Sito Eratostenesa w języku C++

Źródło: congerdesign, domena publiczna.

Sito Eratostenesa jest algorytmem wyznaczającym liczby pierwsze z zadanego przedziału <2, n>. Liczby pierwsze znajdują zastosowanie między innymi w kryptografii, o czym przeczytasz m.in. w e-materiałach:

- Zastosowanie liczb pierwszych,
- Szyfr RSA.

Ten e-materiał poświęcony jest implementacji algorytmu sita Eratostenesa w języku C++.

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

- Sito Eratostenesa w języku Java,
- Sito Eratostenesa w języku Python.

Więcej zadań? Sięgnij do: Sito Eratostenesa – zadania maturalne.

> **Twoje cele**
>
> - Przeanalizujesz zasadę działania algorytmu sita Eratostenesa.
> - Zaimplementujesz algorytm Eratostenesa w języku C++.
> - Rozwiążesz kilka problemów wymagających wyznaczenia liczb pierwszych z zadanego przedziału <2, n>.

Film samouczek

Polecenie 1

Napisz program wyszukujący w zadanym przedziale <2, n> liczby pierwsze. Swoje rozwiązanie porównaj z filmem.

Przetestuj rozwiązanie dla *n* wynoszącego 99.

Specyfikacja problemu:

Dane:

- n – liczba naturalna dodatnia; górna granica przedziału

Wynik:

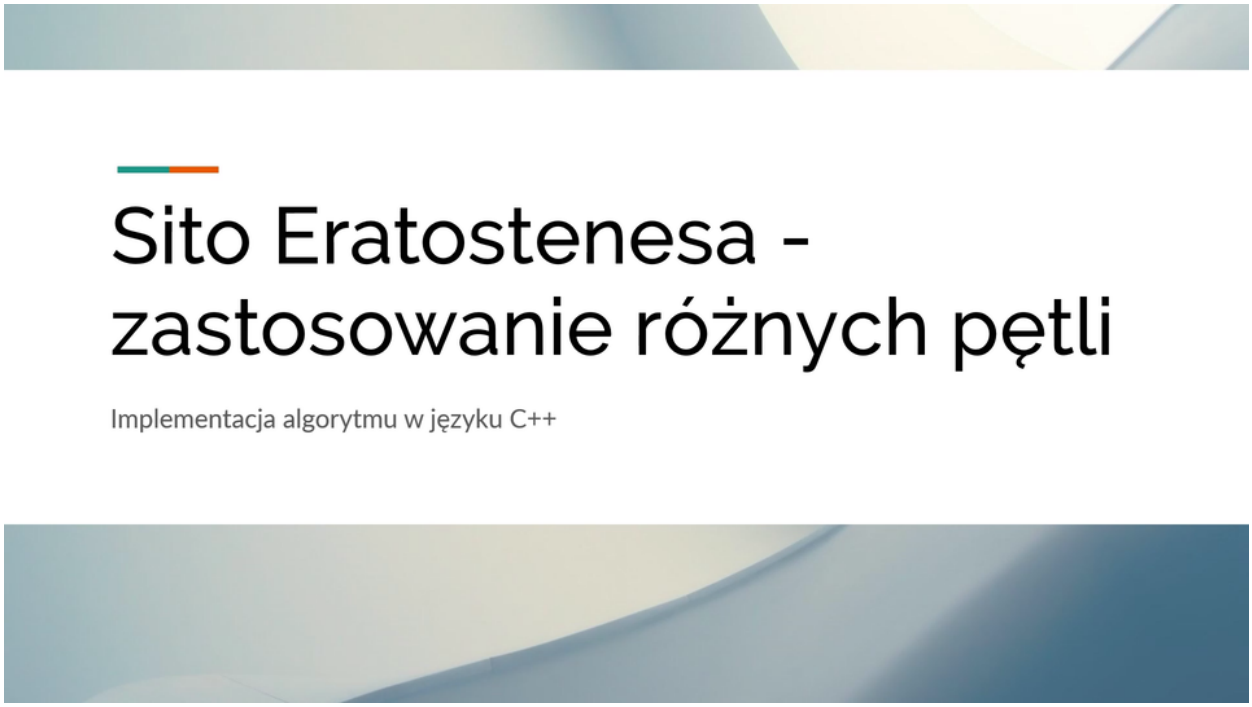
Program na standardowym wyjściu wyświetla kolejne liczby pierwsze z zakresu <2, n>.

```
#include <iostream>

int main () {

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

