

## **Wykład: 3**

### **Instrukcje sterujące, operatorzy**



## Instrukcje sterujące

### Pętla for - przykłady



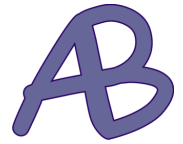
```
for ( instrukcja_ini ; wyrażenie_warunkowe ; instrukcja_krok )  
    tresc_petli ;
```

- **instrukcja\_ini** - instrukcja wykonywana zanim pętla zostanie poraz pierwszy uruchomiona,
- **wyrażenie\_warunkowe** – wyrażenie obliczane przed każdym obiegiem pętli. Jeżeli jest ono różne od zera, to pętla będzie dalej wykonywana,
- **instrukcja\_krok** – instrukcja wykonywana po zakończeniu każdego obiegu pętli,

**Przykład:**

Program wypisze n razy tekst „naucze się programować w c++”

```
int main()
{
    cout <<"Ile razy:";
    int ile;
    cin >> ile;
    for (int i=0 ; i < ile; i++)
        cout << i+1 <<": Naucze sie programowac w c++\n";
    return 0;
}
```

**Przykład:**

Program wypisuje znaki kodu ASCII wraz z ich numerami

```
int main()
{
    for (int i = 28; i < 256; i++ )
        cout << "[" << i << "]" << (char)i << " ";
    return 0;
}
```

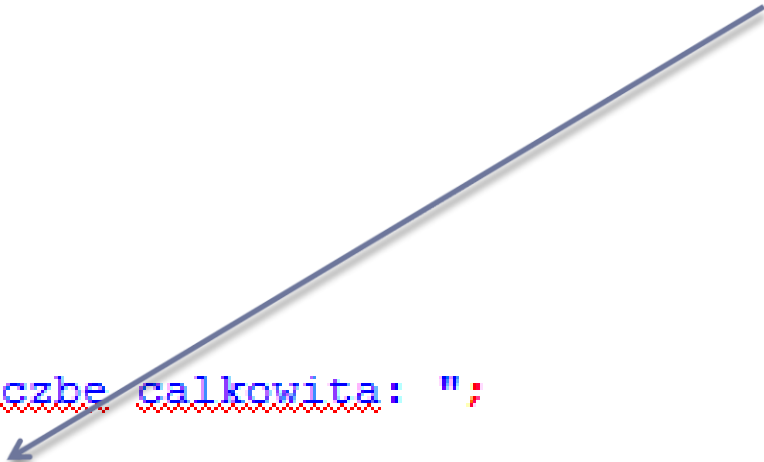
**Przykład:**

Program wylicza liczbę cyfr podanej liczby całkowitej.

```
#include <iostream>

using namespace std;
int main()
{
    long n;
    int c;
    cout << "Podaj liczbę całkowitą: ";
    cin >> n;
    for (c=1; n/=10; c++) ;
    cout << "Liczba cyfr wynosi: " << c << endl;
    return 0;
}
```

Pusta pętla for  
Pozornie „nic nie robi”



## Pętla for – pętla w pętli

**Przykład:** tabliczka mnożenia – przykład pętli zagnieżdżonej (pętla w pętli)

```
#include <iostream>

using namespace std;

int main()
{
    int i, j;
    for(i=1; i<=10; i++)
    {
        for(j=1; j<=10; j++)
        {
            cout << (i*j) << "\t";
        }
        cout << endl;
    }
    return 0;
}
```

