Szyfr RSA w języku C++

- Wprowadzenie
- Film samouczek
- Przeczytaj
- Sprawdź się
- Dla nauczyciela

Szyfr RSA w języku C++

Źródło: Skitterphoto, dostępny w internecie: pixabay.com, domena publiczna.

W świecie, w którym dominującą formą komunikacji jest przekaz cyfrowy, niezbędne stało się opracowywanie algorytmów szyfrujących. Jedną z często używanych przez kryptografów metod jest szyfr RSA. W tym e-materiale zajmiemy się jego implementacją w języku C++.

Ciekawi cię, jak wyglądają implementacje w innych językach programowania? Możesz się z nimi zapoznać w dwóch pozostałych e-materiałach z tej serii:

- Szyfr RSA w języku Java,
- Szyfr RSA w języku Python.

Więcej zadań? Sięgnij do: Szyfr RSA – zadania maturalne.

Twoje cele

- Zaimplementujesz w języku C++ klucze prywatny i publiczny szyfru RSA.
- Napiszesz program, który za pomocą algorytmu RSA wygeneruje podpis i zweryfikuje jego poprawność.
- Wykonasz ćwiczenia dotyczące szyfrowania i deszyfrowania wiadomości za pomocą szyfru RSA.

Film samouczek

Problem 1
Napisz program, który za pomocą algorytmu RSA wygeneruje dla danej wiadomości podpis i zweryfikuje jego poprawność. Algorytm powinien działać w oparciu o podane wartości liczb pierwszych p i q.

Program przetestuj dla wiadomości o treści „Ola ma kota” i wartości liczb pierwszych $p = 13$ i $q = 11$.

Specyfikacja problemu:

Dane:

- `wiadomosc` – łańcuch znaków; podana wiadomość do zaszyfrowania
- p – liczba całkowita dodatnia; jedna z liczb pierwszych potrzebnych do wyznaczenia kluczy
- q – liczba całkowita dodatnia; druga z liczb pierwszych potrzebnych do wyznaczenia kluczy

Wynik:

- szyfrogram podanej wiadomości i jego odszyfrowana postać jako dowód odwracalności szyfrowania

Polecenie 1
Porównaj swoje rozwiązanie z zaprezentowanym w filmie.

Wystąpił błąd

Film dostępny pod adresem /preview/resource/RuZAuuhNIMmWE

Film nawiązujący do treści materiału: Szyfr RSA - Podpis elektroniczny i jego zastosowanie.

Plik o rozmiarze 2.27 KB w języku polskim

Przeczytaj

Klucze RSA w języku C++

Specyfikacja problemu:

Dane:

- `p` – liczba naturalna pierwsza; jedna z liczb pierwszych potrzebnych do wyznaczenia kluczy
- `q` – liczba naturalna pierwsza; druga z liczb pierwszych potrzebnych do wyznaczenia kluczy

Wynik:

- `klucz_prywatny` – para liczb naturalnych; klucz prywatny algorytmu RSA
- `klucz_publiczny` – para liczb naturalnych; klucz publiczny algorytmu RSA

> **Przykład 1**
>
> Wygenerujmy w języku C++ klucze prywatny i publiczny szyfru RSA.

Algorytm szyfrowania RSA wymaga zaimplementowaniu kilku funkcji pomocniczych lub zastosowania odpowiednich bibliotek. Jego tworzenie rozpoczniemy od wybrania dwóch liczb pierwszych i oznaczenia ich jako `p` i `q`. Do ich odnalezienia możemy zastosować sito Eratostenesa (jego implementacja w języku C++ została omówiona w e-materiale: Sito Eratostenesa w języku C++).

Następnie obliczamy wartość `n = p · q` oraz wartość funkcji Eulera dla `n` (informacje o niej znajdziesz w e-materiale Szyrf RSA).

W kolejnym kroku należy odnaleźć liczbę `e` spełniającą warunek `(1 < e < φ(n))`, która będzie względnie pierwsza do liczby `φ`. W tym celu można posłużyć się podstawową wersją algorytmu Euklidesa (jego implementacja w języku C++ została omówiona w e-materiale: Algorytm Euklidesa w języku C++).