```

```

Polecenie 2

Porównaj swoje rozwiązanie z przedstawionym w filmie.



Film dostępny pod adresem /preview/resource/R1IVH5hshWBEE

Film nawiązujący do treści materiału: Sito Eratostenesa - zastosowanie różnych pętli.

Przeczytaj

Przykład optymalizacji sita Eratostenesa w języku C++

Algorytm sita Eratostenesa możemy optymalizować. Przeanalizujmy jeden z przykładów takiej optymalizacji.

Zanim przystąpimy do pisania kodu, trzeba zauważyć, że jedyną parzystą liczbą pierwszą jest 2 - każda kolejna liczba parzysta: 4, 6, 8, ... jest złożona.

Utwórzmy tablicę tab, składającą się z 26 komórek, której elementy będą przechowywać kolejne liczby nieparzyste, zaczynając od liczby 3. W języku C++ tablice indeksujemy od 0 – na potrzeby omawianego algorytmu komórkę kryjącą się pod tym indeksem wypełnimy wartością 0; pomoże nam to w lepszym zrozumieniu algorytmu. Tablica tab będzie więc składała się z następujących elementów: pod indeksem 1 będzie kryła się wartość 3. Możemy przyjąć, że: $tab[2] = 5$, $tab[3] = 7$ ... $tab[25] = 51$.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
k	0	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31	33	35	37	39	41	43	45	47	49	51

Zauważmy, że dzięki wypełnieniu tablicy danymi od indeksu 1 możemy wprowadzić korelacje między numerem indeksu, a wartością, jaką ten indeks przechowuje.

Komórka k o indeksie i będzie przechowywać liczbę o wartości:

$$k = 2 * i + 1$$

Np. komórka o indeksie 9 będzie przechowywać wartość $2 * 9 + 1 = 19$, z kolei komórka o indeksie 15: $2 * 15 + 1 = 31$.

Wyprowadzenie wzoru przedstawionego powyżej pozwala nam w łatwy sposób obliczyć indeks komórki, w której została zapisana wartość k.

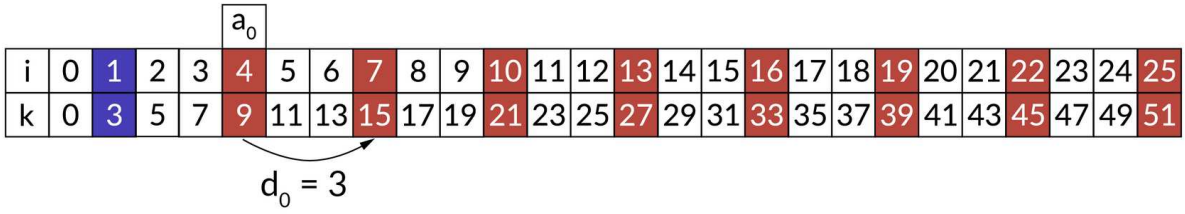
$i = \frac{k-1}{2}$

Ponownie rozpatrzmy kilka przykładów. Liczba 59 będzie przechowywana pod indeksem 29, ponieważ $\frac{59-1}{2} = 29$; z kolei liczba 49 będzie przechowywana pod indeksem 24.

Na potrzeby niniejszego algorytmu wprowadźmy dwie zmienne indeksowane:

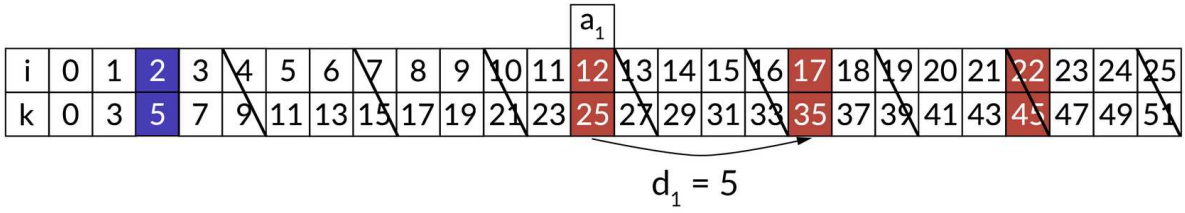
- a_n – indeks kwadratu liczby, której wielokrotność usuwamy z tablicy tab,
- d_n – odstęp (liczba komórek w tablicy tab) między kolejnymi wielokrotnościami, które będziemy wykreślać.

Algorytm zaprezentowany w poprzedniej sekcji tłumaczył, dlaczego pierwszą „wykreślaną" wielokrotnością jest kwadrat przetwarzanej liczby. Tablica tab pod indeksem 1 przechowuje liczbę 3 – jej kwadratem jest liczba 9, kryjąca się pod indeksem $a_0 = 4$. Zauważmy, że kolejne wielokrotności liczby 3 znajdują się w odległości $d_0 = 3$ – te liczby nie są liczbami pierwszymi; wykreślamy je.



