6. Wyszukujemy i sumujemy

Do rozwiązania wielu problemów przydają się algorytmy, w których wielokrotnie trzeba powtarzać szereg obliczeń. Takie obliczenia, zwane cyklicznymi, potrzebne są np. w grafice komputerowej do uzyskania efektu cienia w scenie 3D. Komputer o mocy obliczeniowej liczonej w miliardach operacji na sekundę jest w stanie realizować tego typu zadania znacznie szybciej niż człowiek. Oprócz szybkości komputery mają jeszcze jedną zaletę: nie mylą się podczas wykonywania żmudnych, powtarzalnych czynności. Dlatego w tym temacie nauczysz się zapisywać w języku C++ proste algorytmy obliczeń cyklicznych z wykorzystaniem pętli.

Cele lekcji

- Zrozumiesz, na czym polega iteracyjne rozwiązywanie problemów.
- Nauczysz się tworzyć algorytmy zawierające instrukcje powtarzania.
- Napiszesz w języku C++ programy z użyciem pętli.

Warto wiedzieć

Słowo „iteracyjny" pochodzi od łacińskiego czasownika *iterare*, który znaczy „powtarzać".

Do tej pory każdy nasz program zakładał jednokrotne wykonanie instrukcji zapisanych w tekście programu. Napiszemy teraz programy, które będą wielokrotnie wykonywały ten sam fragment kodu. Będą one komputerowymi realizacjami algorytmów nazywanych iteracyjnymi, czyli takich, w których aby uzyskać określony efekt, powtarza się jakąś czynność wielokrotnie.

6.1. Szukamy największej liczby. Pętla `while`

Warto wiedzieć

Określenie jednostek, w których wyrażone są różne wielkości, jest dla programisty bardzo ważne, ponieważ musi on odpowiednio wybrać typ danych. Na przykład jeśli dokładność pomiaru skoku w dal wynosi 1 cm, to wyniki podawane w centymetrach będą liczbami całkowitymi. Jeśli jednak wyniki podajemy w metrach, mogą być ułamkami.

Rozważmy następujący problem: trener skoku w dal mierzy podczas treningu długości kolejnych skoków i wpisuje wyniki na bieżąco do programu komputerowego. Chcemy stworzyć program, który na zakończenie treningu wyświetli długość najdłuższego skoku. Zakładamy, że wyniki podawane są w metrach z dokładnością do 0,01 m, czyli 1 cm.

Rys. 6.1. Skok w dal

Nie wiemy, ile skoków lekkoatleta wykona podczas treningu. Nawet jeśli trener zaplanuje jakąś ich liczbę, to i tak niektóre skoki będą prawdopodobnie spalone (nieważne).

Program komputerowy trzeba więc napisać tak, aby trener mógł poinformować komputer o tym, że trening został zakończony. Jak to zrobić?

Jednym ze sposobów może być wpisanie przez trenera liczby 0, która zakończy ciąg podawanych liczb. Wtedy program komputerowy powinien wypisać długość najdłuższego skoku.

Dobra rada

Ciąg potraktuj jako układ liczb, w którym następują one w określonej kolejności.

Specyfikacja

Dane: ciąg liczb zapisanych z dokładnością do dwóch miejsc po przecinku zakończony liczbą 0.

Wynik: największa z podanych liczb.

Aby wykonać to zadanie, program musi porównać długości skoków. Komputer nie potrafi jednak jednocześnie porównywać wielu liczb, nawet gdyby miał je wszystkie zapisane w pamięci. Pracuje on sekwencyjnie i nie może np. objąć wzrokiem kilku liczb, aby wskazać największą. W jednej operacji może brać pod uwagę tylko dwie liczby. Pozostałe są niedostępne.

Na poniższym schemacie zaprezentowano, jaką strategię wyboru największej spośród czterech podanych liczb stosuje człowiek, a jaką – komputer.

Warto wiedzieć

W żargonie programistów specjalną wartość (np. 0), która wskazuje koniec ciągu wczytywanych do programu wartości, nazywa się wartownikiem (ang. *sentinel*). Wartość ta musi być unikatowa, aby nie można było jej pomylić z wartościami podanymi przez użytkownika programu.

Jak widzi komputer?

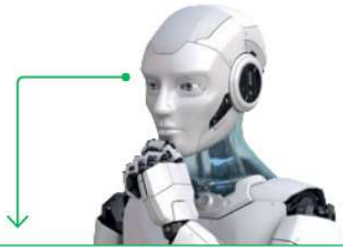
Zadanie:
Która z podanych liczb jest największa?

