikujemy, korzystając z odpowiedniego algorytmu liczbowego. W e-materiale Algorytmy liczbowe poznaliśmy podstawowe informacje na ich temat.

W tym e-materiale poznamy implementacje wybranych algorytmów liczbowych w języku C++.

Implementacje w innych językach programowania znajdziesz w e-materiałach:

- Algorytmy liczbowe w języku Java,
- Algorytmy liczbowe w języku Python.

Więcej zadań? Przejdź do Algorytmy liczbowe – zadania maturalne.

**Twoje cele**

- Zastosujesz w praktyce wiedzę dotyczącą algorytmów liczbowych.
- Prześledzisz przykładowe algorytmy liczbowe zapisane w języku C++.
- Przeanalizujesz, napisaną w języku C++, implementację algorytmu służącego do sprawdzania, czy liczby są pierwsze, zaprzyjaźnione, doskonałe czy bliźniacze.
- Wykonasz kilka ćwiczeń związanych z algorytmami liczbowymi.

# Przeczytaj

## Liczby pierwsze

Zgodnie z definicją liczba pierwsza to taka, która spełnia następujący warunek:

- ma tylko dwa dzielniki: jeden i samą siebie.

Zaimplementujmy w języku C++ algorytm pozwalający sprawdzić, czy podana liczba jest liczbą pierwszą.

Zacznijmy od utworzenia dwóch zmiennych: jednej typu całkowitego (`int`) o nazwie `liczbaTest` oraz drugiej typu `bool`, która będzie się nazywać `czyPierwsza`. Do `liczbaTest` zostanie zapisana liczba, którą użytkownik wpisał na klawiaturze, w celu sprawdzenia, czy jest ona liczbą pierwszą. Natomiast zmienna `czyPierwsza` będzie przyjmować wartość `true` lub `false` w zależności od tego, czy mamy do czynienia z liczbą pierwszą, czy też nie.

Ponieważ wspomnieliśmy o interakcji programu z użytkownikiem, napiszmy kod odpowiedzialny za możliwość wprowadzenia przez użytkownika własnych danych, czyli w tym przypadku liczby, która ma zostać poddana sprawdzeniu.

```
int liczbaTest;
bool czyPierwsza;

std::cin >> liczbaTest;
```

Następnym krokiem będzie napisanie pętli, w której będziemy sprawdzać liczby od 2 do liczby mniejszej o jeden od podanej. Ponieważ dzielnikami liczby pierwszej jest 1 i podana liczba, więc w przedziale $<2,\ liczbaTest\ -\ 1>$ nie powinno być żadnego dzielnika podanej liczby. W przypadku wykrycia dzielnika całkowitego pętla zostanie przerwana instrukcją `break`, a zmienna logiczna `czyPierwsza` przyjmie wartość `false`, co będzie oznaczało, że podana przez użytkownika liczba nie jest liczbą pierwszą.

```
for(int i = 2; i < liczbaTest; i++) {
    if(liczbaTest % i == 0) {
        czyPierwsza = false;
        break;
    }
}
```

**Ważne!**

Możemy zoptymalizować nasz algorytm i sprawdzanie, czy liczba jest dzielnikiem `liczbaTest` wykonywać w przedziale $<2, \sqrt{liczbaTest}>$. Pętlę `for` można wtedy przekształcić następująco: `for(int i = 2; i*i  <= liczbaTest; i++)`. Zapis `i*i <= liczbaTest` wynika z podniesienia dwóch stron do kwadratu, aby pozbyć się pierwiastka. Dlaczego sprawdzanie wystarczy dokonać tylko do pierwiastka liczby? Przeanalizujmy przykłady liczb 18 oraz 16 i wypiszmy ich dzielniki:

$$\sqrt{18} \approx 4,24$$

Dzielniki 18 = {1, 2, 3, **|4.24|**, 6, 9, 18}

$$\sqrt{16} = 4$$

Dzielniki 16 = {1, 2, **|4|**, 8, 16}

Pomiędzy znakami **||** zapisano wartość bezwzględną pierwiastka liczby $n$.