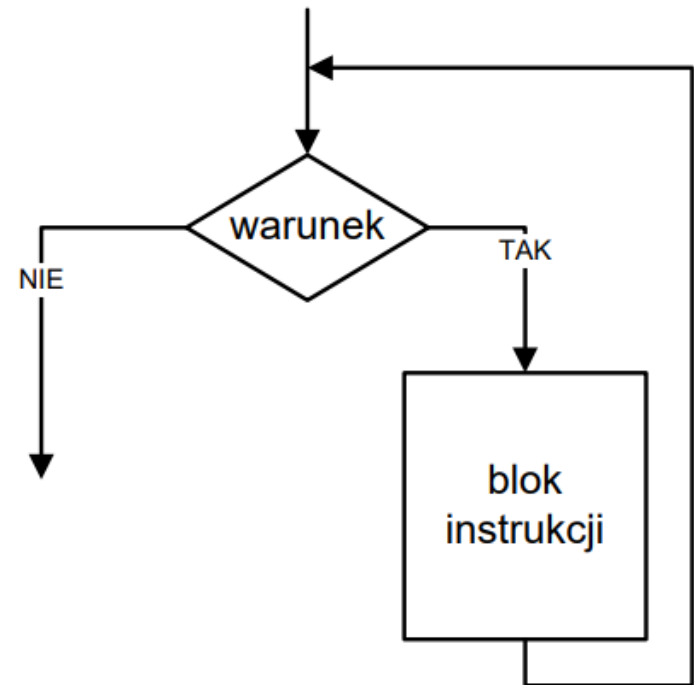


## Instrukcje sterujące

### Pętla while



```
while (warunek) instrukcja;  
.....  
while (warunek)  
{  
    instrukcja_1;  
    instrukcja_2;  
}
```



*// pętla wyświetlająca liczby 1, 2, 3 ...*

```
int i = 1;  
while( i <=10 ) cout << i++ << " , ";
```

**Przykład:**

pętla wypisująca liczby od 1 do 10

```
int i = 1;  
while( i <=10 ) cout << i++ << ", ";
```



## Instrukcje sterujące

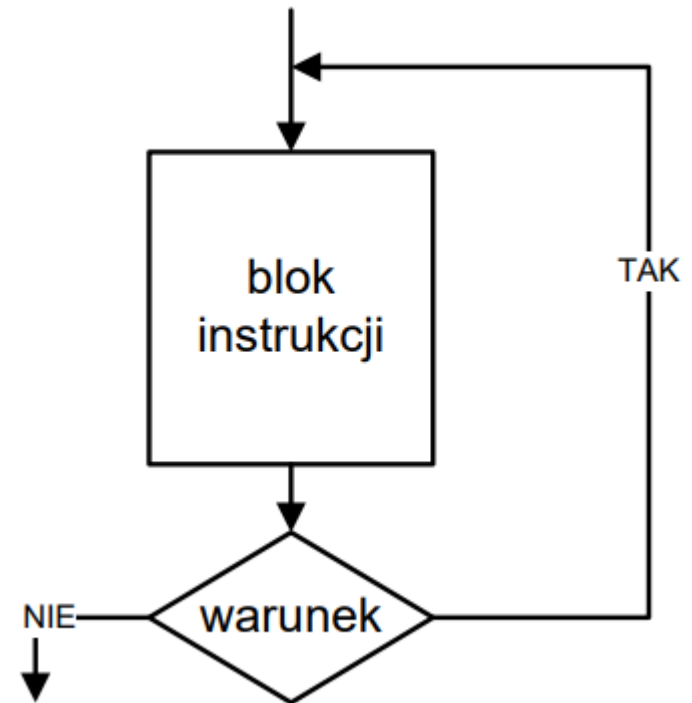
### Pętla while

## Pętla do while



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

```
do
{
    instrukcja_1;
    instrukcja_2;
}
while ( warunek );
```



## Pętla do while

---

### Przykład:

pętla wypisująca liczby od 1 do 10 (włącznie)

```
int i = 1;
do
{
    cout << i << ", ";
    i = i + 1;
}
while( i<=10);
```

## Instrukcje break i continue

**break** - kończy wykonywanie najbliższej otaczającej pętli lub instrukcji warunkowej, w której występuje. Jeśli po końcu przerwanej instrukcji występuje kolejna, sterowanie przechodzi do niej.