W następnej kolejności należy obliczyć liczbę `d` spełniającą równanie:

$$d \cdot e \ mod \ \varphi(n) = 1, \ lub \ inaczej \ d = e^{-1} \ mod \ \varphi(n)$$

Aby to osiągnąć, można posłużyć się rozszerzonym algorytmem Euklidesa, również przedstawionym we wspomnianym wyżej materiale.

Kluczem publicznym jest para liczb `(e, n)`, a kluczem prywatnym `(d, n)`, gdzie:

- e – liczba względnie pierwsza do liczby φ,
- n – p · q, gdzie p i q to liczby pierwsze i p != q,
- d – odwrotność modulo φ(n) liczby e, gdzie φ(n) to funkcja Eulera.

Przedstawiona poniżej funkcja pozwala na wyznaczenie największego wspólnego dzielnika lub sprawdzenie, czy podane liczby są względnie pierwsze.

Funkcja przyjmuje cztery argumenty: liczby, dla których szukamy największego wspólnego dzielnika, czyli a i b oraz wskaźniki na liczby, które będą miały spełniać tzw. tożsamość Bezouta, czyli x i y. Liczby a, b, x i y są liczbami całkowitymi. Tożsamość Bezouta wygląda następująco: $ax + by = \mathrm{NWD}(a,b)$

Przedstawiana funkcja będzie opierała się na mechanizmie rekurencji, w związku z czym wymaga ona warunku zakończenia, po którego spełnieniu zwróci wartość, co pozwoli na zakończenie rekurencji (zapobiegnie powstaniu nieskończonej pętli). Dla rozszerzonego algorytmu Euklidesa warunkiem tym jest osiągnięcie przez pierwszą z porównywanych liczb wartości 0. Wtedy jedynym liczącym się współczynnikiem w tożsamości Bezouta będzie współczynnik by, który będzie równy wartości największego wspólnego dzielnika liczb a i b, więc wartość x ustawiamy na 0, a y na 1 (zerujemy pierwszy wyraz równania, a drugi przyjmuje wartość b).

Następnym krokiem jest stworzenie zmiennych x1 i y1, które będą wykorzystane do wyznaczenia końcowych wartości x i y po zakończeniu rekurencji, w każdym wywołaniu funkcji.

Kolejnym etapem tego algorytmu będzie rekurencyjne wywołanie funkcji ze zmienionymi argumentami. Pierwszy argument powinien przyjąć wartość b%a, a jako drugi podajemy dotychczasowy argument a. W miejsce x i y podajemy wskaźniki na x1 i y1. W ten sposób uzyskamy cykl rekurencji zakończony w momencie wystąpienia warunku zakończenia – zredukowania pierwotnych wartości porównywanych do jednej różnej od 0, będącej ich największym wspólnym dzielnikiem.

Ostatni etap to modyfikacja wartości x i y, tak by spełniały one tożsamość Bezouta, i zwrócenie wyznaczonej rekurencyjnie wartości największego wspólnego dzielnika.

```
int RozszerzonyEuklides(int a, int b, int* x, int* y)
{
    if (a == 0)
    {
        *x = 0, * y = 1;
        return b;
    }
    int x1, y1;
```

```
    int NWD = RozszerzonyEuklides(b % a, a, &x1, &y1);
    *x = y1 - (b / a) * x1;
    *y = x1;
    return NWD;
}
```

Przedstawiona poniżej funkcja pomocnicza ma na celu wyznaczenie odwrotności modulo podanych jej liczb.

Pierwszym krokiem będzie stworzenie zmiennych `x` i `y`, których adresy następnie zostaną użyte przy wywołaniu poprzednio opisanej funkcji.

Aby możliwe było wykonanie operacji odwrotności modulo, liczby, na których chcemy wykonać tę operację, muszą być względnie pierwsze. Sprawdzamy to za pomocą opisanej wcześniej funkcji realizującej rozszerzony algorytm Euklidesa.