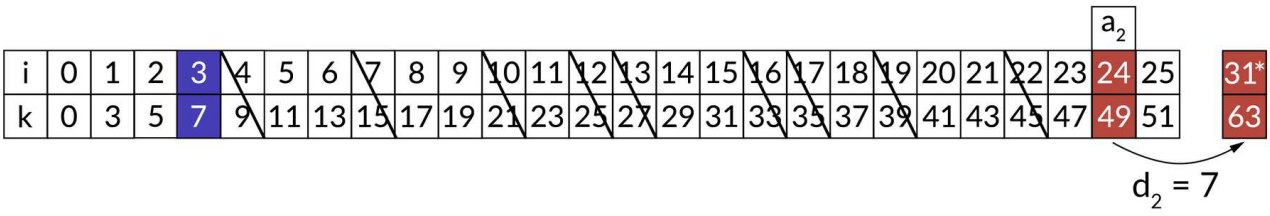
Źródło: Contentplus.pl sp. z o.o., licencja: CC BY-SA 3.0.

Kolejną niewykreśloną liczbą jest 5. Jej kwadrat – 25 kryje się pod indeksem $a_1 = 12$. Zauważmy, że wartość a_1 możemy przedstawić również jako $a_1 = a_0 + 8 = 4 + 8 = 12$ (dokładne omówienie wzoru nastąpi poniżej). Dystans między kolejnymi wielokrotnościami liczby 5 wynosi $d_1 = 5$ i możemy go przedstawić jako $d_1 = d_0 + 2 = 3 + 2 = 5$. Ponownie wykreślamy wielokrotności.



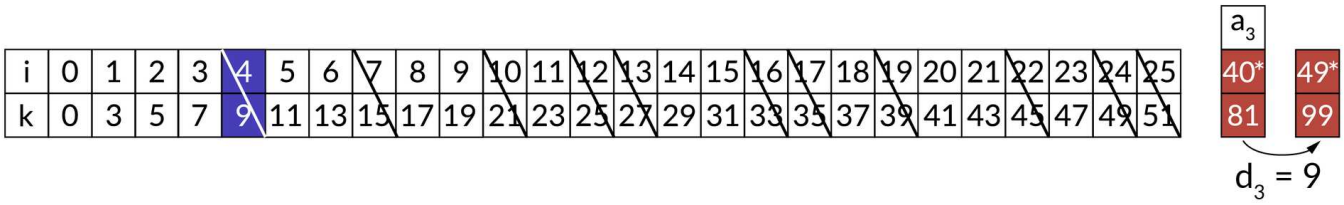
Źródło: Contentplus.pl sp. z o.o., licencja: CC BY-SA 3.0.

Proces powtarzamy dla liczby 7. Jej kwadrat – 49 został zapisany pod indeksem $a_2 = 24$ ($a_2 = a_1 + 12 = a_0 + 8 + 12 = 4 + 8 + 12 = 24$). Dystans między jej kolejnymi wielokrotnościami wynosi $d_2 = 7$ ($d_2 = d_1 + 2 = d_0 + 2 + 2 = 3 + 2 + 2 = 7$).



Źródło: Contentplus.pl sp. z o.o., licencja: CC BY-SA 3.0.

Następna liczba (9) została wykreślona już wcześniej. Aby lepiej zrozumieć algorytm, omówmy i ją. Wskazujemy indeks komórki, w której umieszczono kwadrat liczby 9, czyli 81: $a_3 = 40$ ($a_3 = a_2 + 16 = 24 + 16 = 40$). Odległość między wielokrotnościami wynosi $d_3 = 9$ ($d_3 = d_2 + 2 = 7 + 2 = 9$). Zauważmy, że liczba 81 (kwadrat liczby 9) znajduje się poza zakresem tablicy *tab*. Jest to moment, w którym przerywamy algorytm.



Źródło: Contentplus.pl sp. z o.o., licencja: CC BY-SA 3.0.

Na podstawie przedstawionych rozważań możemy zbudować tabelę:

i	d_i	$2 * (d_i - 1)$	a_i
0	3		4
1	3 + 2 = 5	2 * (5 – 1) = 8	4 + 8 = 12
2	5 + 2 = 7	2 * (7 – 1) = 12	12 + 12 = 24
3	7 + 2 = 9	2 * (9 – 1) = 16	24 + 16 = 40

Zauważmy, że kolejne d powstaje poprzez dodanie do poprzedniej wartości liczby 2. Podobną zależność jesteśmy w stanie wyprowadzić dla a_i – do poprzedniej wartości dodajemy $2 * (d_i - 1)$.

W ten sposób dotarliśmy do momentu, w którym możemy wyprowadzić następujący wzór rekurencyjny:

- $d_0 = 3$,
- $a_0 = 4$,
- dla każdego $i > 0$:
  - $d_i = d_{i-1} + 2$
  - $a_i = a_{i-1} + 2 * (d_i - 1)$.

Implementacja algorytmu

Implementacje algorytmu rozpocznijmy od zadeklarowania funkcji `main`, wczytania biblioteki `iostream` oraz poinformowania kompilatora o wykorzystaniu typów pochodzących z biblioteki standardowej.