Przykład:

Pętla kończy się po wypisaniu  $i = 4$

```
for (int i = 1; i < 10; i++)  
{  
    cout << i << '\n';  
    if (i == 4)  
        break;  
}
```



Wyjście z pętli

## Instrukcje break i continue

---

Instrukcja **break** jest używana z instrukcją warunkową **switch**, a także z instrukcjami pętli **do**, **for** i **while**.

W instrukcji **switch**, instrukcja **break** powoduje, że program wykonuje kolejną instrukcję, która występuje po instrukcji **switch**. Bez instrukcji **break**, wykonywane są wszystkie instrukcje od dopasowanej etykiety **case** do końca instrukcji **switch**, łącznie z klauzulą **default**.

W pętlach, instrukcja **break** kończy wykonywanie najbliższej otaczającej instrukcji **do**, **for** lub **while**. Sterowanie przechodzi do instrukcji następującej po zakończonej, jeśli taka istnieje.

## lbreak - przykład

---

Program wczytuje i sumuje liczby, aż do podania pierwszej liczby ujemnej

```
int main()
{
    int x;
    int suma = 0;
    while(1)
    {
        cout << "x=";
        cin >> x;
        if (x < 0)
            break;
        suma = suma + x;
    }
    cout << "suma=" << suma;
    return 0;
}
```




## Instrukcja break i co

**continue** - wymusza przekazanie kontroli do wyrażenia kontrolującego najmniejszej pętli **do**, **for**, lub **while**.

Przykład:

Pętla wypisze liczby od 1 do 10 pomijając liczbę 5

```
for (int i = 1; i <= 10; i++)  
{  
    if (i == 5) continue;  
    cout << i << '\n';  
}
```



Powrót na górę pętli  
(z pominięciem instrukcji  
poniżej continue)

## Instrukcja break i continue

**continue** – przykład – program wczytuje 5 liczb naturalnych. Próby podania liczby ujemnej ignoruje

```
const int LICZBA = 5;
int main()
{
    int ile = 0, suma = 0, x;
    do
    {
        cout<<"x("<<ile+1<<" "= " <<endl;
        cin>>x;
        if(x<0) continue;
        suma +=x;
        ile++;
    } while (ile<LICZBA);
    cout<<"suma="<<suma<<endl;
    return 0;
}
```

# Instrukcje break i continue

## Przykład:

### Algorytm Euklidesa – wersja 2

```
#include <iostream>
using namespace std;
int main()
{
    unsigned int a, b;
    cout << "a = ";    cin >> a;
    cout << "b = ";    cin >> b;
    while (1)
    {
        if (a>b) a-=b;
        else if (a<b) b-=a;
        else break;
    }
    cout << "NWP(a,b) = " << a << endl;
    return 0;
}
```

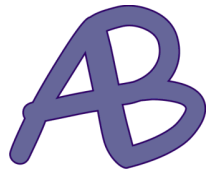
Pętla nieskończona

Wyjście z pętli

## Instrukcja return

**return** - kończy wykonywanie funkcji i zwraca sterowanie do funkcji wywołującej (lub do systemu operacyjnego, jeśli kontrola zostanie przeniesiona z funkcji main). Wykonanie wznowia działanie w funkcji wywołującej w punkcie bezpośrednio po wywołaniu.

```
int normalizuj (int a)
{
    if (a>0) return(a);
    else if (a<0) return (-a);
    else return (0);
}
```



## Operatorzy



## Operatory arytmetyczne:

$a = b + c$  ;      // dodawanie  
 $a = b - c$  ;      // odejmowanie  
 $a = b * c$  ;      // mnożenie  
 $a = b / c$  ;      // dzielenie  
 $a = b \% c$  ;      // modulo – reszta z dzielenia

Priorytet operatorów arytmetycznych jest taki sam, jak w matematyce. Czyli zapis:

**$a + b \% c * d - f$**

oznacza to samo co

**$a + (b \% c) * d - f$**



## Operatory arytmetyczne – zapis skrócony:

Dodawanie	$x = x + y;$	$x += y;$
Odejmowanie	$x = x - y;$	$x -= y;$
Mnożenie	$x = x * y;$	$x *= y;$
Dzielenie	$x = x / y;$	$x /= y;$
reszta z dzielenia	$x = x \% y;$	$x \% = y;$



## Operatory inkrementacji i dekrementacji

**i++** ; // inkrementacja - oznacza to:  $i = i + 1$

**i--** ; // dekrementacja – oznacza to:  $i = i - 1$

Operatory te mogą mieć dwie formy:

- **Prefix: ++a** (przed argumentem) - najpierw zmienna jest zwiększana o 1, następnie ta zwiększona wartość staje się wartością wyrażenia.
- **Postfix: a++** (po argumentem) - najpierw brana jest stara wartość zmiennej i ona staje się wartością wyrażenia, a następnie zwiększany jest on o 1





## Operator przypisania

Każde przypisanie samo w sobie jest także wyrażeniem mającym taką wartość, jaka jest przypisywana.

Zatem wartość wyrażenia  $(x = 2)$  jako całości jest równa 2



## Operatory relacji

<	mniejszy
<=	mniejszy lub równy
>	większy
>=	większy lub równy
==	czy równy
!=	czy różny

### Uwaga na błąd:

```
int a = 10;  
int b = 20;  
if (a=b) ..... // zamiast if (a==b)...
```

Nie jest błędem, tyle że oznacza

```
if (20) ..... // ogólnie if (b != 0) .....
```



## Operatory sumy logicznej i iloczynu

- ||** - sumę logiczną - operację logiczną LUB (alternatywa)
- &&** - iloczyn logiczny - czyli operację I (koniunkcja)

„prawda” daje rezultat 1, a wynik „fałsz” daje rezultat 0

Wyrażenia logiczne tego typu obliczane są od lewej do prawej. Kompilator oblicza wartość wyrażenia dotąd, dopóki na pewno nie wie jaki będzie wynik.