Jeśli funkcja obliczająca największy wspólny dzielnik zwróciła wartość inną niż 1, oznacza to, że liczby, na których chcemy wykonać operację odwrotności modulo, nie są względnie pierwsze, zatem powinniśmy zakończyć wykonywanie funkcji z kodem oznaczającym błąd.

Jeżeli liczby, na których chcemy wykonać działanie, są względnie pierwsze, obliczamy odwrotność modulo z wykorzystaniem wyznaczonej przez rozszerzony algorytm Euklidesa wartości `x` i zwracamy wynik.

```
int odwrotnoscMod(int a, int b)
{
    int x, y;
    int g = RozszerzonyEuklides(a, b, &x, &y);
    if (g != 1) //jeśli g != 1 to liczby a i b nie są względnie p
        return -1;
    else
    {
        /*Liczba x może być ujemna, aby tego uniknąć
        dodajemy b, jeśli liczba jest dodatnia, to zewnętrzna
        operacja % gwarantuje nam poprawny wynik*/
        int odwrotnosc = (x%b + b) % b;
        return odwrotnosc;
    }
}
```

W tym momencie posiadamy już wszystkie narzędzia do wyznaczania klucza publicznego, klucza prywatnego oraz do szyfrowania i deszyfrowania wiadomości.

Aby zaszyfrować wiadomość, dzielimy ją na bloki o wartości liczbowej nie większej niż `n`, a następnie implementujemy następującą funkcję, w której `P` oznacza szyfrowaną wiadomość:

$$f(P) = P^e \ mod \ n$$

Warto zwrócić uwagę, że wartości `(e, n)` stanowią klucz publiczny osoby, do której wysyłamy wiadomość.

Funkcje szyfrującą i deszyfrującą łatwo jest zaimplementować z wykorzystaniem funkcji `pow()`, dostępnej w bibliotece `cmath`.

```
unsigned long long szyfrowanieRSA(int wiadomosc, int e, int n)
{
    return (unsigned long long)pow(wiadomosc, e) % n;
}
```

Jeżeli nie jest możliwe użycie funkcji `pow()`, to funkcja szyfrująca może mieć następującą formę:

```
unsigned long long szyfrowanieRSA(int wiadomosc, int e, int n)
{
    unsigned long long rezultat = wiadomosc;
    for (int i = 1; i < e; i++)
    {
        rezultat = (rezultat * wiadomosc) % n;
    }
    return rezultat;
}
```

Aby odszyfrować wiadomość, wykonujemy następującą funkcję, w której `C` oznacza szyfrogram, a para `(d, n)` jest kluczem prywatnym osoby, która odebrała tę wiadomość:

$$f^{-1}(C) = C^d \ mod \ n$$

```
unsigned long long deszyfrowanieRSA(int szyfrogram, int d, int n)
{
    return (unsigned long long)pow(szyfrogram, d) % n;
}
```

Podobnie jak w poprzednim przypadku – jeżeli nie jest możliwe użycie funkcji `pow()`, to funkcja szyfrująca może przyjąć następującą postać:

```
unsigned long long deszyfrowanieRSA(int szyfrogram, int d, int n)
{
    unsigned long long rezultat = szyfrogram;
    for (int i = 1; i < d; i++)
    {
        rezultat = (rezultat * szyfrogram) % n;
    }
    return rezultat;
}
```

Problemy w implementacji szyfru RSA w języku C++

Specyfikacja problemu:

Dane:

- `p` – liczba naturalna pierwsza; jedna z liczb pierwszych potrzebnych do wyznaczenia kluczy
- `q` – liczba naturalna pierwsza; druga z liczb pierwszych potrzebnych do wyznaczenia kluczy
- `wiadomoscSzyfrowana` – liczba naturalna; liczba szyfrowana

Wynik:

- `szyfrogram` – zaszyfrowana wiadomość

Implementując szyfr RSA w języku C++, możemy napotkać problemy, wynikające z ograniczeń samego języka. Liczby pierwsze używane do generowania klucza publicznego i prywatnego powinny być „duże”. Język C++ domyślnie oferuje jedynie liczby 8-bajtowe bez znaku (*unsigned long long*) – liczba taka przyjmuje wartości od 0 do 18 446 744 073 709 551 615. Nie jest ona wystarczająca dla większych wartości kluczy. Dlatego używanie tego algorytmu w języku C++ bez wprowadzania nowych typów danych bardzo ogranicza jego możliwości.

