

- **przestrzeń nazw** — blok chroniący swe identyfikatory przed przypadkowym użyciem w innym znaczeniu ■
- **std** — przestrzeń nazw chroniąca nazwy używane w standardowej bibliotece C ++ ■
- aby odwołać się do zmiennej wewnątrz przestrzeni nazw, piszemy nazwę przestrzeni nazw, parę dwukropków i nazwę zmiennej ■
- na przykład: **std::cout** ■

Zdarza się, że dwie biblioteki mają te same nazwy funkcji, program może nie zrozumieć o jaką funkcję chodzi, dlatego istnieje opcja przestrzeni, na początku przygody warto to ominąć używając std::

# Pierwszy program ponownie <sup>29</sup>

- Aby zaimportować całą nazwę z przestrzeni nazw, używamy dyrektywy

```
using namespace std;  <--- Polecane do użycia
```



```
#include <iostream.h>
using namespace std;
int main () {
    cout << "Our first program";  <-- Skraca pisanie
}
```

# Operacje arytmetyczne <sup>30</sup>

- Możemy dodawać, odejmować, mnożyć i dzielić liczby i zmienne za pomocą operatorów  $+$ ,  $-$ ,  $*$ ,  $/$ .■
- Aby zachować wynik, podstawiamy go do zmiennej za pomocą operatora  $=$

```
celsjus = (fahrenheit - 32) / 9 * 5;
```

Ukazanie wszystkich podstawowych operatorów w przyklad9.cpp

```
#include <iostream>
int main(){
    float celsjus , fahrenheit;
    cout << "temperature in Farenheit degrees: ";
    cin >> fahrenheit; // read a number
    celsjus = (fahrenheit - 32)/9*5; // convert
    cout << fahrenheit << "F = " << celsjus << "C\n";
}
```

**Exercise 0.2.** Napisz, skompiluj i uruchom program, który konwertuje temperaturę w stopniach Celsjusza na temperaturę w stopniach Fahrenheita

Przykład opisany w przyklad10.cpp

Exercise 0.2 napisany w przyklad11.cpp

**Exercise 0.3.** Napisz, skompiluj i uruchom program, który wczytuje trzy liczby zmiennoprzecinkowe i drukuje ich średnią.■

Ex. 0.3 napisany w przyklad12.cpp

**Exercise 0.4.** Napisz, skompiluj i uruchom program, który konwertuje upływ czasu podany w godzinach, minutach i sekundach na czas w samych sekundach.

Ex. 0.4 napisany w przyklad13.cpp

# Inkrementacja i dekrementacja <sup>33</sup>

- Wyrażenie  **$i+=d$**  jest skrótem dla  **$i=i+d$**  ■
- Wyrażenia  **$i++$**  i  **$++i$**  są skrótami dla zwiększenia o jeden:  **$i=i+1$**  ■
- Tak  **$i++$**  jak  **$++i$**  zwracają również wartość: pierwszy operator zwraca  **$i$**  przed inkrementacją, a drugi  **$i$**  po inkrementacji. ■
- Podobnie działają operatory zmniejszania:  **$i--$** ,  **$--i$** ,  **$i-=d$**

**Exercise 0.5.** Napisz, skompiluj i uruchom program, który wprowadza liczbę i wypisuje tę liczbę powiększoną o 1. Eksperymentuj z trzema różnymi, ale równoważnymi notacjami. ■

Ex. 0.5 zapisany w przyklad14.cpp

## Dzielenie całkowite i operator modulo <sup>35</sup>

```
int main(){  
    cout << "7/3= " << 7/3 << "\n";  
    cout << "7/3.= " << 7/3. << "\n";  
    cout << "7%3= " << 7%3 << "\n";  
    cout << "(-7)%3= " << (-7)%3 << "\n";  
}
```

7/3= 2  
7/3.= 2.33333  
7%3= 1  
(-7)%3= -1

Opisane w przyklad15.cpp



## Dzielenie całkowite i operator modulo <sup>36</sup>

- Wynikiem dzielenia dwóch liczb całkowitych jest zawsze liczba całkowita!
- Operator modulo % zwraca resztę po podzieleniu jednej liczby całkowitej przez drugą.