Np.: jeżeli w wyrażeniu

`(a == 0) && (b != 10) && (c > 100)`

A będzie różne od zero, kolejne porównania nie będą wyliczane

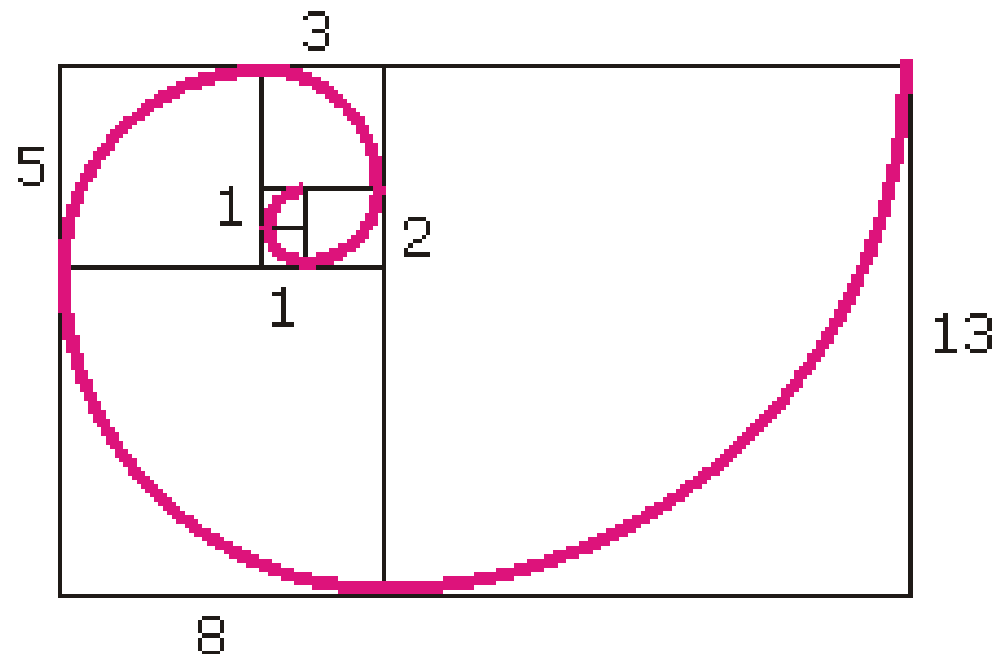


# Przykłady algorytmów

## Ciąg Fibonacciego

**Ciąg Fibonacciego** to ciąg liczb, w którym:

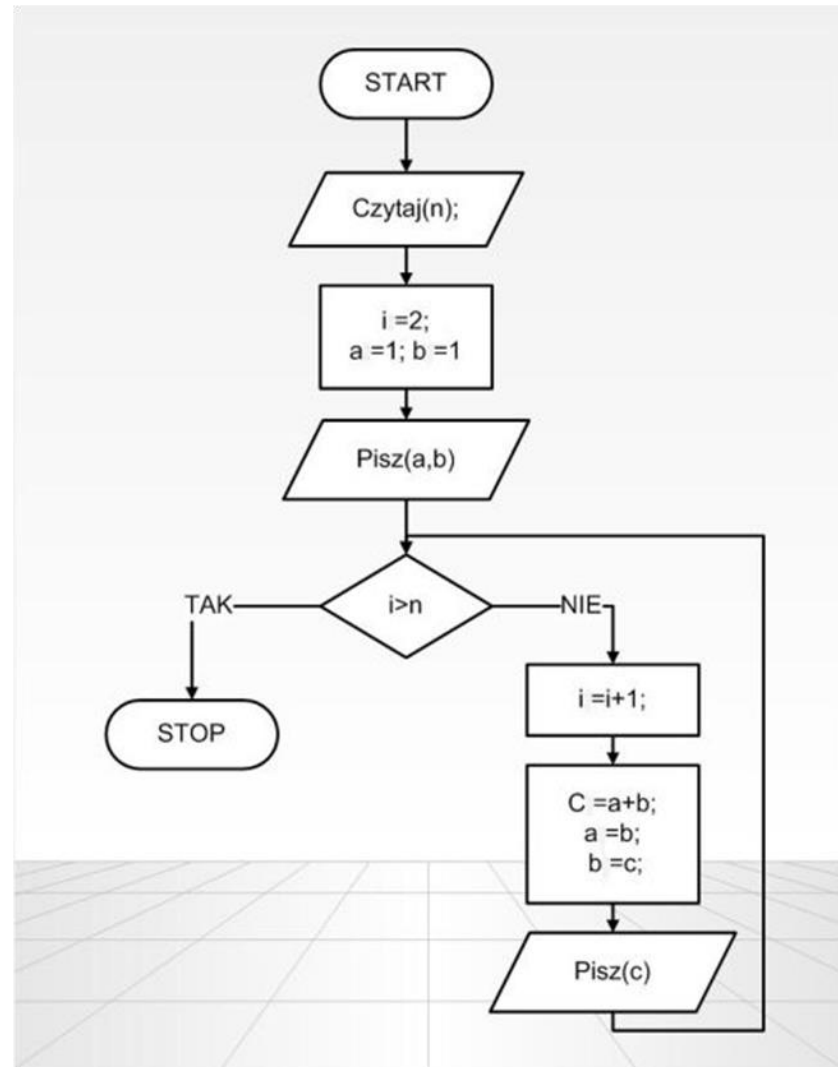
- pierwszy wyraz jest równy **1**,
- drugi jest równy **1**,
- każdy następny jest **sumą dwóch poprzednich**.