> **Przykład 2**

> Dla większych wartości kluczy zakres zmiennej w postaci *unsigned long long* może być niewystarczający. Dla danych: `p` = 17, `q` = 29 i `e` = 3 otrzymujemy `n` = 493 oraz `d` = 299. Zmienna `wiadomoscSzyfrowana` podnoszona jest do trzeciej potęgi, co w przypadku „większych" wiadomości może okazać się za dużą wartością, nawet dla zmiennej typu *unsigned long long*. Dodatkowo podczas deszyfrowania szyfrogram podnoszony jest do 299 potęgi, co dla dowolnej liczby większej od 1 daje wynik, którego wartość zdecydowanie wykracza poza zakres. Jest to przypadek skrajny, jednak wyraźnie pokazuje, że używanie szyfru RSA wymaga dużej mocy obliczeniowej.

Przykładowy program:

```
#include <iostream>
#include <cmath>

using namespace std;

int RozszerzonyEuklides(int a, int b, int *x, int *y)
{
    if (a == 0)
    {
        *x = 0, *y = 1;
        return b;
    }
    int x1, y1;
    int NWD = RozszerzonyEuklides(b%a, a, &x1, &y1);
    *x = y1 - (b/a) * x1;
    *y = x1;

    return NWD;
}

int odwrotnoscMod(int a, int b)
{
    int x, y;
    int g = RozszerzonyEuklides(a, b, &x, &y);
    if (g != 1) //jeśli g != 1 to liczba nie jest pierwsza
        return -1;
    else
    {
        /*Liczba x może być ujemna, aby tego uniknąć
        dodajemy m, jeśli liczba jest dodatnia, to zewnętrzna
```

```
        operacja % gwarantuje nam poprawny wynik*/
        int odwrotnosc = (x%b + b) % b;
        return odwrotnosc;
    }
}

unsigned long long szyfrowanieRSA(int wiadomosc, int e, int n)
{
    return (unsigned long long)pow(wiadomosc, e) % n;
}

unsigned long long deszyfrowanieRSA(int szyfrogram, int d, int n)
{
    return (unsigned long long)pow(szyfrogram, d) % n;
}

int funkcjaEulera(int p, int q)
{
    return (p-1)*(q-1);
}

int main()
{
    //Jako osoba chcąca odbierać zaszyfrowane wiadomości generuję
    int p = 5;
    int q = 7;
    int n = p * q;
    int euler = funkcjaEulera(p, q);
    int e = 1;
    int pom1; //zmienne pomocnicze
    int pom2;//Aby użyć rozszerzonego algorytmu Euklidesa jak zwy
    do //odnajdywanie e takiego, że 1 < e < euler
    {
        e++;
    }
    while(RozszerzonyEuklides(e, euler, &pom1, &pom2) != 1);
    int d = odwrotnoscMod(e, euler);
    /*e, n - klucz publiczny
    d, n - klucz prywatny*/
    int wiadomoscSzyfrowana = 32;
    int szyfrogram = szyfrowanieRSA(wiadomoscSzyfrowana, e, n);
    cout<<"Szyfrogram to: "<<szyfrogram<<" "<<endl;
```

```
73
74      return 0;
75  }
```

Wynik działania programu:

```
1  Szyfrogram to: 2
```

Wynikiem działania programu, szyfrowania liczby naturalnej, jest inna liczba naturalna.