6,40	6,53	6,34	6,55

Człowiek

Przy małej liczbie danych człowiekowi wystarczy jedno spojrzenie, aby wskazać największą liczbę:

6,40	6,53	6,34	**6,55**



Komputer

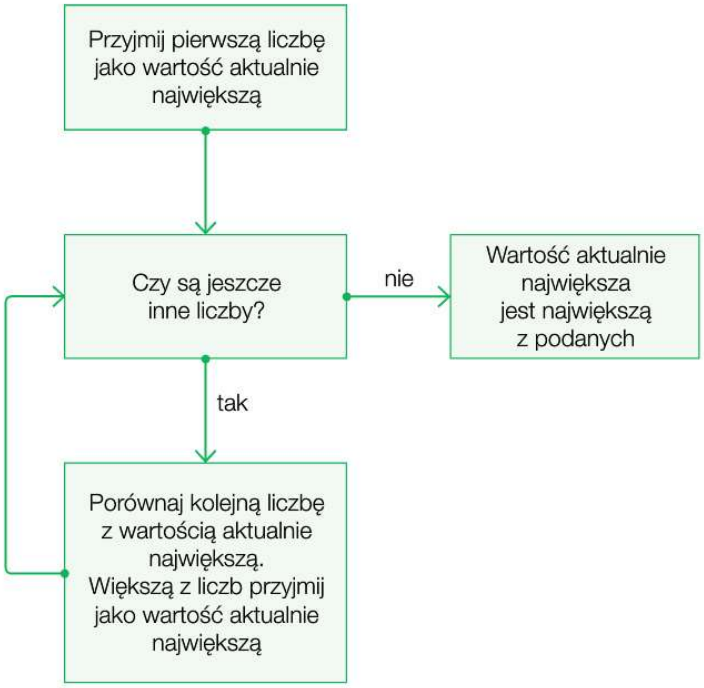
Komputer, nawet przy małej liczbie danych, etapami porównuje kolejne pary liczb i wskazuje większą liczbę z pary:

Etap 1	6,40	**6,53**		
Etap 2		**6,53**	6,34	
Etap 3		**6,53**		**6,55**

Aby program mógł wyznaczyć najdłuższy z podanych skoków, będziemy porównywać ich długości parami: porównamy pierwszy skok z drugim, dłuższy z nich porównamy z trzecim skokiem, dłuższy z nich z czwartym itd. Operacje te program będzie wykonywał do momentu, aż użytkownik wpisze liczbę 0.

Dzięki takiemu rozwiązaniu program będzie musiał zapamiętywać tylko dwie wartości: aktualnie podaną przez użytkownika i największą z dotychczas podanych. W naszej sytuacji pamiętanie pozostałych wartości nie jest potrzebne.

Poniższy diagram (rys. 6.2) pokazuje, na czym polega cykliczne wykonywanie tej samej operacji na kolejnych danych. Zauważ, że graficznie ma ono postać zbliżoną do pętli.



Rys. 6.2. Diagram przedstawiający algorytm wyszukiwania największej wartości

Lista kroków algorytmu wyznaczania największej liczby z ciągu liczb mogłaby wyglądać następująco:

1. `Jako wartość aktualnie największą (maksimum) przyjmij 0 i wczytaj pierwszą liczbę.`
2. `Dopóki ostatnia wczytana liczba jest różna od 0, wykonuj krok 3 i krok 4.`
   3. `Jeśli ostatnia wczytana liczba jest większa od maksimum, to podstaw ją za maksimum.`
   4. `Wczytaj kolejną liczbę z klawiatury.`
5. `Wypisz maksimum na ekranie.`

Warto wiedzieć

Procesor porównujący ze sobą dwie liczby używa sprzętowej instrukcji porównywania. Jej działanie polega na sprawdzeniu, czy różnica porównywanych liczb jest dodatnia, ujemna czy równa 0.

Przeanalizujemy teraz działanie algorytmu na przykładowych danych. Wygodnie będzie zrobić to w tabeli.

Wartość aktualnie największa (maksimum)	Wczytana liczba	Czy ostatnia wczytana liczba jest różna od 0?	Czy ostatnia wczytana liczba jest większa od aktualnego maksimum?
0	6,20	tak	tak
6,20	6,23	tak	tak
6,23	6,34	tak	tak
6,34	6,30	tak	nie
6,34	6,41	tak	tak
6,41	6,14	tak	nie
6,41	6,34	tak	nie
6,41	6,27	tak	nie
6,41	0	nie	–

Tabela 6.1. Działanie algorytmu wyznaczania największej liczby

Dobra rada

Jeżeli trudno ci zrozumieć, jak działa algorytm, możesz utworzyć tabelę zawierającą pośrednie wartości zmiennych i warunków logicznych.

Tabelę uzupełniamy wierszami od lewej do prawej, według kolejnych kroków algorytmu. Kolorem zielonym zaznaczono pewien etap analizy działania algorytmu: wczytano w tym momencie liczbę 6,41. Kolejnymi operacjami będą: sprawdzenie, czy liczba 6,41 jest różna od 0 (krok 2), a następnie – czy jest większa od 6,34 (krok 3). Ponieważ oba warunki są spełnione, liczba 6,41 staje się aktualnie największą. W kolejnych iteracjach podawane są liczby od niej mniejsze, dlatego w momencie wpisania liczby 0 liczba aktualnie największa staje się po prostu największą spośród wszystkich podanych liczb.