## Ciąg Fibonacciego



- pierwszy wyraz jest równy **1**,
- drugi jest równy **1**
- każdy następny jest **sumą dwóch poprzednich**.





```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int n, a1=1, a2=1, temp;
6
7      cout << "Ile elementow wyliczyc? ";
8      cin >> n;
9      if(n<=1)
10         cout << a1 << "; ";
11     else if (n<=2)
12         cout << a1 << "; " << a2 << "; ";
13     else
14     {
15         cout << a1 << "; " << a2 << "; ";
16         for (int i=3; i<=n; i++)
17         {
18             temp = a1 + a2;
19             a1 = a2;
20             a2=temp;
21             cout << a2 << "; ";
22         }
23     }
24     return 0;
25 }
```

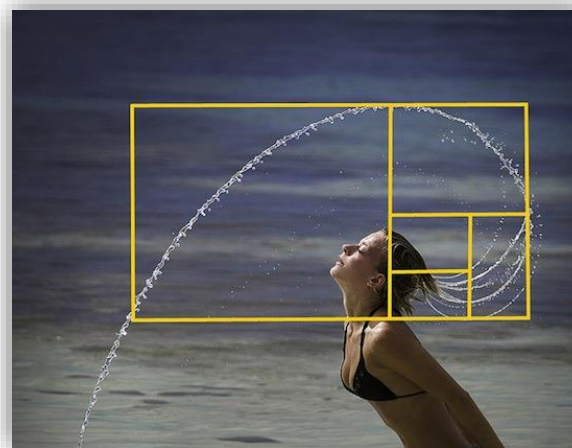
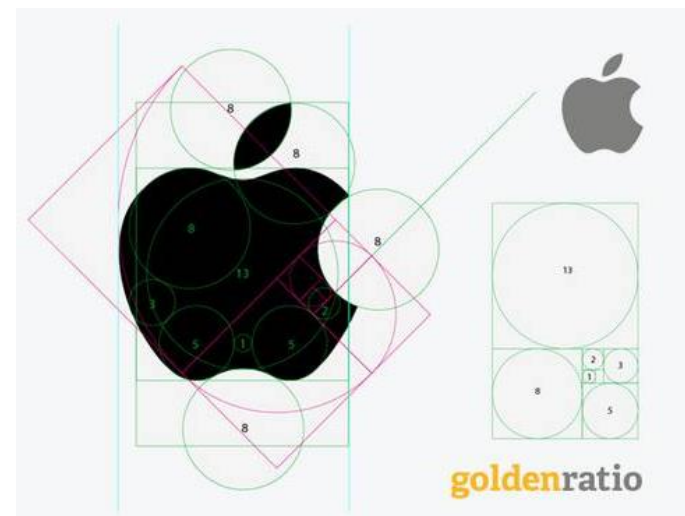
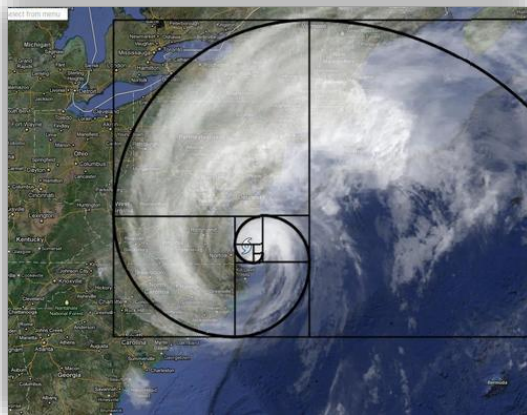
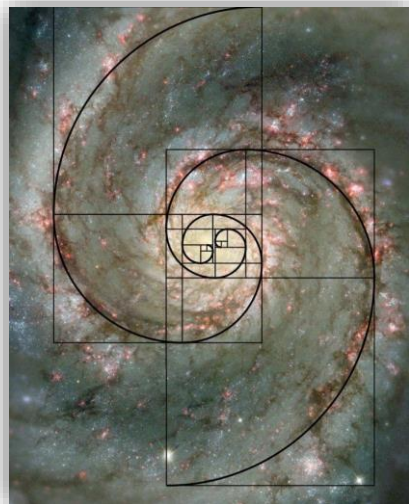
**Przykład:**