```
#include <iostream>

using namespace std;


int main() {

}

```

Tworzymy zmienną `n`, która będzie przechowywać rozmiar generowanego sita. Zapisujemy w niej liczbę naturalną dodatnią pobraną od użytkownika.

```
#include <iostream>

```

```
3  using namespace std;
4
5
6  int main() {
7     int n = 0;
8     cin >> n;
9  }
10
```

Przedstawiony algorytm nie wymaga do swojego działania tablicy o długości n. Już na wstępie do omawiania algorytmu założyliśmy wykreślenie wszystkich liczb parzystych większych od 2 – potrzebujemy tylko połowy komórek, które użylibyśmy w przypadku niezoptymalizowanego algorytmu sita Eratostenesa.

Sprawdzamy, czy wprowadzona przez użytkownika liczba n jest nieparzysta. Jeżeli badany warunek jest prawdziwy, wówczas zwiększamy wartość n o 1.

Tworzymy zmienną pomocniczą `polowa`, którą wykorzystamy do tworzenia tablicy, a także jako warunek wykonania kolejnych pętli.

```
1  #include <iostream>
2
3  using namespace std;
4
5
6  int main() {
7     int n = 0;
8     cin >> n;
9
10    if(n % 2 == 1){
11        n += 1;
12    }
13
14    int polowa = n / 2;
15 }
```

```
16
```

Następnie tworzymy tablicę wartości logicznych `tab` o długości `polowa`. Wypełniamy ją wartościami `true`. Dlaczego w komórkach tablicy przechowujemy wartości logiczne, skoro podczas omawiania algorytmu przez cały czas pokazywaliśmy liczby naturalne?

Zauważmy, że dzięki wyprowadzeniu korelacji między i – indeksem komórki tablicy oraz k – wartością przechowywaną w komórce, nie musimy zapisywać informacji o liczbie kryjącej się pod indeksem. Podobnie jak w przypadku niezoptymalizowanego algorytmu sita Eratostenesa, jeżeli komórka kryjąca się pod indeksem `i` przechowuje wartość `false`, oznacza to wykreślenie liczby `k`. Co więcej, sam algorytm opiera się na badaniu odległości między kolejnymi indeksami zapisanymi w tablicy – same liczby są wyłącznie dodatkową reprezentacją.

```
#include <iostream>

using namespace std;


int main() {
   int n = 0;
   cin >> n;

   if(n % 2 == 1){
       n += 1;
   }

   int polowa = n / 2;
   bool tab[polowa];

   for(int x = 0; x < polowa; x++){
       tab[x] = true;
   }
}
```

Dodajemy kolejne zmienne:

- `i` – będzie wskazywać akualnie przetwarzany indeks tablicy `tab`,
- `d` – będzie wskazywać odległość między kolejnymi wykreślanymi indeksami,
- `a` – będzie przechowywać pierwszą wielokrotność liczby kryjącej się pod indeksem `i`.

Zmienną `d` inicjalizujemy wartością 3, z kolei zmienną `a` wartością 4. Są to wartości początkowe wcześniej przedstawionego wzoru rekurencyjnego.

```
#include <iostream>

using namespace std;


int main() {
   int n = 0;
   cin >> n;

   if(n % 2 == 1){
       n += 1;
   }

   int polowa = n / 2;
   bool tab[polowa];

   for(int x = 0; x < polowa; x++){
       tab[x] = true;
   }

   int i = 1;
   int d = 3;
   int a = 4;
}
```

Dodajemy pierwszą pętlę – `while`. Będzie się wykonywać, dopóki nie przekroczymy badanego zakresu. W jej wnętrzu sprawdzamy, czy komórka kryjąca się pod indeksem `i` przechowuje wartość `true` – oznacza to, że badana liczba nie została wykreślona wcześniej, czyli jest pierwsza.

