Wyznaczanie liczby π metodą Monte Carlo w języku C++

Wyznaczanie liczby π metodą Monte Carlo w języku C++

Źródło: Nadine Shaabana, domena publiczna.

Istnieje wiele rodzajów obliczeń matematycznych, które trudno jest wykonać, korzystając z metod analitycznych. Czasami okazuje się to wręcz niemożliwe. Za przykład niech posłużą obliczenia całek oznaczonych funkcji, wyznaczanie pól powierzchni figur o skomplikowanych kształtach lub działania na liczbach niewymiernych.

W takich sytuacjach sięga się po numeryczne techniki obliczeniowe. Jednej z nich – metodzie Monte Carlo – poświęcimy ten e-materiał.

Czy potrafisz wskazać inne sytuacje, w których sięga się po numeryczne techniki obliczeniowe?

Podstawowe informacje na temat wyznaczania liczby π metodą Monte Carlo znajdziesz w e-materiale: Wyznaczanie liczby π metodą Monte Carlo.

W tym e-materiale poznamy specyfikę wyznaczania liczby π metodą Monte Carlo w języku C++.

Jeśli chcesz dowiedzieć się, jak to zagadnienie wygląda w przypadku innych języków programowania, sięgnij do e-materiałów:

- Wyznaczanie liczby π metodą Monte Carlo w języku Python,

- Wyznaczanie liczby π metodą Monte Carlo w języku Java.

Twoje cele

- Scharakteryzujesz sposoby generowania liczb losowych w języku C++.
- Przeanalizujesz, jak losuje się punkty w obrębie danej figury geometrycznej.
- Zaimplementujesz algorytm obliczania wartości liczby π metodą Monte Carlo w języku C++.

Film samouczek

Problem 1

Napisz program, który wyznaczy przybliżoną wartość liczby π. Wykorzystaj metodę

Specyfikacja problemu:

Dane:

- `liczba_prob` – liczba prób podczas używania metody Monte Carlo; liczba naturalna większa niż 0

Wynik:

- `pi` – obliczona wartość liczby π; liczba rzeczywista

```
#include <iostream>
#include <time.h>

using namespace std;

int main ()

{

// tutaj uzupełnij swój kod

return 0;

}
```

```
```

Polecenie 1

Porównaj swoje rozwiązanie z przedstawionym w filmie.

Film dostępny pod adresem /preview/resource/R1Ja2wxDWkaDZ

Lekcja poświęcona wyznaczaniu liczby π metodą Monte Carlo w języku C++.

Plik o rozmiarze 823.00 B w języku polskim

Polecenie 2

Sprawdź, co się stanie, gdy użyjesz generatora `random_device` zamiast generatora `mt19937`. Aby przeanalizować, co się dzieje, wypisuj w konsoli wszystkie wylosowane punkty.

Przeczytaj

Generator liczb pseudolosowych

W programowaniu nie mamy dostępu do prawdziwie losowych liczb, dlatego używamy generatorów liczb pseudolosowych. Są one powtarzalne, co oznacza, że możemy w identyczny sposób powtórzyć proces losowania. W języku C++ możemy ustawić ziarno dla generatora pseudolosowego. Należy jednak pamiętać, że implementacja takiego generatora może być inna w zależności od kompilatora. Co oznacza, że powtórzenie procesu losowania na innym komputerze może dać inne rezultaty. Ziarno w języku C++ możemy ustawić funkcją `srand()` z biblioteki `<cstdlib>` w następujący sposób:

```
unsigned int ziarno = 15;
srand(ziarno);
```

Poniżej prezentujemy kod, który trzy razy wywołuje funkcję wyznaczającą liczbę π metodą Monte Carlo. Za każdym razem przed rozpoczęciem procesu losowania ustawiane jest jednak inne ziarno.

```
#include <iostream>
#include <cstdlib>

using namespace std;

double wyznaczPi(int liczbaLosowan) {
    int liczbaTrafien = 0;

    for (int i = 0; i < liczbaLosowan; ++i) {
        // losowe liczby z przedziału <-1, 1>
        double x = 2 * double(rand()) / RAND_MAX - 1;
        double y = 2 * double(rand()) / RAND_MAX - 1;

        if (x * x + y * y <= 1) {
            ++liczbaTrafien;
        }
    }

    double pi = 4 * double(liczbaTrafien) / liczbaLosowan;

```

```
    return pi;
}

int main() {
    srand(1);
    cout << wyznaczPi(10000) << endl;
    srand(2);
    cout << wyznaczPi(10000) << endl;
    srand(3);
    cout << wyznaczPi(10000) << endl;

    return 0;
}
```

Przykładowe wyniki uzyskane po uruchomieniu programu:

```
3.1464
3.1568
3.12
```

> **Polecenie 1**
>
> Przeanalizuj prezentację, w której porównujemy dwa dostępne w języku C++ generatory liczb pseudolosowych.