Można zauważyć, że jeśli liczba $n$ ma dzielniki większe od 1, to posiada ich tyle samo po obu stronach od $\sqrt{n}$. W przypadku liczb kwadratowych np. 16, 25, 36 itp. pierwiastek liczby będzie jej dzielnikiem.

```
for(int i = 2; i*i <= liczbaTest; i++) {
    if(liczbaTest % i == 0) {
        czyPierwsza = false;
        break;
    }
}
```

Na koniec określamy, jaki komunikat ma się pojawić na standardowym wyjściu, w zależności od tego, jaka jest wartość zmiennej `czyPierwsza`.

```
if(czyPierwsza == true) {
    std::cout << "Podana liczba jest liczbą pierwszą" << std::end
} else {
    std::cout << "Podana liczba nie jest liczbą pierwszą" << std:
}
```

**Ciekawostka**

> Instrukcję warunkową `if(czyPierwsza == true)` można zapisać również jako `if(czyPierwsza)`.

Zobaczmy kod całego programu.

```
#include <iostream>

using namespace std;

int main() {
    int liczbaTest;
    bool czyPierwsza = true;

    cout << "Podaj liczbę, którą chcesz poddać sprawdzeniu:" << e
    cin >> liczbaTest;

    for(int i = 2; i < liczbaTest; i++) {
        if(liczbaTest % i == 0) {
            czyPierwsza = false;
            break;
        }
    }

    if(czyPierwsza == true) {
        cout << "Podana liczba jest liczbą pierwszą" << std;
    } else {
        cout << "Podana liczba nie jest liczbą pierwszą" << std;
    }

    return 0;
}
```

Warto zauważyć, że w tej implementacji algorytmu wychodzimy z założenia, że podana liczba jest liczbą pierwszą, ponieważ przy deklaracji zmiennej logicznej `czyPierwsza` nadajemy jej wartość `true`. Po wykonaniu pętli `for`, jeżeli wartość `czyPierwsza` nie zostanie zmieniona, jej wartość nadal będzie wynosić `true`, a my uzyskamy pewność, że mamy do czynienia z liczbą pierwszą.

## Liczby doskonałe

Zacznijmy od przypomnienia warunku, jaki musi spełniać liczba doskonała:

- Suma dzielników mniejszych od sprawdzanej liczby musi być równa tej liczbie.

Przykładami liczb doskonałych są:

- 6, ponieważ:

$$1 + 2 + 3 = 6$$

gdzie 1, 2 i 3 to dzielniki właściwe liczby 6.

- 28, ponieważ:

$$1 + 2 + 3 + 4 + 7 + 14 = 28$$

gdzie 1, 2, 3, 4, 7, 14 to dzielniki właściwe liczby 28.

Można zauważyć, że nasz algorytm będzie polegał na znalezieniu dzielników podanej liczby, zsumowaniu ich i sprawdzeniu, czy wynik jest równy sprawdzanej liczbie. Przeanalizujmy kod programu, który stanowi rozwiązanie tego problemu:

```
#include <iostream>

using namespace std;

int main() {
    int liczba;
    int sumaDzielnikow = 0;

    cout << "Podaj liczbę do sprawdzenia:" << std;
    cin >> liczba;

    for(int i = 1; i < liczba; i++) {
        if(liczba % i == 0) {
            sumaDzielnikow = sumaDzielnikow + i;
        }
    }

    if(sumaDzielnikow == liczba) {
        cout << "Dana liczba jest liczbą doskonałą";
    } else {
        cout << "Dana liczba nie jest liczbą doskonałą";
    }
```

```
}
```

W pętli `for` sprawdzamy, czy kolejne liczby (zaczynając od 1) są dzielnikami zmiennej `liczbaTest`. Jeżeli znajdziemy taki dzielnik, dodajemy go do zmiennej `sumaDzielnikow`. Następnie porównujemy wartość zmiennej `sumaDzielnikow` z podaną do sprawdzenia liczbą. Jeżeli są one równe, to jest to liczba doskonała.