- Utworzenie tablicy

```
int age [12];    <--Numerowane od 0 do 11 !!
```

- Dostęp do elementów

```
age [0] = 2; age [1] = 5; age [2] = 7;
```

Numerowanie obiektów w tablicy zaczyna się od zera!

## Instrukcja **for** 38

- ```
for ( i=0; i < 12; i=i+1){  
    std::cout << "Podaj zarobki w miesiacu ";  
    std::cout << i+1 << ": ";  
    std::cin >> zarobki[i];  
}
```

**float** suma=0;  
**for** ( i=0; i < 12; i=i+1) suma=suma+zarobki[i];

Opisany w przyklad16.cpp

## • Funkcje <sup>39</sup>

```
float cubeVolume(float r){  
    return r*r*r;  
}
```

<-- () argumenty pozwalają wysłać do funkcji dane z funkcji która go wywołała  
<-- Return zwraca wartość podana do funkcji która go wywołała

- jeden argument typu **float** i zwracana wartość typu **float** ■
- struktura: **nagłówek** oraz **ciało funkcji** ■
- lista argumentów może być pusta ■
- **void** zamiast zwracanego typu: funkcja nic nie zwraca

```
int main(){  
    cout << "Enter the edge of the cube: ";  
    float edge;  
    cin >> edge;  
    cout << "The edge is ";  
    cout << edge << "\n";  
    cout << "The volume of the cube is ";  
    cout << cubeVolume(edge) << "\n";  
}
```

<--przyklad17.cpp

- Choć niektóre kompilatory na to pozwalają, użycie **void** jako typ zwracany przez funkcję **main** jest niezgodny ze standardem C / C ++ ■
- Wartość zwracana przez funkcję **main** jest używana przez system operacyjny do ustalenia czy wywołanie programu zakończyło się sukcesem (wartość zero) czy błędem (wówczas zwracany jest kod błędu). ■

**Exercise 0.6.** Napisz funkcję, która przyjmuje jako argument dwa boki prostokąta i zwraca pole prostokąta. Napisz, skompiluj i uruchom program główny, który demonstruje użycie tej funkcji. `<-- przyklad18.cpp`

- Math functions require the **cmath** library

```
#include <iostream>
```

Funkcje, które zawiera biblioteka cmath

```
#include <cmath>
```

```
using namespace std;
```

```
int main(){
```

```
    cout << "Square root of 2 is " << sqrt(2) << "\n";
```

```
    cout << "Absolute value of 2 is " << abs(2) << "\n";
```

```
    cout << "Natural log of 2 is " << log(2) << "\n";
```

```
    cout << "Sinus of 2 is " << sin(2) << "\n";
```

```
    cout << "Cosinus of 2 is " << cos(2) << "\n";
```

```
    cout << "Tangens of 2 is " << tan(2) << "\n";
```

```
}
```

Ex. 0.7 przyklad19.cpp

**Exercise 0.7.** Napisz funkcję **przekatna**, która przyjmuje jako argument dwa boki prostokąta i zwraca przekątną prostokąta. Napisz, skompiluj i uruchom program główny, który demonstruje użycie tej funkcji.

Pole trójkąta o bokach  $a, b, c$  można obliczyć według wzoru Herona

$$T = \sqrt{s(s-a)(s-b)(s-c)}$$

where

$$s = \frac{a + b + c}{2}.$$

**Exercise 0.8.** Napisz funkcję **poleTrojkata**, która przyjmuje jako argumenty trzy boki trójkąta i zwraca pole trójkąta. Napisz, skompiluj i uruchom program główny, który demonstruje użycie tej funkcji.