A to ciekawe

Muzyczne pętle

Pętlą lub loopem w muzyce elektronicznej nazywa się odpowiednio przygotowany fragment utworu (sampel), który wielokrotnie powtarza się w innym utworze. Jedną z najczęściej wykorzystywanych pętli perkusyjnych w muzyce rozrywkowej jest fragment z utworu Jamesa Browna *Funky Drummer*. Według serwisu whosampled.com wykorzystano go w ponad 1500 utworach innych artystów.

Ćwiczenie 1

Zastanówcie się w grupie, czy jeśli na liście kroków ze s. 114 zamienimy miejscami kroki 3 i 4, lista nadal będzie przedstawiać poprawny algorytm wyszukiwania wartości największej. W jaki sposób można to sprawdzić?

Zauważ, że program wyznaczający największą liczbę z ciągu powinien cyklicznie:

1. sprawdzać, czy liczba wprowadzona przez użytkownika jest liczbą 0, której podanie kończy działanie programu,
2. sprawdzać, czy wprowadzona liczba jest większa od aktualnie największej,
3. wczytywać kolejną wprowadzoną liczbę.

Instrukcja iteracyjna (instrukcja powtarzania, pętla)

W języku C++ operacje cykliczne realizuje się w specjalnych **instrukcjach iteracyjnych** (**powtarzania**), potocznie nazywanych **pętlami** (ang. *loop*). W językach programowania stosuje się różne rodzaje instrukcji iteracyjnych. W poniższym przykładzie wykorzystamy jeden z rodzajów pętli – **pętlę `while`**, która ma następującą budowę:

Pętla `while`

```
while (warunek) instrukcja;
```

Logika matematyczna i operatory logiczne, s. 252

Ta pętla pozwala wykonywać daną instrukcję (prostą lub złożoną), dopóki warunek jest spełniony, tzn. przyjmuje **wartość logiczną** prawda. Wartość logiczną wyrażenia program sprawdza przed każdym wykonaniem instrukcji, może więc: nie wykonać danej instrukcji, wykonać ją raz albo wiele razy.

Kod źródłowy programu *Najdłuższy skok* może wyglądać następująco:

Kod źródłowy programu *Najdłuższy skok*

```
1.  #include <iostream>
2.
3.  using namespace std;
4.
5.  int main()
6.  {
7.      float a,maks;
8.
9.      maks=0;
10.     cout << "Podaj kolejne dlugosci skokow" << endl;
11.     cin >> a;
12.
13.     while (a!=0)
14.     {
15.         if (a>maks) maks = a;
16.         cin >> a;
17.     }
```

Warto wiedzieć

Zmiennym zadeklarowanym w funkcji `main` kompilator nie przypisuje żadnej domyślnej wartości. Zmienna ma więc domyślnie wartość przypadkową.

```
18.
19.     cout << "\nNajdluzszy skok: " << maks << " m" << endl;
20.
21.     return 0;
22. }
```

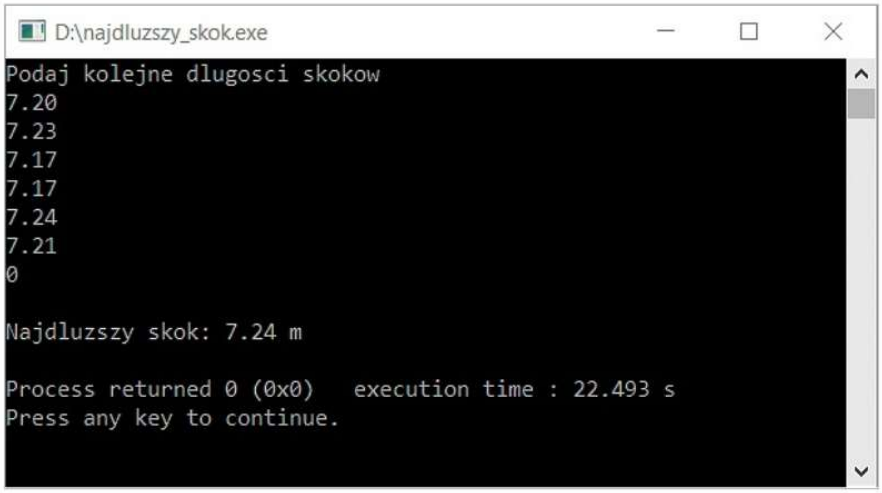
W wierszu 7 deklarujemy dwie zmienne typu **float** (a oraz maks), które przechowują liczby wymierne. W zmiennej a będzie przechowywana liczba jako ostatnia wczytana z klawiatury, a w zmiennej maks – aktualnie największa z wczytanych liczb. Na początku przypisujemy tej zmiennej wartość 0. W wierszu 11 wczytujemy wartość zmiennej a.