## Liczba doskonała – optymalizacja algorytmu

Zastanówmy się, czy można zoptymalizować przedstawiony wyżej algorytm. Pierwszą rzeczą, którą możemy zrobić, jest rozpoczęcie przetwarzanie pętli od liczby 2, ponieważ każda liczba jest podzielna przez 1. Unikniemy dzięki temu jednej, niepotrzebnej iteracji.

Możemy także dodać warunek wewnątrz pętli. Pozwoli to sprawdzić, czy aktualna wartość zmiennej `sumaDzielnikow` nie jest większa od badanej liczby. W ten sposób uda się uniknąć wykonywania wielu niepotrzebnych cykli pętli.

Niżej przedstawiamy kod programu po wprowadzeniu zmian optymalizacyjnych.

```
#include <iostream>

using namespace std;

int main() {
    int liczba;
    int sumaDzielnikow = 1;

    cout << "Podaj liczbę do sprawdzenia:" << std;
    cin >> liczba;

    for(int i = 2; i < liczba; i++) {
        if(liczba % i == 0) {
            sumaDzielnikow = sumaDzielnikow + i;

            if(sumaDzielnikow > liczba) {
                break;
            }
        }
    }

    if(sumaDzielnikow == liczba) {
```

```
        cout << "Dana liczba jest liczba doskonałą";
    } else {
        cout << "Dana liczba nie jest liczba doskonałą";
    }
}
```

## Liczby zaprzyjaźnione

Liczby zaprzyjaźnione to takie, których suma dzielników właściwych pierwszej liczby jest równa wartości drugiej liczby, a suma dzielników właściwych drugiej liczby jest równa wartości pierwszej liczby.

Przykładem pary takich liczb są: 220 i 284.

Dzielniki właściwe liczby 220 to: $1 + 2 + 4 + 5 + 10 + 11 + 20 + + 22 + 44 + 55 + 110 = 284$. Natomiast dzielniki właściwe liczby 284 to: $1 + 2 + 4 + 71 + 142 = 220$.

Przeanalizujmy implementację algorytmu, która pozwoli sprawdzić, czy podane przez użytkownika dwie liczby są zaprzyjaźnione.

```
#include <iostream>

using namespace std;

int main() {
    int liczba1, liczba2;
    int l1Suma = 0;
    int l2Suma = 0;

    cout << "Podaj pierwszą liczbę do sprawdzenia:" << std;
    cin >> liczba1;
    cout << "Podaj drugą liczbę do sprawdzenia:" << std;
    cin >> liczba2;

    for(int i = 1; i < liczba1; i++) {
        if(liczba1 % i == 0) {
            l1Suma += i;
        }
    }

    for(int i = 1; i < liczba2; i++) {
```

```
22          if(liczba2 % i == 0) {
23              l2Suma += i;
24          }
25      }
26
27      if(liczba1 == l2Suma && liczba2 == l1Suma) {
28          cout << "To są liczby zaprzyjaźnione";
29      } else {
30          cout << "To nie są liczby zaprzyjaźnione";
31      }
32  }
```

Na początku programu deklarujemy zmienne `liczba1` i `liczba2`. To w nich zapiszemy liczby, które użytkownik podał do sprawdzenia. Kolejne zmienne to `l1Suma` oraz `l2Suma`. Będą one odpowiadały kolejno sumie dzielników właściwych liczby pierwszej i sumie dzielników właściwych liczby drugiej.

Następnie w pętlach `for` sprawdzamy kolejne dzielniki liczb i sumujemy je. W końcowej części programu porównujemy wartość pierwszej liczby z sumą dzielników liczby drugiej i wartość liczby drugiej z sumą dzielników liczby pierwszej, a następnie wypisujemy odpowiedni komunikat.

## Słownik

> **dzielniki właściwe**
>
> dzielniki liczby mniejsze od niej samej

# Gra edukacyjna

**Problem 1**
Napisz program, który wyszuka liczby bliźniacze z podanego zakresu. Przetestuj jego działanie dla zmiennych a i b o wartościach kolejno 2 i 2000.

**Specyfikacja:**

*Dane:*

- a, b – zmienna typu *int*

*Wynik:*

Na standardowym wyjściu wyświetlany jest komunikat „Znalezione pary” oraz, w osobnych liniach, znalezione liczby bliźniacze.