Słownik

algorytm Euklidesa

algorytm służący do wyznaczania największego wspólnego dzielnika (NWD) podanych dwóch liczb całkowitych

funkcja Eulera

(funkcja ϕ) funkcja przypisująca każdej liczbie naturalnej liczbę liczb względnie pierwszych z nią i nie większych od niej

klucz prywatny

rodzaj klucza służącego w szyfrach asymetrycznych do deszyfrowania szyfrogramów; dostęp do niego powinien mieć tylko odbiorca zaszyfrowanych wiadomości

klucz publiczny

rodzaj klucza służącego w szyfrach asymetrycznych do szyfrowania tekstu jawnego; nie musi być utrzymywany w sekrecie, a nawet może zostać udostępniony publicznie

liczby względnie pierwsze

liczby całkowite, których największym wspólnym dzielnikiem jest 1

sito Eratostenesa

algorytm wyznaczania liczb pierwszych w zadanym przedziale

Sprawdź się

Pokaż ćwiczenia:

Ćwiczenie 1

Napisz program obliczający odwrotność modulo z wykorzystaniem rozszerzonego algorytmu Euklidesa. Program ma obliczyć $e \bmod \mathrm{euler}$, gdzie e to pierwsza wyznaczona liczba względnie pierwsza z wartością euler, a euler jest równe wartości funkcji Eulera dla dwóch liczb pierwszych: p i q. Program ma zwrócić liczbę będącą wynikiem odwrotności modulo. Wartość e ma spełniać warunek $1 < e < \mathrm{euler}$.

Informacje na temat funkcji Eulera znajdziesz w sekcji „Przeczytaj".

Program przetestuj dla wartości $p = 17$ i $q = 13$.

Specyfikacja problemu:

Dane:

- p – liczba całkowita dodatnia; wartość jednej z liczb pierwszych potrzebnych do obliczenia odwrotności modulo
- q – liczba całkowita dodatnia; wartość drugiej z liczb pierwszych potrzebnych do obliczenia odwrotności modulo

Wynik:

- `odwrotnosc` – liczba naturalna; wyznaczona wartość odwrotności modulo

Ćwiczenie 2

Napisz program wyznaczający klucz publiczny i klucz prywatny na podstawie podanej pary liczb pierwszych p i q. Uzyskane klucze podaj w formie par $(e,\ n)$ i $(d,\ n)$ oddzielonych znakiem nowej linii.

Program przetestuj dla wartości $p = 23$ i $q = 11$.

Specyfikacja problemu:

Dane:

- p – liczba całkowita dodatnia; wartość jednej z liczb pierwszych potrzebnych do wyznaczenia wartości kluczy
- q – liczba całkowita dodatnia; wartość drugiej z liczb pierwszych potrzebnych do wyznaczenia wartości kluczy

Wynik:

- klucz publiczny i klucz prywatny

Ćwiczenie 3

Napisz program pozwalający na zaszyfrowanie i odszyfrowanie podanych wartości liczbowych szyfrem RSA, na podstawie podanych liczb pierwszych p i q. W zadaniu nie używaj funkcji `pow()`.

Program przetestuj dla wartości $p = 23$ i $q = 13$. Otrzymanymi kluczami zaszyfruj liczbę 72 i odszyfruj liczbę, której szyfrogram ma postać liczby 75.

Specyfikacja problemu:

Dane:

- `p` – liczba całkowita dodatnia; wartość jednej z liczb pierwszych potrzebnych do wyznaczenia wartości kluczy
- `q` – liczba całkowita dodatnia; wartość drugiej z liczb pierwszych potrzebnych do wyznaczenia wartości kluczy
- `liczbaDoZaszyfrowania` – liczba naturalna; liczba do zaszyfrowania kluczem publicznym
- `szyfrogram` – liczba naturalna; wiadomość do odszyfrowania kluczem prywatnym

Wynik:

- zaszyfrowana postać liczby `liczbaDoZaszyfrowania`
- odszyfrowana postać szyfrogramu `szyfrogram`

Dla nauczyciela

Autor: Maurycy Gast
Przedmiot: Informatyka

Temat: Szyfr RSA w języku C++

Grupa docelowa:

Szkoła ponadpodstawowa, 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.

V. Przestrzeganie prawa i zasad bezpieczeństwa. Respektowanie prywatności informacji i ochrony danych, praw własności intelektualnej, etykiety w komunikacji i norm współżycia społecznego, ocena zagrożeń związanych z technologią i ich uwzględnienie dla bezpieczeństwa swojego i innych.