Ex 0.8 przyklad20.cpp



# Warunki<sub>3</sub>

- **warunek**: wyrażenie equiuowane do **true** albo **false** ■
- podstawowe przykłady:

$x < y$ ,  $x \leq y$ ,  $x > y$ ,  $x \geq y$ ,  $x == y$ ,  $x != y$

przyklad21.cpp

- warunki są używane w instrukcjach warunkowych ■
- każda niezerowa liczba traktowana jako warunek jest konwertowana na **true** ■
- zero traktowane jako warunek jest konwertowane na **false** ■

## Warunki<sub>4</sub>

- Automatyczna konwersja wartości liczbowych na warunki jest zachowana w C ++ dla kompatybilności wstecznej, ale może prowadzić do niebezpiecznych błędów
- dzieje się tak, gdy warunek  $x == y$  przez pomyłkę zapisano jako  $x=y$  ■
- W pierwszym przypadku mamy do czynienia z operatorem porównania który zwraca **true** wtedy i tylko wtedy gdy  $x$  jest równe  $y$ . ■
- W drugim przypadku mamy do czynienia z operatorem przypisania, który zwraca  $y$  skonwertowane do **true** gdy  $y$  jest niezerowe, a do **false** gdy  $y$  wynosi zero.

**Exercise 2.1.** Napisz program, który czyta ze standardowego wejścia liczby  $x$  i  $y$ , ewaluuje wyrażenia  $x == y$  i  $x = y$  i wypisuje wynik. Eksperymentuj z danymi wejściowymi, aby zobaczyć, jak różnią się dwa wyrażenia w zależności od wartości  $x$  i  $y$

**Exercise 2.2.** Rozszerz powyższy program, aby również ewaluował i drukował wyrażenia  $x < y$ ,  $x > y$ ,  $x <= y$ ,  $x >= y$  i  $x != y$ .

Ex. 2.2 przyklad22.cpp

# Instrukcja **if**<sub>6</sub>

```
#include <iostream>
int main(){
    std::cout << "Enter a positive number:";
    int number;
    std::cin >> number;
    if(number <= 0) std::cout << ". Wrong number.\n";
}
```

# Instrukcja **else**<sub>7</sub>

- wariant z **else**

```
if (number <= 0){  
    std::cout << ". This is a wrong number.\n";  
    std::cout << "Try again: ";  
    std::cin >> number;  
}else{  
    std::cout << "This is a positive number.\n";  
    std::cout << "Thank you\n";  
}
```

# Instrukcje i bloki<sub>8</sub>

- **Instrukcja** - pojedyncza instrukcja zakończona średnikiem
- **Blok** - sekwencja instrukcji ujęta w nawiasy klamrowe { ... }

```
{  
    std::cout << "Wrong number.\n";  
    std::cout << "Try again: ";  
    std::cin >> number;  
}
```

- blok nazywany jest również **instrukcją złożoną** ■

Nie umieszczamy średnika po zamknięciu nawiasu }!!!

# Bloki<sub>9</sub>

- Blok nie jest potrzebny w przypadku wystąpienia pojedynczej instrukcji **if** albo **else**

```
if (number < 0) std::cout << "Negative number.\n";  
else    std::cout << "Non-negative number.\n";
```



**Exercise 2.3.** Zmodyfikuj program, obliczający pole prostokąta, aby sprawdzał, czy wysokość i szerokość podane przez użytkownika są prawidłowe. Wydrukuj wiadomość informującą o złych danych, jeśli podane zostaną nieprawidłowe dane.■

Ex 2.3 przyklad23.cpp

**Exercise 2.4.** Wzór Herona na pole trójkąta może się nie powieść, jeśli podane liczby nie są bokami trójkąta. Zmodyfikuj program obliczający pole trójkąta tak, aby wykrywał, czy dostarczone dane są poprawne przed obliczeniem pola. Wydrukuj wiadomość informującą o złych danych, jeśli podane zostaną nieprawidłowe dane.■

Ex 2.4 przyklad24.cpp



# Bloki zagnieżdżone <sup>11</sup>

- Blok może być zagnieżdżony w innym bloku

```
if (a != 0){  
    if (a < 0){  
        std::cout << "a is negative.\n";  
    } else {  
        std::cout << "a is positive.\n";  
    }  
} else {  
    std::cout << "a is zero.\n";  
}
```

- 
- Zwiększamy czytelność poprzez głębsze wcięcie wewnętrznych bloków.