W wierszach 13–17 pojawia się pętla **while**, która najpierw sprawdza, czy wartość zmiennej a jest różna od 0. Jeżeli tak, to program wykonuje grupę instrukcji w nawiasach klamrowych. Instrukcje wykonywane są tyle razy, ile razy użytkownik podał liczbę różną od zera.

W wierszu 15 sprawdzamy, czy podana przez użytkownika liczba jest większa od wartości zmiennej maks. Jeśli tak, to do zmiennej maks przypisywana jest aktualna liczba. Po zakończeniu wprowadzania liczb (czyli po wpisaniu przez użytkownika liczby 0) aktualnie największa liczba staje się liczbą największą dla tego zestawu danych. Zauważ, że cała instrukcja **if** jest zapisana w jednym wierszu, ponieważ wykonywana jest w niej tylko jedna operacja.

W linii 19 użyto zapisu \n, który odpowiada za przeniesienie kursora do kolejnego wiersza. Zapis **cout** << "\n" można stosować wymiennie z zapisem **cout** << **endl**.

Okno programu *Najdłuższy skok* przedstawiono na rysunku 6.3. Zwróć uwagę, że w programie liczby ułamkowe podaje się z kropką zamiast przecinka.



Rys. 6.3. Wywołanie programu *najdluzszy_skok.exe*

Dobra rada

Jeżeli używasz w pętli **while** warunku porównującego zmienne typu **float**, staraj się stosować operatory porównania <, > i != zamiast <=, >= oraz ==. Pozwoli to uniknąć niektórych błędów wynikających z dokładności przybliżeń wartości zmiennych.

Dobra rada

We wnętrzu pętli może wystąpić również pojedyncza instrukcja. Wówczas nie musisz stosować nawiasów klamrowych.

Warto wiedzieć

Od lewego ukośnika zapisuje się tzw. stałe znakowe znaków niegraficznych kodu ASCII (np. \n dla nowego wiersza, \t dla tabulacji poziomej) lub znaków specjalnych (np. \" dla znaku ", gdy chcemy go wyświetlić na ekranie).

Warto wiedzieć

Zapisywanie liczb ułamkowych z kropką, a nie z przecinkiem, jest zwyczajem anglosaskim. To dlatego na klawiaturze numerycznej znajduje się właśnie znak kropki.

Ćwiczenie 2

Utwórz plik *najdluzszy_skok.cpp* i zapisz w nim kod źródłowy programu *Najdłuższy skok*. Skompiluj kod i uruchom program kilkukrotnie dla różnych danych wpisanych z klawiatury. Aby przyspieszyć wprowadzanie danych do programu, możesz podać je w jednym wierszu rozdzielone spacjami, np. `6.11 6.24 6 6.01 6.12 0`, a następnie nacisnąć **Enter**.

6.2. Sumujemy liczby. Pętla `for`

Pętla `for`

Drugą z instrukcji iteracyjnych w języku C++ jest **pętla `for`**. W odróżnieniu od instrukcji **`while`**, stosuje się ją w przypadkach, gdy można z góry określić liczbę cykli, czyli wykonań pętli, nawet jeśli dokładna liczba powtórzeń zależy od danych podanych przez użytkownika. Pętlę **`for`** wykorzystamy w kolejnym programie.

Problem, który sobie postawimy, to: wyznaczyć sumę n kolejnych liczb parzystych.

Specyfikacja

Dane: liczba naturalna n.

Wynik: suma n kolejnych liczb parzystych.

Aby rozwiązać problem od strony algorytmicznej, posłużymy się prostą matematyką: będziemy wielokrotnie dodawać kolejne składniki (kolejne liczby parzyste) do sumy częściowej, aż liczba dodanych składników będzie równa n. Zaczniemy od sumy równej 0 i składnika 2.

Sposobem na wielokrotne wykonanie instrukcji zwiększania sumy będzie instrukcja pętli. Do zapamiętania, ile razy została wykonana pętla, użyjemy **zmiennej sterującej** pętlą **`for`**, nazywanej też licznikiem.

Zmienna sterująca

Warto wiedzieć

W języku Scratch do tworzenia pętli służą bloki **powtarzaj aż** oraz **powtórz**, znajdujące się wśród bloków kontrolnych.

Składnia instrukcji **`for`** ma następującą postać:

```
for (licznik; warunek powtarzania; krok) instrukcja
```

W nawiasie po słowie **`for`** zapisuje się:

- instrukcję inicjującą, która nadaje wartość początkową zmiennej sterującej pętlą (licznik),
- wyrażenie testowe, czyli warunek logiczny, od którego zależy powtarzanie instrukcji znajdujących się w pętli,
- operację zwiększenia wartości zmiennej sterującej (ustalenie kroku pętli).

Jeśli powtarzana ma być instrukcja złożona, czyli składająca się z wielu instrukcji, to powinna być zapisana między nawiasami { i }.