```
#include <iostream>

using namespace std;


int main() {
    int n = 0;
    cin >> n;

    if(n % 2 == 1){
        n += 1;
    }

    int polowa = n / 2;
    bool tab[polowa];

    for(int x = 0; x < polowa; x++){
        tab[x] = true;
    }

    int i = 1;
    int d = 3;
    int a = 4;

    while(a < polowa){
        if(tab[i] == true){

        }
    }
}
```

Jeżeli liczba kryjąca się pod indeksem `i` jest pierwsza, czyli przedstawiony warunek jest prawdziwy, musimy wykreślić jej wielokrotności. Tworzymy zmienną pomocniczą `z`, do której przypisujemy indeks pierwszej wielokrotności liczby kryjącej się pod indeksem `i`. Następnie w zagnieżdżonej pętli `while` ustawiamy wartość `false` kolejnym indeksom oddalonym o `d`.

```
#include <iostream>

using namespace std;


int main() {
   int n = 0;
   cin >> n;

   if(n % 2 == 1){
       n += 1;
   }

   int polowa = n / 2;
   bool tab[polowa];

   for(int x = 0; x < polowa; x++){
       tab[x] = true;
   }

   int i = 1;
   int d = 3;
   int a = 4;

   while(a < polowa){
       if(tab[i] == true){
           int z = a;
           while(z < polowa){
               tab[z] = false;
```

```
                z += d;
            }
        }
    }
}
```

Poza instrukcją warunkową zwiększamy wartość iteratora `i`. Zgodnie z przedstawionym wcześniej wzorem wyznaczamy nowe wartości zmiennych `d` oraz `a`.

```
#include <iostream>

using namespace std;


int main() {
    int n = 0;
    cin >> n;

    if(n % 2 == 1){
        n += 1;
    }

    int polowa = n / 2;
    bool tab[polowa];

    for(int x = 0; x < polowa; x++){
        tab[x] = true;
    }

    int i = 1;
    int d = 3;
    int a = 4;

    while(a < polowa){
        if(tab[i] == true){
```

```
            int z = a;
            while(z < polowa){
                tab[z] = false;
                z += d;
            }
        }
        i++;
        d += 2;
        a = a + 2 * (d - 1);
    }
}
```

Implementacje algorytmu kończymy, wypisując wszystkie niewykreślone liczby pierwsze, zaczynając od liczby 2. Wewnątrz pętli `for` sprawdzamy, czy wartość zapisana pod indeksem `i` przechowuje wartość `true`. Jeżeli warunek jest prawdziwy, wówczas liczymy wartość zmiennej `k` zgodnie z wcześniej przedstawionym wzorem i drukujemy na standardowym wyjściu.

```
#include <iostream>

using namespace std;


int main() {
    int n = 0;
    cin >> n;

    if(n % 2 == 1){
        n += 1;
    }

    int polowa = n / 2;
    bool tab[polowa];