Wiem już, w jaki sposób można przeprowadzić symulację metodą Monte Carlo, wykorzystując funkcję `rand()` z biblioteki `<cstdlib>`. Przypomnijmy, że kod z implementacją takiego rozwiązania prezentuje się następująco:

```
#include <iostream>
#include <cstdlib>

using namespace std;

```

```
6  double wyznaczPi(int
   liczbaProb) {
7      int punktyKola = 0;
8
9      for (int i = 0; i <
   liczbaProb; ++i) {
10         // losowe liczby z
   przedziału <-1, 1>
11         double losowyX = 2
   * double(rand()) /
   RAND_MAX - 1;
12         double losowyY = 2
   * double(rand()) /
   RAND_MAX - 1;
13
14         if (losowyX *
   losowyX + losowyY *
   losowyY <= 1) {
15             ++punktyKola;
16         }
17     }
18
19     double pi = 4 *
   double(punktyKola) /
   liczbaProb;
20
21     return pi;
22 }
23
24 int main() {
25     cout <<
   wyznaczPi(10000) << endl;
26
27     return 0;
28 }
```

2

Innym (nowocześniejszym) rozwiązaniem jest zastosowanie generatora z biblioteki `<random>`.

3

Stwórzmy generator liczb pseudolosowych. Spośród wielu możliwości dostępnych w bibliotece standardowo zalecane jest używanie generatora Mersenne Twister w wariancie MT19937. Generatory z biblioteki `<random>` można zainicjować ziarnem, podając je jako parametr podczas inicjalizacji.

Implementacja generatora MT19937 w języku C++ wygląda następująco:

```
int ziarno = 1;
mt19937 generator(ziarno);
```

4

W kolejnym kroku implementujemy rozkład liczb pseudolosowych. W ramach omawianej biblioteki najważniejsze dystrybucje, z których możemy skorzystać w tym celu, to `uniform_int_distribution<>` oraz `uniform_real_distribution<>`, czyli rozkłady jednorodne określone na liczbach całkowitych lub zmiennoprzecinkowych. Podczas inicjalizacji wybranej dystrybucji podajemy przedział (obustronnie domknięty), w którym znajdować mają się losowane liczby. Wewnątrz nawiasów określamy typ zmiennej, którą będzie zwracał utworzony rozkład. Linijka kodu, w której zapisano jednorodny rozkład prawdopodobieństwa z przedziału [-1,1]:

```
uniform_real_distribution<
double> dystrybucja(-1.0,
1.0);
```

5

Fragment kodu odpowiedzialny za generowanie losowych liczb z utworzonego rozkładu:

```
double losowaLiczba = dystrybucja(generator);
```

6

Zastanów się, w jaki sposób zmodyfikować napisaną wcześniej funkcję `wyznaczPi()`, wykorzystując do tego bibliotekę `<random>`.

7

Przykładowe rozwiązanie, w którym nie wykorzystano ziarna w celu inicjalizacji generatora:

```
#include <iostream>
#include <cstdlib>
#include <random>

using namespace std;

double wyznaczPi_nowy(int liczbaProb) {
    int punktyKola = 0;

    // inicjalizacja generatora
    mt19937 generator;

    // inicjalizacja rozkładu jednorodnego <-1, 1>

    uniform_real_distribution dystrybucja(-1.0, 1.0);

    for (int i = 0; i < liczbaProb; ++i) {
```

```
17          // losowe liczby z
przedziału <-1, 1>
18          double losowyX =
dystrybucja(generator);
19          double losowyY =
dystrybucja(generator);
20
21          if (losowyX *
losowyX + losowyY *
losowyY <= 1) {
22              ++punktyKola;
23          }
24      }
25
26      double pi = 4 *
double(punktyKola) /
liczbaProb;
27
28      return pi;
29  }
30
31  int main(int argc, char
**argv)
32  {
33      cout <<
wyznaczPi_nowy(100000);
34      return 0;
35  }
```

8

Zdarzają się przypadki, w których chcielibyśmy w losowy sposób wybrać ziarno dla generatora. W takiej sytuacji stosujemy bibliotekę `<random>`. Zawiera ona specjalny generator losowy, którego działanie opiera się na losowości pochodzącej ze sterowników. Generator ten nie nadaje się do wielokrotnego użytku, ponieważ losowość nie zawsze zmieni się przed kolejnym użyciem losowej zmiennej. W rezultacie może się zdarzyć, że program będzie korzystał z takiej samej liczby losowej

przez dłuższy czas. Generator taki sprawdza się, gdy chcemy ustawić ziarno dla innego generatora.