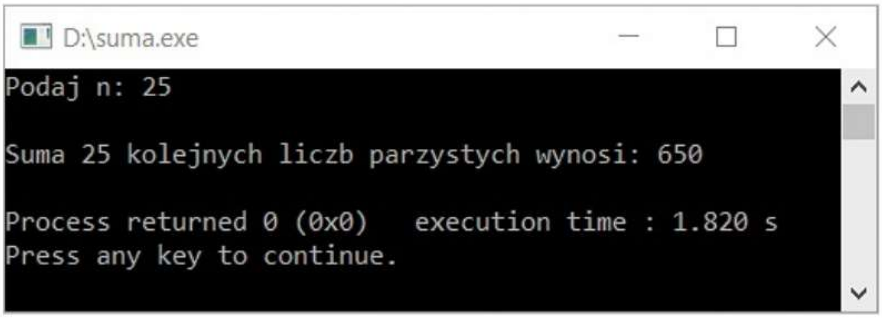
Kod źródłowy programu sumującego *n* kolejnych liczb parzystych może wyglądać następująco:

Kod źródłowy programu *Suma*

```
1.  #include <iostream>
2.
3.  using namespace std;
4.
5.  int main()
6.  {
7.      int n;
8.      cout << "Podaj n: "; cin>>n;
9.
10.     int i=1, suma=0, skladnik=2;
11.     for (i=1; i<=n; i++)
12.     {
13.         suma = suma + skladnik;
14.         skladnik = skladnik + 2;
15.     }
16.
17.     cout << "Suma " << n << " kolejnych liczb ";
18.     cout << "parzystych wynosi: " << suma <<  endl;
19.
20.     return 0;
21. }
```

Zwróć uwagę na wiersze 11–15, czyli na składnię instrukcji **for**. Podczas wykonywania pętli najpierw jest ustalana wartość początkowa zmiennej sterującej: i=1. Wartość zmiennej sterującej zwiększa się o 1 w każdym cyklu pętli do momentu osiągnięcia przez nią wartości zmiennej n. Aby zwiększyć wartość zmiennej typu **int** o 1, stosuje się zwykle specjalną składnię. Zamiast i=i+1 można napisać i++ (analogicznie zmniejszenie wartości o 1 można zapisać i-- zamiast i=i–1). Każde powtórzenie w pętli powoduje zwiększenie wartości zmiennej suma oraz ustalenie kolejnej wartości zmiennej skladnik (jako kolejnej liczby parzystej).

Okno programu *Suma* przedstawiono na rysunku 6.4.



Rys. 6.4. Wywołanie programu *suma.exe*

Warto wiedzieć

W języku C++ stosuje się również zapisy ++i oraz --i. Zapis ++i powoduje zwiększenie licznika zmiennej i jeszcze przed jej zastosowaniem, podczas gdy i++ powoduje zwiększenie licznika po jej zastosowaniu. Na przykład wartości i oraz a po wykonaniu kodu:

```
i=1;
a=i++;
```

będą odpowiednio równe: 2 i 1. Natomiast po wykonaniu kodu:

```
i=1;
a=++i;
```

będą odpowiednio równe: 2 i 2.
Analogicznie działają zapisy --i oraz i--.

Ćwiczenie 3

a. W pliku *suma.cpp* zapisz kod źródłowy programu *Suma*, skompiluj go i sprawdź działanie programu dla różnych wartości *n*.

b. Zmodyfikuj program *Suma* tak, aby wyznaczał sumę określonej liczby kolejnych liczb nieparzystych (począwszy od 1).

Warto wiedzieć

Suma kolejnych liczb nieparzystych jest kwadratem liczby składników:

$1 + 3 = 2^2$,

$1 + 3 + 5 = 3^2$,

$1 + 3 + 5 + 7 = 4^2$

itd.

Zapamiętaj

Pętla `while` jest wykonywana, dopóki jej warunek logiczny jest spełniony. Liczba powtórzeń nie jest określona z góry przez programistę.

Pętla `for` jest wykonywana określoną liczbę razy, ustaloną przez programistę lub przez użytkownika już po uruchomieniu programu.

6.3. Wyznaczamy noty sędziowskie

W ostatniej części tematu zmierzymy się z kolejną sytuacją problemową. Wykorzystamy umiejętności zdobyte wcześniej do opracowania odpowiedniego algorytmu i napiszemy program o bardziej rozbudowanej strukturze.

Rozważmy następujący problem: w konkursie skoków narciarskich chcemy wyznaczyć ocenę za styl, którą otrzyma dany zawodnik. Aby to zrobić, spośród pięciu ocen przyznanych przez pięciu sędziów należy odrzucić dwie skrajne noty: najwyższą i najniższą, a pozostałe zsumować.

Specyfikacja

Dane: pięć liczb z przedziału $[0; 20]$ podanych z dokładnością do 0,5.

Wynik: największa i najmniejsza liczba oraz suma liczb pomniejszona o liczbę najmniejszą i liczbę największą.