**Polecenie 1**
Przeanalizuj prezentację i porównaj z nią swoje rozwiązanie.



**Polecenie 2**
Sprawdź swoją wiedzę, biorąc udział w grze.

Test

## Sprawdź swoją wiedzę o algorytmach liczbowych w języku C++, biorąc udział w grze

Poziom trudności:

InteractiveTest.difficultyLevel.easy

Limit czasu:

7 min

Twój ostatni wynik:

-

Trwa wczytywanie...

**Ćwiczenie 1**

Zapoznaj się z kodem i odpowiedz na pytanie.

```
bool main(int liczbaTest) {
    int liczba1 = 0;

    for (int i = 1; i < liczbaTest; i++) {
        if (liczbaTest % i == 0) {
            liczba1 = liczba1 + i;
        }
    }

    return liczba1 == liczbaTest;
}
```

### Ćwiczenie 2

Zapoznaj się z kodem i odpowiedz na pytanie.

```
int main(int liczba1, int liczba2) {
    int l1_suma = 0;
    int l2_suma = 0;

    for (int i = 1; i < liczba1; i++) {
        if (liczba1 % i == 0) {
            l1_suma += i;
        }
    }

    for (int i = 1; i < liczba2; i++) {
        if (liczba2 % i == 0) {
            l2_suma += i;
        }
    }

    return liczba1 == l2_suma && liczba2 == l1_suma;
}
```

### Ćwiczenie 3

Zapoznaj się z kodem i odpowiedz na pytanie.

```
int main(int liczbaTest) {
    bool czy = 1;

    for (int i = 2; i < liczbaTest; i++) {
        if (liczbaTest % i == 0) {
            czy = 0;
            break;
        }
    }

    return czy;
}
```

# Sprawdź się

Pokaż ćwiczenia:

**Ćwiczenie 1**

Program powinien sprawdzać, czy podana liczba jest liczbą doskonałą. Warunek wewnątrz pętli jest niepełny. Dopisz kod i przetestuj działanie programu dla liczby 6.

**Specyfikacja:**

*Dane:*

- `liczba`, `sumaDzielnikow` – zmienne typu *int*

*Wynik:*

W konsoli wyświetla się *Dana liczba jest liczbą doskonałą*, jeżeli liczba spełnia kryteria liczby doskonałej. W przeciwnym razie wyświetla komunikat: *Dana liczba nie jest liczbą doskonałą*.

**Ćwiczenie 2**

Program powinien sprawdzać, czy podana liczba jest liczbą pierwszą. Kod programu nie jest pełny. Dopisz brakujące instrukcje i przetestuj działanie programu dla liczby 19.

**Specyfikacja:**

*Dane:*

- `liczba` – zmienna typu *int*
- `czyPierwsza` – zmienna typu *bool*

*Wynik:*

W konsoli wyświetla się *Dana liczba jest liczbą pierwszą*, jeżeli liczba spełnia kryteria liczby pierwszej. W przeciwnym razie wyświetla komunikat: *Dana liczba nie jest liczba pierwszą*.

**Ćwiczenie 3**

Napisz program wyszukujący liczby bliźniacze z podanego przez użytkownika zakresu. Program powinien wypisywać tylko te pary liczb bliźniaczych, w których jedna lub dwie z liczb kończą się cyfrą 3. Przetestuj jego działanie dla zakresu <2, 100>.

**Specyfikacja:**

*Dane:*

- `i` – zmienna typu *int*; dolny zakres wyszukiwania liczb bliźniaczych
- `j` – zmienna typu *int*; górny zakres wyszukiwania liczb bliźniaczych

*Wynik:*

Na standardowym wyjściu wyświetlane są liczby bliźniacze w następujący sposób: każda para liczb w nowej linii, liczby w parze powinny być oddzielone literą „v”.

# Dla nauczyciela

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

**Temat: Algorytmy liczbowe w języku C++**

**Grupa docelowa:**

Liceum ogólnokształcące i technikum, liceum ogólnokształcące, technikum, zakres podstawowy i 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.