**Ciąg Fibonacciego**

Wersja iteracyjna

fib={1,1,2,3,5,8,13,21,.....}

# Ciąg Fibonacciego – reprezentacja graficzna





# Trochę historii

Algorytm Euklidesa – obliczanie największego wspólnego dzielnika dwóch liczb naturalnych.

$$NWD(a, b) = \begin{cases} a & \text{dla } b = 0 \\ NWD(b, a \bmod b) & \text{dla } b \geq 1 \end{cases}$$

1. Dane są dwie liczby naturalne a i b.
2. Oblicz c jako resztę z dzielenia a przez b
3. Zastąp a przez b, zaś b przez c.
4. Jeżeli b = 0, to szukane NWD wynosi a, w przeciwnym wypadku wróć do punktu drugiego i kontynuuj.

NWD(liczba całkowita a, liczba całkowita b)

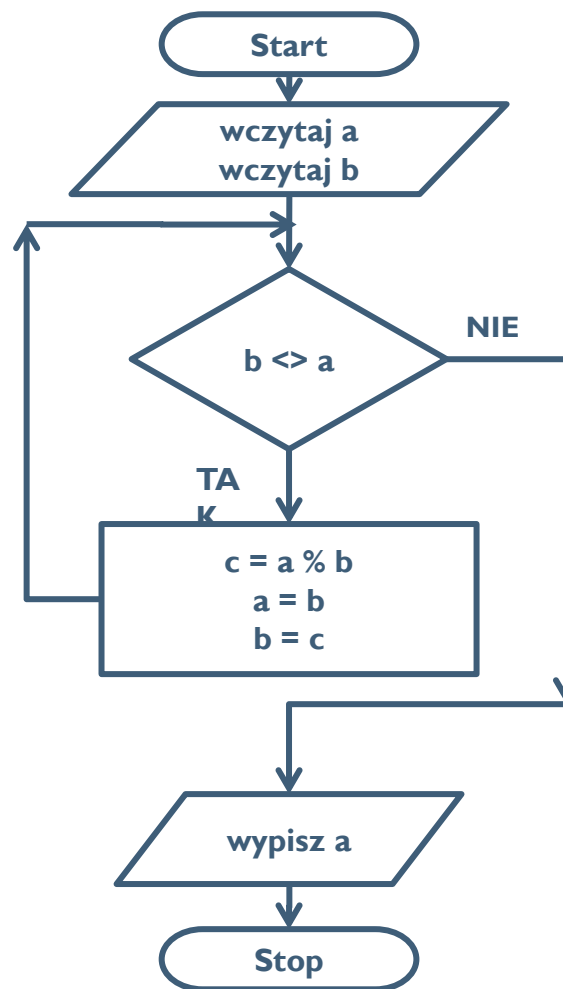
**dopóki** b różne od 0

c = **reszta z dzielenia a przez b**

a = b

b = c

**zwróć** a

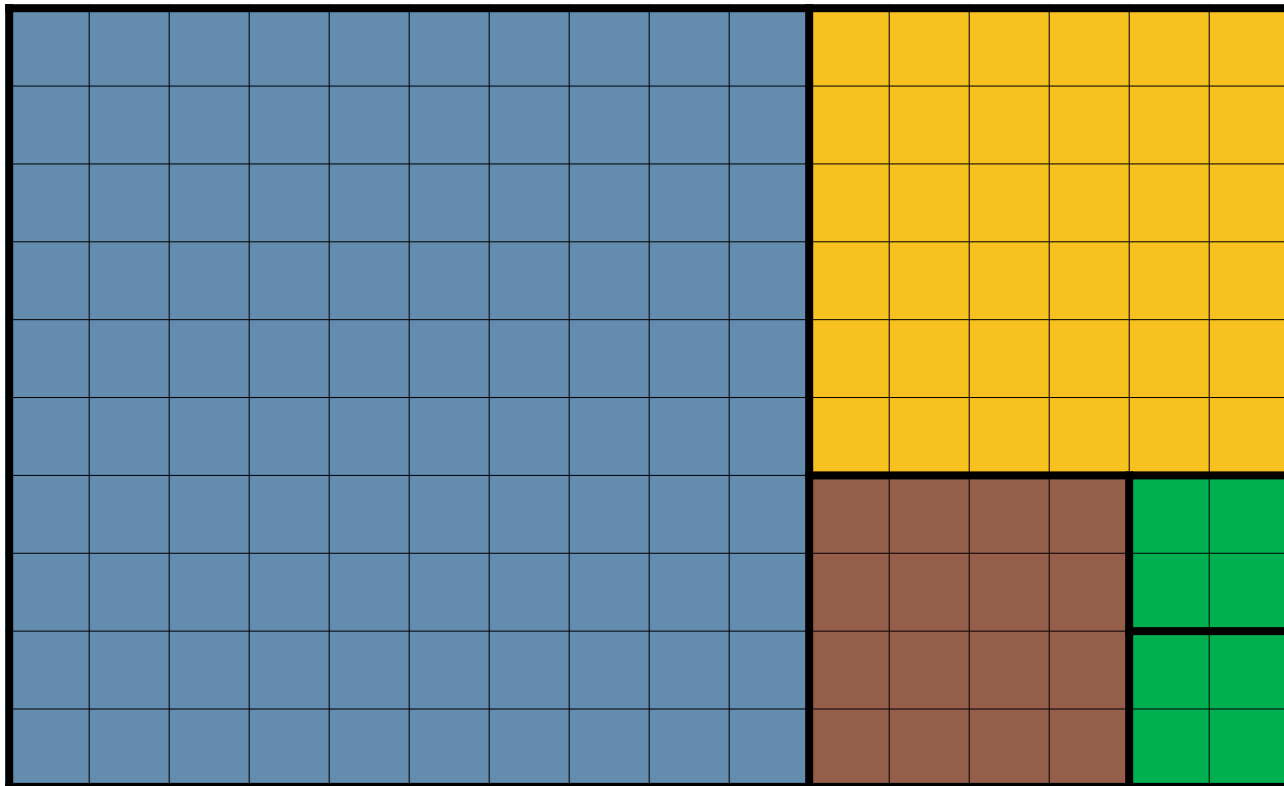


## Trochę historii

Algorytm Euklidesa – obliczanie  
największego wspólnego dzielnika  
dwóch liczb naturalnych.

16

10



**Przykład:**

## Algorytm Euklidesa

```
#include <iostream>
using namespace std;
int main()
{
    unsigned int a, b;
    cout << "a = ";    cin >> a;
    cout << "b = ";    cin >> b;
    while (a!=b)
        if (a>b) a-=b; else b-=a;
    cout << "NWP(a,b) = " << a << endl;
    return 0;
}
```

## Literatura:

---

W prezentacji wykorzystano przykłady i fragmenty:

- Grębosz J.: ***Symfonia C++, Programowanie w języku C++ orientowane obiektowo***, Wydawnictwo Edition 2000.
- Jakubczyk K.: *Turbo Pascal i Borland C++ Przykłady*, Helion.

Warto zająrzeć także do:

- Sokół R.: ***Microsoft Visual Studio 2012 Programowanie w Ci C++***, Helion.
- Kernighan B.W., Ritchie D. M.: ***język ANSI C***, Wydawnictwo Naukowo Techniczne.

Dla bardziej zaawansowanych:

- Grębosz J.: ***Pasja C++***, Wydawnictwo Edition 2000.
- Meyers S.: ***język C++ bardziej efektywnie***, Wydawnictwo Naukowo Techniczne