Warto wiedzieć

W statystyce stosuje się pojęcie średniej ucinanej. Średnia ta wyliczana jest na podstawie próbki wyników po odrzuceniu pewnej części wartości skrajnych – znacznie większych oraz znacznie mniejszych od pozostałych. W arkuszu kalkulacyjnym taką średnią wyznacza funkcja `ŚREDNIA.WEWN`.

Załóżmy, że nasz symulator wyznaczania oceny za styl będzie programem, który nie będzie zapamiętywał wszystkich pięciu liczb (not sędziowskich). Dzięki temu rozwiązanie stanie się bardziej uniwersalne: program będzie można łatwo zmodyfikować i zastosować do innego podobnego problemu, w którym stosuje się zasadę odrzucania wartości skrajnych (np. skrajnych wyników pomiarów w doświadczeniu przeprowadzanym na lekcji fizyki).

Algorytm będzie wyznaczał szukaną sumę przez odjęcie od sumy wszystkich pięciu liczb wartości najmniejszej oraz wartości największej. W przypadku gdy wartości skrajnych jest więcej, będziemy odejmować każdą tylko raz.

Punktacja w skokach narciarskich

Zawodnicy startujący w zawodach skoków narciarskich otrzymują za każdy skok ocenę, która składa się z dwóch części: punktów za uzyskaną odległość oraz noty za styl. Tę sumę koryguje się w zależności od warunków pogodowych i długości rozbiegu.

NOR	19.0
SUI	~~19.5~~
SVK	19.0
AUT	19.0
ITA	~~18.5~~
WIND:	+11.1
1	144.7

Noty sędziowskie za styl od 0 do 20 punktów: skrajne noty (najwyższa i najniższa) są usuwane, a pozostałe sumowane

Korekta punktacji ze względu na siłę i kierunek wiatru oraz długość rozbiegu

Suma punktów za odległość i styl z uwzględnieniem korekty

W zagadnieniu 6.1 analizowaliśmy przykład wyznaczania największej z liczb wprowadzonych z klawiatury. Zauważ, że tamto rozwiązanie można zaadaptować do obecnego problemu.

Oto propozycja listy kroków algorytmu:

1. Wczytaj pierwszą liczbę i przyjmij ją jako wartość minimum oraz jednocześnie wartość maksimum. Jako wartość sumy przyjmij wartość wczytanej liczby.
2. Powtórz cztery razy kroki od 3 do 6.
   3. Wczytaj kolejną liczbę.
   4. Zwiększ wartość sumy o wartość wczytanej liczby.
   5. Jeśli wczytana liczba jest mniejsza od minimum, to podstaw ją za minimum.
   6. Jeśli wczytana liczba jest większa od maksimum, to podstaw ją za maksimum.
7. Wypisz na ekranie minimum, maksimum oraz sumę pomniejszoną o minimum i maksimum.

Kod źródłowy programu może wyglądać następująco:

Kod źródłowy programu ● *Noty sędziowskie*

```
1.  #include <iostream>
2.
3.  using namespace std;
4.
5.  int main()
6.  {
7.      float nota,mini,maks,suma;
8.
9.      cout<<"Wprowadz noty za skok"<<endl;
10.     cin>>nota;
11.     mini=nota;
12.     maks=nota;
13.     suma=nota;
14.
15.     for (int i=1; i<=4; i++)
16.     {
17.         cin>>nota;
18.         suma=suma+nota;
19.         if (nota<mini) mini=nota;
20.         if (nota>maks) maks=nota;
21.     }
22.
23.     suma=suma-mini-maks;
24.     cout<<"\nNoty skrajne: "<<mini;
25.     cout<<" i "<<maks<<endl;
```

```
26.     cout<<"Nota za skok: "<<suma<<endl;
27.
28.     return 0;
29. }
```

W liniach 9–13 pobieramy pierwszą notę za skok i nadajemy tę wartość zmiennym `mini`, `maks` i `suma`. W linii 23 od sumy odejmujemy wartości najmniejszą i największą, aby uzyskać docelową notę.