**Exercise 2.5.** Napisz program, który wypisze rozwiązanie równania kwadratowego  $ax^2 + bx + c = 0$  dla parametrów  $a, b, c$  odczytanych ze standardowego wejścia. Przeanalizuj przypadki, w których istnieją dwa, jedno lub brak rozwiązania.

Ex 2.5 przyklad25.cpp

**Exercise 2.6.** Napisz program, który wypisze rozwiązanie układu równań liniowych

$$ax + by = c$$

$$dx + ey = f$$

dla parametrów  $a, b, c, d, e, f$  odczytanych ze standardowego wejścia. Przeanalizuj przypadki, gdy istnieje dokładnie jedno rozwiązanie, brak rozwiązania lub nieskończenie wiele rozwiązań.

Ex. 2.6 przyklad26.cpp

**Exercise 2.7.** napisz program, który wypisze rozwiązanie nierówności

$$|x + a| > |3x + b|$$

dla parametrów  $a, b$  odczytanych ze standardowego wejścia. Przeanalizuj liczbę rozwiązań.

Ex 2.7 przyklad27.cpp

## operator wyrażenia warunkowego <sup>13</sup>

- używany w wyrażeniach: zamiast instrukcji warunkowej możemy użyć wyrażenia warunkowego ?: ■
- Instrukcja

```
if (u > 0) x=u; else x=0;
```

jest równoważne ewaluacji wyrażenia

```
x= (u > 0 ? u : 0);
```

**Exercise 2.8.** Napisz program, który wykorzystuje operator wyrażenia warunkowego do drukowania wartości bezwzględnej liczby odczytanej ze standardowego wejścia.■

Ex. 2.8 przyklad28.cpp

**Exercise 2.9.** Napisz program, który wykorzystuje operator wyrażenia warunkowego do wypisania maksimum i minimum dwóch liczb odczytanych ze standardowego wejścia.■

Ex 2.9 przyklad29.cpp

- instrukcja **switch**

```
switch(integerExpression) {  
    case e_1: statementList_1;  
    case e_2: statementList_2;  
  
    .....  
    case e_n: statementList_n;  
    default: statementList;  
}
```



- *e\_1, e\_2 ... e\_n* muszą być liczbami całkowitymi znanymi w czasie kompilacji ■

- The most common form

```
switch(integerExpression) {  
    case e_1: statementList_1; break;  
    case e_2: statementList_2; break;  
    .....  
    case e_n: statementList_n; break;  
    default: statementList;  
}
```

Gdyby nie było break po case po wykonaniu odpowiedniego case program zacząłby wykonywać inne case po danym case



**Exercise 2.10.** Napisz program, który wczyta ze standardowego wejścia numer dnia w tygodniu i użyje instrukcji wyboru do wydrukowania nazwy tego dnia. Wykorzystaj instrukcję **default** do poinformowania, że podany numer jest niepoprawny . ■

Ex. 2.10 przyklad30.cpp

**Exercise 2.11.** Napisz program, który wczyta ze standardowego wejścia numer miesiąca i użyje instrukcji wyboru do wydrukowania nazwy tego miesiąca. Wykorzystaj instrukcję **default** do poinformowania, że podany numer jest niepoprawny . ■

Ex 2.11 przyklad31.cpp

**Exercise 2.12.** Napisz program, który wczyta ze standardowego wejścia numer miesiąca i użyje instrukcji wyboru do wydrukowania liczby dni tego miesiąca. Wykorzystaj instrukcję **default** do poinformowania, że podany numer jest niepoprawny . ■