    for(int x = 0; x < polowa; x++){
```

```
18          tab[x] = true;
19      }
20
21      int i = 1;
22      int d = 3;
23      int a = 4;
24
25      while(a < polowa){
26          if(tab[i] == true){
27              int z = a;
28              while(z < polowa){
29                  tab[z] = false;
30                  z += d;
31              }
32          }
33          i++;
34          d += 2;
35          a = a + 2 * (d - 1);
36      }
37
38      cout << 2 << " ";
39      for(int i = 1; i < polowa; i++){
40          if(tab[i] == true){
41              int k = 2 * i + 1;
42              cout << k << " ";
43          }
44      }
45  }
```

Przykład działania kodu dla $n = 100$.

Słownik

> **liczba pierwsza**
>
> liczba naturalna większa od 1, która dzieli się tylko przez jeden i przez samą siebie

sito Eratostenesa

algorytm, który przesiewa liczby z określonego zakresu w taki sposób, że zostają w nim tylko liczby pierwsze; służy on do znajdywania wszystkich liczb pierwszych w podanym przedziale

wielokrotność liczby

wielokrotność liczby a to taka liczba b, która powstaje przez pomnożenie liczby a przez liczbę naturalną n

Sprawdź się

Pokaż ćwiczenia: 

Ćwiczenie 1

Piotr Bajtocki prowadzi firmę budowlaną. Zgodnie z prawem ma obowiązek zaopatrzyć swoich pracowników w wodę. Butelki wody o pojemności 1,5 litra są pakowane w zgrzewkach.
W każdej ze zgrzewek zapakowano pewną liczbę butelek. Liczba ta jest liczbą pierwszą.
Najmniejsza zgrzewka opakowuje `m` butelek; największa `n`. Litr wody kosztuje `x` złotych.
Budżet Piotra wynosi `y` złotych. Jak dużą zgrzewkę wody może kupić?

Jeśli liczba butelek, które w zgrzewce może kupić pan Bajtocki, jest większa od `n`, oznacza to, że jego budżet pozwala na zakup największej zgrzewki.

W swoim rozwiązaniu nie wykorzystuj funkcji `sqrt`.

Swoje rozwiązanie przetestuj dla `x` wynoszącego 0,80; `y` równego 70, `n` równego 97, `m` wynoszącego 5.

Specyfikacja problemu:

Dane:

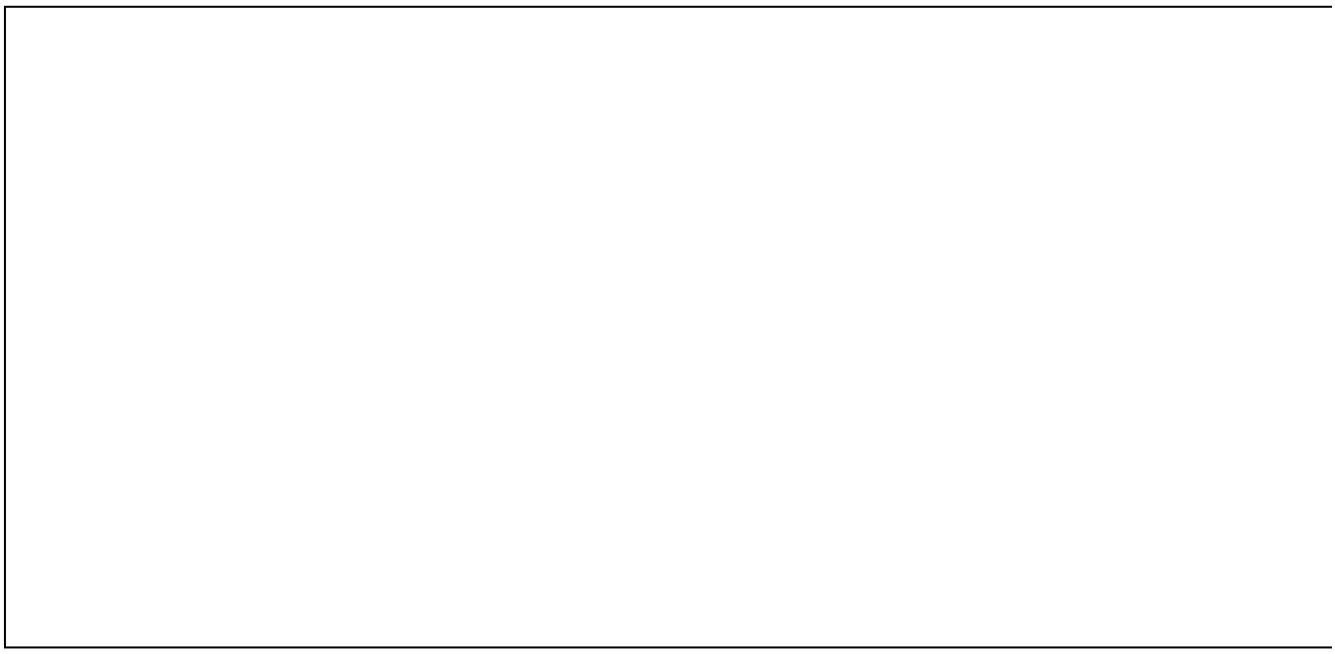
- `x` – cena litra wody; liczba zmiennoprzecinkowa dodatnia
- `y` – budżet Piotra; liczba zmiennoprzecinkowa dodatnia
- `n` – maksymalna liczba butelek w zgrzewce; liczba pierwsza; $n > 11$
- `m` – liczba butelek w najmniejszej zgrzewce; liczba pierwsza; $m \geq 5$

Wynik:

Na standardowym wyjściu program wypisuje największą liczbę butelek w zgrzewce, która mieści się w budżecie Piotra.

Twoje zadania

1. Program wypisuje największą liczbę butelek w zgrzewce, która mieści się w budżecie Piotra.



Ćwiczenie 2

Zygmunt Bajtek otrzymał od rodziców prezent – zegarek analogowy. Zafascynowany podarkiem postanowił policzyć, ile razy wskazówka minutowa wskaże liczbę pierwszą. Uzyskawszy tę wiedzę, policzył kąty ϕ między osią biegnącą od środka tarczy do minutowego 0, a wskazówką minutową wskazującą liczby pierwsze (przykład na rysunku). Napisz program, który w kolejnych liniach wypisze kolejne wartości ϕ, zaczynając od trzeciej minuty.

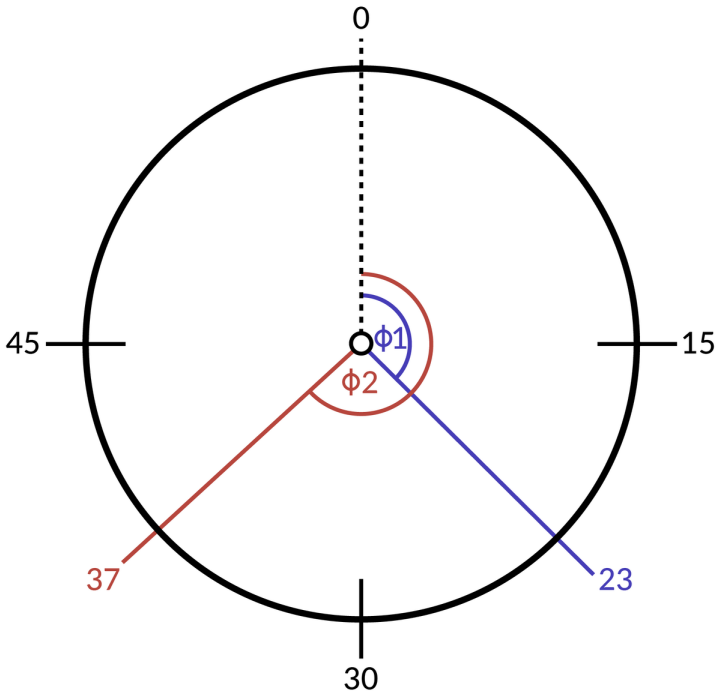
Specyfikacja problemu:

Dane:

- `A` – 61-elementowa tablica wartości logicznych wypełniona wartościami true

Wynik:

Na standardowym wyjściu program wypisuje w kolejnych wierszach wartości kąta ϕ.



Źródło: Contentplus.pl sp. z o.o., licencja: CC BY-SA 3.0.

Twoje zadania

1. Program wypisuje kolejne wartości kąta fi.

Ćwiczenie 3

Hipoteza Goldbacha zakłada, że każda liczba naturalna parzysta większa od 2 jest sumą dwóch liczb pierwszych. Napisz program weryfikujący hipotezę Goldbacha dla liczb parzystych nie większych od n.

Swoje rozwiązanie zweryfikuj dla n wynoszącego 1000.

Specyfikacja problemu:

Dane:

- n – parzysta liczba naturalna dodatnia

Wynik:

Na standardowym wyjściu program wypisuje równania potwierdzające hipotezę Goldbacha, zgodnie z poniższym schematem; `NIE` w przeciwnym przypadku.

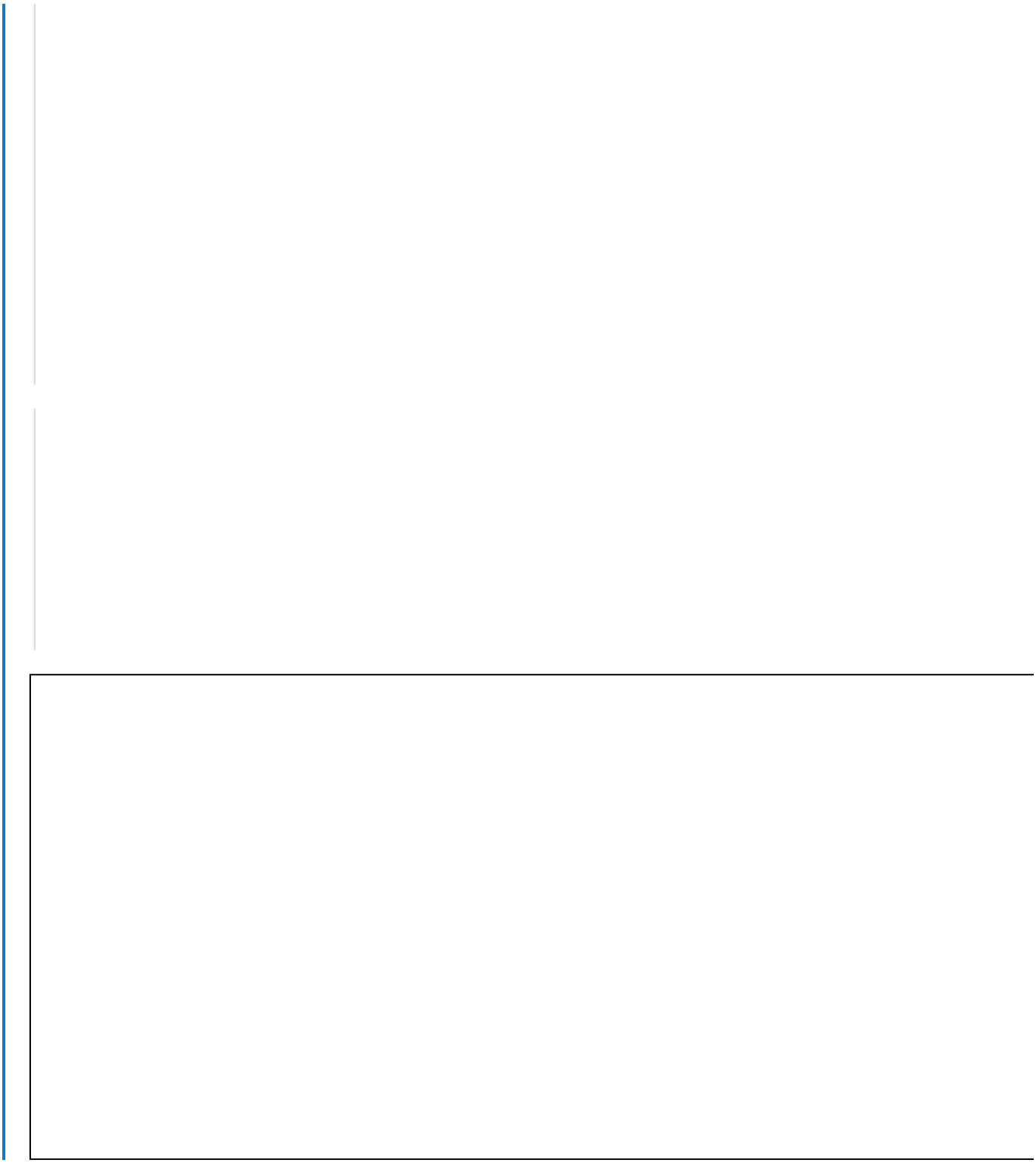
Przykład wyjścia:

Lewy składnik dodawania jest mniejszy od prawego; kolejne sumy wypisywane są rosnąco:

```
2 + 2 = 4
3 + 3 = 6
3 + 5 = 8
...
3 + 997 = 1000
```

Twoje zadania

1. Program wypisuje kolejne równania potwierdzające hipotezę Goldbacha; słowo NIE w przeciwnym przypadku.



Dla nauczyciela

Autor: Maurycy Gast
Przedmiot: Informatyka

Temat: Sito Eratostenesa 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 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:

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

c) generowania liczb pierwszych metodą sita Eratostenesa,

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):

- Przeanalizujesz zasadę działania algorytmu sita Eratostenesa.
- Zaimplementujesz algorytm Eratostenesa w języku C++.
- Rozwiążesz kilka problemów wymagających wyznaczenia liczb pierwszych z zadanego przedziału <2, n>.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

- dyskusja;
- 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: „Sito Eratostenesa w języku C++”. Uczniowie mają zapoznać się z treściami w sekcji „Film samouczek”.

Faza wstępna:

1. Przedstawienie tematu zajęć oraz wspólne z uczniami ustalenie kryteriów sukcesu.
2. **Rozpoznanie wiedzy uczniów.** Nauczyciel wyświetla na tablicy pytania zawarte w sekcji „Wprowadzenie":
   – do czego służy algorytm zwany sitem Eratostenesa?
   – jaka liczbę nazywamy liczba pierwszą?
   Chętni uczniowie udzielają na nie odpowiedzi.

Faza realizacyjna:

1. **Praca z multimedium.** Uczniowie zapoznają się z poleceniem 1 z sekcji „Film samouczek". Pracując w parach przygotowują program, następnie porównują swoje rozwiązanie z przedstawionym na filmie.
2. **Praca z tekstem.** Uczniowie analizują treści z sekcji „Przeczytaj" wyświetlone na tablicy.
3. **Ćwiczenie umiejętności.** Uczniowie wykonują indywidualnie ćwiczenie nr 1 z sekcji „Sprawdź się" , a następnie porównują swoje odpowiedzi z kolegą lub koleżanką.
4. Ćwiczenia nr 2 i 3 z sekcji „Sprawdź się" uczniowie wykonują w grupach czteroosobowych, a następnie porównują swoje odpowiedzi z inną grupą.

Faza podsumowująca:

1. Nauczyciel prosi uczniów o podsumowanie zgromadzonej wiedzy w zakresie programowania w języku C++.

Praca domowa:

1. Uczniowie dodają do swoich rozwiązań z sekcji „Sprawdź się" komentarze tak, by nawet osoba, która nie zna programowania mogła zrozumieć zachodzące procesy.

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:

- Uczniowie mogą wykorzystać treści w sekcjach: „Film samouczek", „Przeczytaj", „Sprawdź się" jako materiał do lekcji powtórkowej.