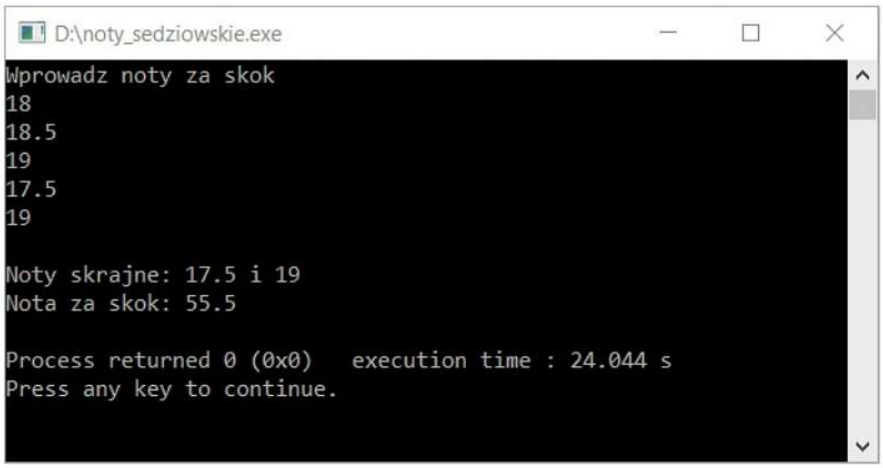
Kod programu wymaga jeszcze jednego dodatkowego wyjaśnienia: zwróć uwagę na wiersz 15. Zmienna `i`, czyli licznik pętli, została zadeklarowana dopiero w tym wierszu. Wielu programistów poleca taki sposób deklaracji zmiennych kontrolnych pętli, ponieważ:

- tak zadeklarowana zmienna ma tzw. mniejszy zasięg – poza kodem tworzącym daną pętlę **for** zmienna o takiej nazwie nie może być przypadkowo zastosowana (kompilator pokaże błąd),
- jest to wygodne – deklarujemy tę pomocniczą zmienną w momencie, gdy tworzymy kod dla pętli.

Okno programu *Noty sędziowskie* przedstawiono na rysunku 6.5.

Warto wiedzieć

W kodzie można użyć ponownie zmiennej o takiej samej nazwie, pod warunkiem że na nowo ją zadeklarujemy, np. w innej pętli **for**.



Rys. 6.5. Wywołanie programu *noty_sedziowskie.exe*

Ćwiczenie 4

a. Utwórz plik *noty_sedziowskie.cpp* i zapisz w nim kod źródłowy programu. Skompiluj kod i sprawdź działanie programu dla różnych zestawów danych.

b. Zmodyfikuj program tak, aby pozwalał obliczać sumę bez wartości skrajnych dla dowolnej liczby składników (wprowadzonych z klawiatury).

Podsumowanie

- Oprócz algorytmów liniowych i algorytmów z warunkami w programowaniu stosuje się algorytmy iteracyjne.
- Iteracyjne rozwiązywanie problemów polega na powtarzaniu jakiejś czynności skończoną liczbę razy, aż uzyska się oczekiwany efekt. Wykorzystuje się do tego instrukcje powtarzania (iteracyjne).
- W języku C++ instrukcjami powtarzania, pozwalającymi wielokrotnie wykonać ten sam fragment kodu, są `while` oraz `for`.
- Pętla `while` charakteryzuje się tym, że instrukcja wykonywana w pętli jest powtarzana, dopóki pewien warunek logiczny jest spełniony.
- W pętli `for` instrukcje wykonywane w pętli są powtarzane określoną liczbę razy.

Zadania

* **1** Napisz program, który wyznaczy najmniejszą liczbę z ciągu liczb całkowitych dodatnich podawanych z klawiatury. Przyjmij, że podanie liczby 0 kończy wprowadzanie liczb.

* **2** Napisz program wypisujący 10 kolejnych wielokrotności dodatniej liczby naturalnej wczytanej z klawiatury.

* **3** Zmodyfikuj kod źródłowy programu *Logowanie* z tematu 5 (s. 106) tak, aby program kończył działanie po trzech nieudanych próbach logowania.

* **4** Zmodyfikuj program *Najdłuższy skok* tak, aby trener mógł dodatkowo zliczać liczbę skoków nieważnych (spalonych).

* **5** Napisz program, który będzie obliczać średnią wartość (obciętą do części całkowitej) wszystkich liczb wprowadzonych z klawiatury, z pominięciem liczb najmniejszej i największej.

* **6** Przeanalizuj dokładnie działanie kodu źródłowego programu *Najdłuższy skok* na trzech konkretnych przykładach. Efekty przedstaw w tabeli w pliku tekstowym:

a	maks
4	4
2	
5	5
7	7
5	
0	

a	maks
8	
6	
2	
8	
9	
0	

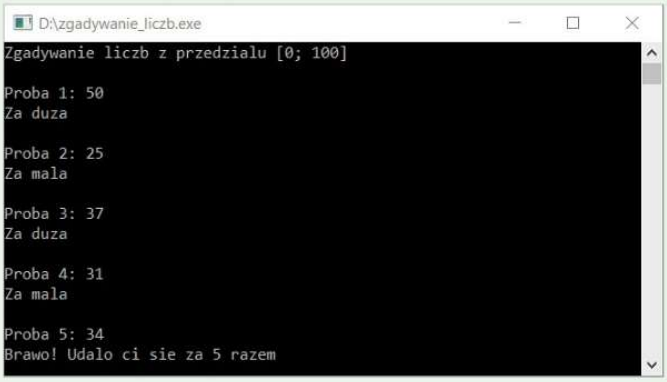
a	maks
12	
9	
12	
9	
12	
0	

** **7** Napisz program, który zliczy osobno, ile liczb ujemnych i liczb dodatnich wprowadzono z klawiatury. Przyjmij, że wczytanie liczby 0 kończy wczytywanie liczb.