Ex 2.12 przyklad32.cpp

```
#include <iostream>
using namespace std;

int main(){
    for(int i=0;i<5;i++){
        cout << "Square of " << i;
        cout << " is " << i*i << "\n";
    }
}
```



## Drukowanie ciągu liczb <sup>20</sup>

**Exercise 2.13.** Napisz program, który drukuje pierwiastki kwadratowe liczb całkowitych od 1 do 10.■

Ex. 2.13 przyklad33.cpp

**Exercise 2.14.** Napisz program, który drukuje kwadraty liczb zmiennoprzecinkowych od  $a = 1.0$  do  $b = 2.0$  w krokach co  $d = 0.1$ .■

Ex. 2.14 przyklad34.cpp

**Exercise 2.15.** Zmodyfikuj powyższy program, aby użytkownik mógł podać  $a$ ,  $b$  i  $d$ .

Ex. 2.15 przyklad35.cpp

**Exercise 2.16.** Napisz program, który wypisuje elementy sekwencji  $\frac{1}{n^2+1}$  dla  $n = 1, 2, \dots, 7$ .■

Ex. 2.16 przyklad36.cpp

## Suma liczb naturalnych<sub>21</sub>

```
#include <iostream>
using namespace std;

int main(){
    int s=0;
    int n=1000;
    for(int i=1;i<=n;i++) s+=i;
    cout << "Sum of squares of integers from 1 to "
    cout << n << " is " << s << "\n";
}
```

## Drukowanie sumy ciągu liczb <sup>22</sup>

**Exercise 2.17.** Napisz program, który wypisze sumę kwadratów liczb całkowitych od 1 do 1000.■

Ex. 2.17 przyklad37.cpp

**Exercise 2.18.** Napisz program, który wypisze sumę pierwiastków kwadratowych liczb całkowitych od 1 do 1000.■

Ex. 2.18 przyklad38.cpp

**Exercise 2.19.** Napisz program, który wypisze iloczyn liczb całkowitych od 1 do 10.■

Ex. 2.19 przyklad39.cpp

**Exercise 2.20.** Napisz funkcję o nazwie silnia, która zwraca iloczyn liczb całkowitych od 1 do  $n$ , gdzie liczba  $n$  jest argumentem funkcji. Napisz program główny, który odczyta liczbę  $n$  ze standardowego wejście i użyje funkcji silnia do drukowania iloczynu liczb całkowitych od 1 do  $n$ .

Ex. 2.20 przyklad40.cpp

# Tablice <sup>23</sup>

- **Tablice** - zmienne, w których można przechowywać więcej niż jedną liczbę
- Aby zadeklarować tablicę piszemy:

```
int age[5];
```


- Aby wykorzystać elementy tablicy:

```
age[0]=2; age[1]=5; age[2]=7; age[3]=10; age[4]=14;
```

Numeracja obiektów w tablicy zaczyna się od zera!

## Czytanie tablic<sup>24</sup>

```
float wages[12];  
for(int i=0; i<12; i++){  
    cout << "Wages in the " << i+1 << "th month: ";  
    cin >> wages[i];  
}
```



```
float s=0;  
for(int i=0; i<12; i++) s+=wages[i];  
cout << "Your total income is " << s << "\n";
```

**Exercise 2.21.** Napisz program, który odczyta ciąg 10 liczb i wypisze średnią wartość oraz odchylenie standardowe liczb w tym ciągu.■

Ex. 2.21 przyklady41.cpp

**Exercise 2.22.** Zmodyfikuj program w poprzednim ćwiczeniu, aby mógł przetwarzać zbiór liczb o dowolnej wielkości. W tym celu najpierw przeczytaj liczbę liczb w kolekcji.■

Ex. 2.22 przyklady42.cpp

```
int main(){  
    const int length=30;  
    char text[length];  
    cout << "Enter a word: ";  
    gets(text);  
    cout << "You wrote: " << text << "\n";  
}
```

przyklad43.cpp

```
int main(){
    const int length=30;
    char text[length];
    cout << "Enter a word: ";
    int i;
    for(i=0;i<length-1;++i){
        char c;
        c=getchar();
        if(c <= ' ') break;
        text[i]=c;
    }
    text[i]=0;
    cout << "You wrote: " << text << "\n";
    cout << "Inverted: ";
    for(int j=i-1;j>=0;--j) cout << text[j];
    cout << "\n";
}
```



**Exercise 2.23.** Napisz program, który odczytuje wiersz tekstu ze standardowego wejścia i wypisuje liczbę cyfr w tym tekście.