Treści nauczania – wymagania szczegółowe

I. Rozumienie, analizowanie i rozwiązywanie problemów.

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

3) objaśnia dobrany algorytm, uzasadnia poprawność rozwiązania na wybranych przykładach danych i ocenia jego efektywność;

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

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

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

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

3) objaśnia, a także porównuje podstawowe metody i techniki algorytmiczne oraz struktury danych, wykorzystując przy tym przykłady problemów i algorytmów, w szczególności:

f) metodę szyfrowania z kluczem publicznym i jej zastosowanie w podpisie elektronicznym,

V. Przestrzeganie prawa i zasad bezpieczeństwa.

Zakres podstawowy. Uczeń:

3) stosuje dobre praktyki w zakresie ochrony informacji wrażliwych (np. hasła, pin), danych i bezpieczeństwa systemu operacyjnego, objaśnia rolę szyfrowania informacji;

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

1) objaśnia rolę technik uwierzytelniania, kryptografii i podpisu elektronicznego w ochronie i dostępie do informacji;

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):

- Zaimplementujesz w języku C++ klucze prywatny i publiczny szyfru RSA.
- Napiszesz program, który za pomocą algorytmu RSA wygeneruje podpis i zweryfikuje jego poprawność.
- Wykonasz ćwiczenia dotyczące szyfrowania i deszyfrowania wiadomości za pomocą szyfru RSA.

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: „Szyfr RSA w języku C++". Nauczyciel prosi uczniów o zapoznanie się z treściami w sekcji „Film samouczek".

Faza wstępna:

1. Nauczyciel wyświetla temat i cele zajęć zawarte w sekcji „Wprowadzenie". Prosi uczniów, by na podstawie wiadomości zdobytych przed lekcją zaproponowali kryteria sukcesu.
2. **Rozpoznanie wiedzy uczniów.** Nauczyciel prosi wybranego ucznia lub uczniów o przedstawienie sytuacji problemowej związanej z tematem lekcji.

Faza realizacyjna:

1. **Praca z multimedium.** Nauczyciel wyświetla zawartość sekcji „Film samouczek". Prosi wybrane osoby o przedstawienie rozwiązania problemu 1, które uczniowie przygotowywali przed lekcją. Pozostali uczniowie komentują przedstawione programy. W razie konieczności nauczyciel wyjaśnia wątpliwości i odpowiada na pytania.
2. **Praca z tekstem.** Nauczyciel wyświetla zawartość sekcji „Przeczytaj". Uczniowie indywidualnie zapoznają się z jej treścią. Zapisują ewentualne problemy i pytania. Po czym następuje dyskusja, w trakcie której nauczyciel wyjaśnia niezrozumiałe treści.

3. **Ćwiczenie umiejętności.** Prowadzący zapowiada uczniom, że będą rozwiązywać ćwiczenie nr 1 z sekcji „Sprawdź się". Uczniowie wykonują je w parach. Po ustalonym czasie następuje porównanie napisanych kodów podczas wspólnego omówienia rozwiązań.
4. Liga zadaniowa - uczniowie pracując w parach, wykonują ćwiczenie nr 2 z sekcji „Sprawdź się", a następnie dzielą się swoimi wynikami przez porównanie napisanego kodu z inną grupą, która również zakończyła zadanie.

Faza podsumowująca:

1. Nauczyciel ponownie wyświetla na tablicy temat i cele lekcji zawarte w sekcji „Wprowadzenie". W kontekście ich realizacji następuje omówienie ewentualnych problemów z rozwiązaniem ćwiczeń z sekcji „Sprawdź się".
2. Na koniec zajęć z programowania w C++ nauczyciel prosi uczniów o rozwinięcie zdania: „Na dzisiejszych zajęciach nauczyłam/łem się jak...".

Praca domowa:

1. Uczniowie wykonują ćwiczenie nr 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 „Film samouczek" można wykorzystać jako podsumowanie i utrwalenie wiedzy uczniów.