** **8** Napisz program, który po wczytaniu 10 liczb wskaże te, które potraktowane jako długości boków utworzą prostokąt o największym polu.

** **9** Napisz kod programu, który pokaże różnicę w działaniu operatorów `++i` oraz `i++` dla zmiennej `i` typu całkowitego.

** **10** Napisz program, który będzie grą w zgadywanie ustalonej liczby z przedziału od 0 do 100. Program powinien działać do momentu podania przez użytkownika właściwej liczby. Po każdej próbie odgadnięcia liczby program powinien wypisać jeden z komunikatów: „Za mała", „Za duża" lub „Brawo! Udało ci się za *x* razem", gdzie *x* oznacza liczbę prób, które wykonał użytkownik (rysunek poniżej).



** **11** Napisz program, który będzie wczytywał podane przez użytkownika jedno- lub dwucyfrowe liczby całkowite i sumował je do momentu otrzymania sumy przekraczającej 123. Program powinien zwrócić informację o liczbie składników sumy i wartość sumy.

** **12** Napisz program, który wypisze pierwszych 100 liczb całkowitych dodatnich podzielnych przez:

a. 3,

b. 7,

c. 3 i 7.

*** **13** Zmodyfikuj program *Najdłuższy skok* tak, aby dodatkowo znajdował najdłuższą serię, w której każdy kolejny skok był dłuższy od poprzedniego, a jako wynik wyświetlał liczbę skoków w tej serii. Na przykład dla skoków o długościach 6,20; 6,30; 6,28; 6,33; 6,40; 6,30; 6,23 program powinien wyświetlić odpowiedź 3.

*** **14** Dla sześciu liczb wprowadzonych z klawiatury zawsze wystarczy 7 porównań, aby wyznaczyć najmniejszą oraz największą z nich. Jak to zrobić? Zapisz algorytm w postaci listy kroków, a następnie jego realizację w języku programowania.

*** **15** Napisz program, który pozwoli rozstrzygnąć, czy wśród *n* liczb całkowitych dodatnich wprowadzonych z klawiatury była taka trójka liczb *x*, *y* i *z*, dla których nie da się zbudować trójkąta o bokach długości *x*, *y*, *z*.

13. Czy ta liczba jest pierwsza?

Liczby pierwsze znasz z lekcji matematyki w szkole podstawowej. Choć odkryto je już w starożytności, przez lata twierdzenia dotyczące liczb pierwszych uchodziły wśród matematyków za piękne, ale bezużyteczne. Dopiero w drugiej połowie XX w. znalazły praktyczne zastosowania w informatyce, zwłaszcza w kryptografii. Twierdzenia te są podstawą działania niektórych protokołów szyfrowania, czyli algorytmów, które zapewniają poufność komunikacji w internecie i bezpieczeństwo transakcji bankowych. W tym temacie dowiesz się więcej na temat liczb pierwszych i ich znaczenia.

Cele lekcji

- Przypomnisz sobie i lepiej zrozumiesz zagadnienie podzielności liczb całkowitych.
- Poznasz algorytmy sprawdzające, czy liczba jest pierwsza.
- Nauczysz się stosować funkcje w języku Python.
- Zastosujesz funkcję `sqrt` z modułu `math`.

13.1. Liczby złożone i liczby pierwsze

Liczby złożone

Dodatnie liczby naturalne, które można przedstawić w postaci iloczynu liczb naturalnych mniejszych od nich, nazywamy **liczbami złożonymi**. Ich przykładami są: $9 = 3 \cdot 3$, $10 = 2 \cdot 5$, $120 = 20 \cdot 3 \cdot 4 \cdot 5$, $1115 = 3 \cdot 5 \cdot 7 \cdot 11$.

Liczby pierwsze

Pozostałe liczby, z wyjątkiem liczby 1, są **liczbami pierwszymi**. Liczbą pierwszą nazywamy więc liczbę, która jest podzielna wyłącznie przez 1 i samą siebie. Wśród liczb mniejszych od 100 jest dwadzieścia pięć liczb pierwszych:
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97.
Na rysunku 13.1 oznaczono je zielonym tłem.

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	

Rys. 13.1. Liczby złożone i liczby pierwsze mniejsze niż 100

Warto wiedzieć

Liczba 1 jest wyjątkowa. Przez setki lat uważano ją za liczbę pierwszą. Dziś przyjmuje się, że nie jest ani pierwsza, ani złożona.

Podstawowe twierdzenie arytmetyki

Jedno z najważniejszych twierdzeń matematycznych dotyczących liczb pierwszych nazywane jest **podstawowym twierdzeniem arytmetyki** i brzmi następująco: każda liczba naturalna (większa od 1) jest albo liczbą pierwszą, albo iloczynem liczb pierwszych.