Ex. 2.23 przyklad45.cpp

**Exercise 2.24.** Napisz program, który odczytuje wiersz tekstu ze standardowego wejścia i drukuje liczbę oddzielnych słów w tekście. Przez słowo rozumiemy ciąg znaków inny niż spacja.

Ex. 2.24 przyklad46.cpp

```
float x;  
cout << "Enter a  number: ";  
cin >> x;  
while(x>=1) —x;  
cout << "Fractional part of your number is: ";  
cout << x << "\n";
```

**Exercise 2.25.** Napisz program, który oblicza ułamkową część odczytanej liczby zmiennoprzecinkowej ze standardowego wejścia. Użyj pętli **while**.

Ex 2.25 przyklad47.cpp

## Instrukcja **do-while** 30

```
float x;  
do{  
    cout << "Give me a positive number: ";  
    cin >> x;  
}while(x<=0);  
cout << "Your positive number is: ";  
cout << x << "\n";
```

**Exercise 2.26.** Napisz program, który odczytuje liczbę zmiennoprzecinkową  $x$  ze standardowego wejścia taką, że  $0 < x < 1$ . Użyj pętli do-while.

Ex. 2.26 przyklad48.cpp

**Exercise 2.27.** Napisz program, który odczytuje dodatnią liczbę zmiennoprzecinkową i oblicza pierwiastek kwadratowy z dobrą dokładnością. Użyj ciągu

$$\begin{cases} x_1 := 1, \\ x_{n+1} := \frac{x_n^2 + a}{2x_n} \end{cases}$$

który zbiega do  $\sqrt{a}$ . ■

Ex. 2.27 przykład49.cpp

## Instrukcja **break** 32

```
float x;  
while(true){  
    cout << "Give me a positive number: ";  
    cin >> x;  
    if(x>0) break;  
    cout << "This is not a positive number.\n";  
};  
cout << "Your positive number is: ";  
cout << x << "\n";
```

**Exercise 2.28.** Napisz program, który odczyta ciąg 10 liczb naturalnych i wydrukuje pozycję pierwszej liczby w ciągu, która dzieli ostatnią liczbę w ciągu. ■

Ex. 2.28 przyklad50.cpp

## Instrukcja **goto** 34

```
for( int i=0; i<12; i++)  
for( int j=i+1; j<12; j++){  
    if( wages[i]==wages[j]){  
        cout << "Wages are the same in months ";  
        cout << i+1 << " and " << j+1 << "\n" ;  
        goto done;  
    }  
}  
cout << "Each month the wages are different\n";  
done;;
```

przyklad51.cpp

**Exercise 2.29.** Napisz program, który odczytuje ciąg 10 liczb i odpowiada na pytanie, czy istnieje para różnych liczb w ciągu, taka że jedna liczba pary dzieli drugą.■

Ex. 2.29 przyklad52.cpp

**Exercise 2.30.** Napisz program, który odczytuje ciąg 10 liczb i odpowiada na pytanie, czy istnieje para różnych liczb w ciągu, taka że ich suma wynosi 10.■

Ex. 2.30 przyklad53.cpp



## Instrukcja **continue** 36

```
float limit=1500;
int cnt=0;
for(int i=0; i<12; i++){
    if(wages[i]>limit) continue;
    ++cnt;
}
cout << "Your wages did not exceed the limit";
cout << " in " << cnt << " months.\n";
```