Treści nauczania – wymagania szczegółowe

I. Rozumienie, analizowanie i rozwiązywanie problemów.

Zakres podstawowy. Uczeń:

1) planuje kolejne kroki rozwiązywania problemu, z uwzględnieniem podstawowych etapów myślenia komputacyjnego (określenie problemu, definicja modeli i pojęć, znalezienie rozwiązania, zaprogramowanie i testowanie rozwiązania).

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera i innych urządzeń cyfrowych.

Zakres podstawowy. Uczeń:

1) projektuje i programuje rozwiązania problemów z różnych dziedzin, stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje warunkowe, instrukcje iteracyjne, funkcje z parametrami i bez parametrów, testuje poprawność programów dla różnych danych; w szczególności programuje algorytmy z punktu I.2);

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

3) sprawnie posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

**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):**

- Zastosujesz w praktyce wiedzę dotyczącą algorytmów liczbowych.
- Prześledzisz przykładowe algorytmy liczbowe zapisane w języku C++.
- Przeanalizujesz, napisaną w języku C++, implementację algorytmu służącego do sprawdzania, czy liczby są pierwsze, zaprzyjaźnione, doskonałe czy bliźniacze.
- Wykonasz kilka ćwiczeń związanych z algorytmami liczbowymi.

**Strategie nauczania:**

- konstruktywizm;
- konektywizm.

**Metody i techniki nauczania:**

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

**Formy pracy:**

- praca indywidualna;
- praca w parach;
- praca w grupach.

**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: „Algorytmy liczbowe w języku C++". Uczniowie zapoznają się z treściami w sekcji „Przeczytaj".

**Faza wstępna:**

1. Ustalenie celu lekcji i kryteriów sukcesu.

**Faza realizacyjna:**

1. **Praca z tekstem.** Jeżeli przygotowanie uczniów do lekcji jest niewystarczające, nauczyciel prosi o indywidualne zapoznanie się z treścią zawartą w sekcji „Przeczytaj". Każdy uczestnik zajęć podczas cichego czytania wynotowuje najważniejsze kwestie poruszane w tekście.
2. **Praca z multimedium.** Uczniowie zapoznają się z problemem 1 z sekcji „Gra edukacyjne". W parach opracowują jego rozwiązanie. Następnie porównują je z przedstawionym w prezentacji.
3. **Ćwiczenie umiejętności.** Prowadzący zapowiada uczniom, że w kolejnym kroku będą rozwiązywać ćwiczenia nr 1-2 z sekcji „Sprawdź się". Każdy z uczniów robi to samodzielnie. Po ustalonym czasie wybrani uczniowie przedstawiają rozwiązania. Nauczyciel w razie potrzeby koryguje odpowiedzi, dopowiada istotne informacje, udziela uczniom informacji zwrotnej.
4. Liga zadaniowa – uczniowie pracując w parach, wykonują ćwiczenie nr 3 z sekcji „Sprawdź się", a następnie dzielą się swoimi wynikami przez porównywanie napisanego kodu z inną grupą, która również zakończyła zadanie.

**Faza podsumowująca:**

1. Na koniec zajęć nauczyciel raz jeszcze wyświetla na tablicy temat lekcji i cele zawarte w sekcji „Wprowadzenie". W odniesieniu do ich realizacji dokonuje szczegółowej oceny rozwiązania zastosowanego przez wybranego ucznia.

**Praca domowa:**

1. Uczniowie biorą udział w grze z sekcji „Gra edukacyjna".

**Materiały pomocnicze:**

- Oficjalna dokumentacja techniczna dla języka C++.
- Oficjalna dokumentacja techniczna dla kompilatora GCC/G++ 4.5 (lub nowszej wersji).

**Wskazówki metodyczne:**

- Nauczyciel może wykorzystać multimedium w sekcji „Przeczytaj” do pracy przed lekcją. Uczniowie zapoznają się z jego treścią i przygotowują do pracy na zajęciach w ten sposób, żeby móc samodzielnie rozwiązać zadania dołączone do e-materiału „Algorytmy liczbowe w języku C++”.