9

Przykład pokazujący, w jaki sposób można zainicjalizować generator losowym ziarnem:

```cpp
#include <iostream>
#include <cstdlib>
#include <random>

using namespace std;

double wyznaczPi_nowy(int liczbaProb) {
    int punktyKola = 0;

    // inicjalizacja generatora
    random_device generator_ziarna;
    // inicjalizacja generatora z losowym ziarnem
    mt19937 generator(generator_ziarna());
    // inicjalizacja rozkładu jednorodnego <-1, 1>

    uniform_real_distribution dystrybucja(-1.0, 1.0);

    for (int i = 0; i < liczbaProb; ++i) {
        // losowe liczby z przedziału <-1, 1>
        double losowyX = dystrybucja(generator);
```

```
20          double losowyY =
dystrybucja(generator);
21
22          if (losowyX *
losowyX + losowyY *
losowyY <= 1) {
23              ++punktyKola;
24          }
25      }
26
27      double pi = 4 *
double(punktyKola) /
liczbaProb;
28
29      return pi;
30  }
31
32  int main()
33  {
34      cout <<
wyznaczPi_nowy(100000);
35      return 0;
36  }
```

10 Uwaga, zaprezentowane alternatywy mogą nie działać w środowisku Code::Blocks.

> **Polecenie 2**
>
> Przeprowadź 100 razy symulację dla 1000 losowań. Za każdym razem zainicjuj generator losowym ziarnem według opisanej w prezentacji metody. Oblicz średnią arytmetyczną uzyskanych wyników. Następnie przeprowadź jedną symulację zainicjowaną losowym ziarnem dla

Słownik

ziarno

liczba lub wektor służący do zainicjowania generatora pseudolosowego
generator liczb pseudolosowych

podprogram zwracający kolejne liczby z deterministycznego ciągu liczb, który ma podobne własności do ciągów losowych
Stanisław Ulam

polski matematyk (ur. 13 kwietnia 1909 r. we Lwowie, zm. 13 maja 1984 r. w Santa Fe), zaliczany do lwowskiej szkoły matematycznej, posiadający wybitny wkład do teorii mnogości oraz topologii, twórca metod numerycznych, współtwórca bomby atomowej
skala logarytmiczna

rodzaj skali pomiarowej, w której wartość przekształcana jest za pomocą funkcji logarytmicznej, najczęściej o podstawie 2, 10 lub e

Sprawdź się

Pokaż ćwiczenia: 

Ćwiczenie 1

Za pomocą metody Monte Carlo wyznacz stosunek pola koła wpisanego w kwadrat do pola tego kwadratu. Użyj n losowych punktów. Przetestuj działanie programu dla `n = 10000`.

Specyfikacja problemu:

Dane:

- n – liczba naturalna

Wynik:

- stosunek pola koła wpisanego w kwadrat do pola tego kwadratu

Twoje zadania

1. Program wypisuje stosunek pola koła wpisanego w kwadrat do pola tego kwadratu.

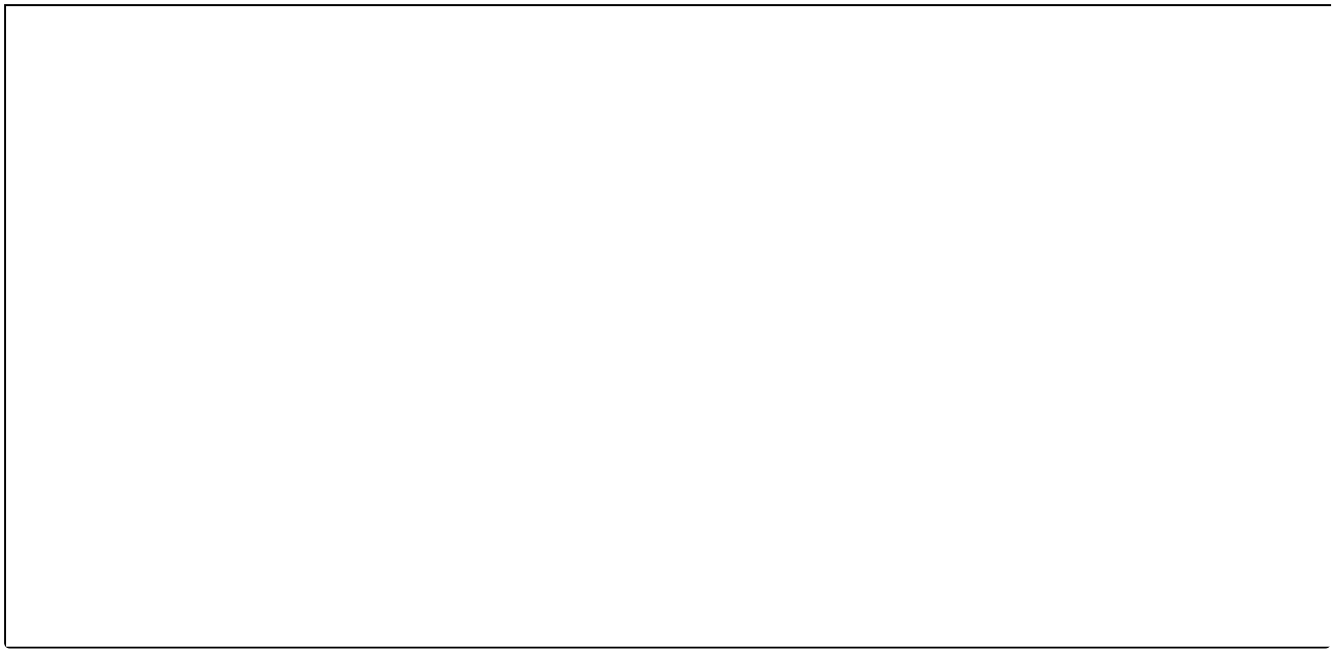
```cpp
#include <iostream>

using namespace std;

int main() {
    // W tym miejscu napisz implementację swojego rozwiązania

    return 0;
}
```

```
1
```



Ćwiczenie 2

Za pomocą metody Monte Carlo wyznacz stosunek objętości kuli wpisanej w sześcian do objętości tego sześcianu. Użyj n losowych punktów. Wykorzystaj równanie kuli o środku w punkcie (a, b, c):

$$(x-a)^2 + (y-b)^2 + (z-c)^2 \leq r^2$$

Przetestuj działanie programu dla `n = 10000`.

Specyfikacja problemu:

Dane:

- n – liczba naturalna

Wynik:

- stosunek objętości kuli wpisanej w sześcian do objętości tego sześcianu

Twoje zadania

1. Program wypisuje stosunek objętości kuli wpisanej w sześcian do objętości tego sześcianu.

```cpp
#include <iostream>

int main() {
    // W tym miejscu napisz implementację swojego rozwiązania

    return 0;
}
```

```
1
```

Ćwiczenie 3

Napisz program, który za pomocą metody Monte Carlo obliczy pole koła. Przetestuj swój program dla koła o promieniu $r = 2$. Wynik wypisz z dokładnością do trzech miejsc po przecinku (nie zaokrąglaj).

Specyfikacja:

Dane:

- `r` - promień koła, którego pole należy obliczyć; liczba rzeczywista

Wynik:

- pole koła o promieniu `r`

Twoje zadania

1. Program oblicza pole koła o danym promieniu metodą Monte Carlo.

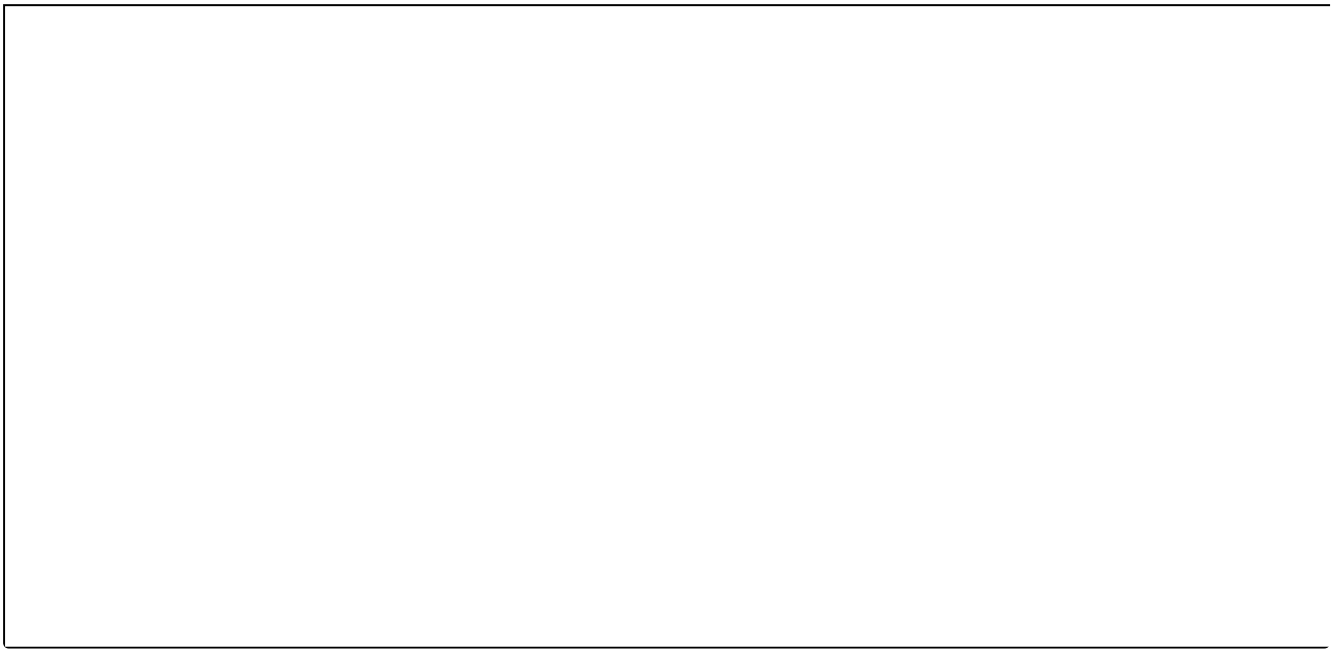
```cpp
#include <iostream>

using namespace std;

int main() {
    // W tym miejscu napisz implementację swojego rozwiązania

    return 0;
}
```

```
1
```



Dla nauczyciela

Autor: Maurycy Gast
Przedmiot: Informatyka

Temat: Wyznaczanie liczby π metodą Monte Carlo 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.

Treści nauczania – wymagania szczegółowe

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

Zakres podstawowy. Uczeń:

2) do realizacji rozwiązań problemów prawidłowo dobiera środowiska informatyczne, aplikacje oraz zasoby, wykorzystuje również elementy robotyki;

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:

h) metodę Monte Carlo (obliczanie przybliżonej wartości liczby π, symulacja ruchów Browna),

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):

- Scharakteryzujesz sposoby generowania liczb losowych w języku C++.
- Przeanalizujesz, jak losuje się punkty w obrębie danej figury geometrycznej.
- Zaimplementujesz algorytm obliczania wartości liczby π metodą Monte Carlo w języku C++.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

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

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: „Wyznaczanie liczby π metodą Monte Carlo w języku C++". Nauczyciel prosi uczniów o zapoznanie się z treściami w sekcji „Film samouczek".

Faza wstępna:

1. Prowadzący wyświetla na tablicy interaktywnej zawartość sekcji „Wprowadzenie" i omawia cele do osiągnięcia w trakcie lekcji.
2. **Rozpoznanie wiedzy uczniów.** Nauczyciel zadaje uczniom pytania dotyczące ich aktualnego stanu wiedzy w obszarze poruszanego tematu, np.
   – w jaki sposób działa metoda Monte Carlo?
   – jak za jej pomocą wyznaczyć liczbę π?
   Chętni lub wybrani uczniowie udzielają na nie odpowiedzi.

Faza realizacyjna:

1. **Praca z multimedium.** Nauczyciel prosi uczniów o przygotowanie i przedstawienie rozwiązania polecenia nr 1 w ramach problemu 1 z sekcji „Film samouczek". W razie potrzeby, jeśli przygotowanie do lekcji jest niewystarczające, prosi by uczniowie ponownie zapoznali się z treścią filmu. Następnie uczniowie wykonują w parach polecenie nr 2 z tej samej sekcji.
2. **Praca z tekstem.** Nauczyciel wyświetla zawartość sekcji „Przeczytaj". Uczniowie analizują treści zawarte w prezentacji oraz testują przedstawione tam podprogramy na swoich komputerach. Następnie uczniowie indywidualnie wykonują polecenie nr 2.
3. **Ćwiczenie umiejętności.** Uczniowie wykonują indywidualnie ćwiczenia nr 1–2 z sekcji „Sprawdź się". Po wykonaniu każdego z nich następuje omówienie rozwiązania przez nauczyciela.

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.
2. Wybrany uczeń podsumowuje zajęcia z programowania w C++, zwracając uwagę na nabyte umiejętności.

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 „Przeczytaj" można wykorzystać na lekcji jako podsumowanie i utrwalenie wiedzy uczniów.