

Podstawy programowania

Laboratorium 2

Anna Prusinowska

Zadanie 2.0

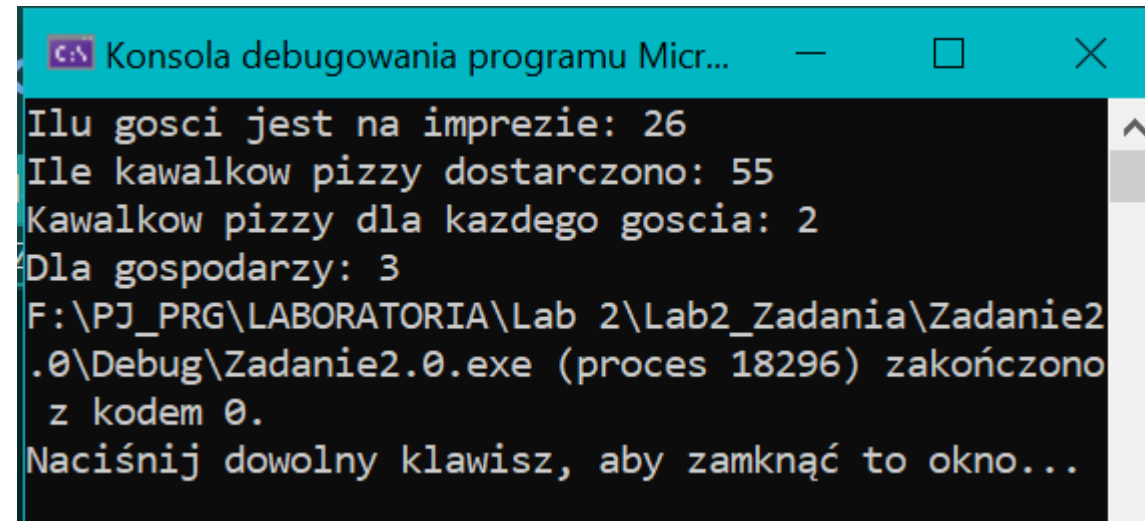
Na rozgrzewkę
- 15 min / 1 pkt

Zadanie 2.0* - na rozgrzewkę

Napisz program, który obliczy, ile kawałków pizzy przypadnie na każdego uczestnika imprezy tak, aby każdy z gości otrzymał tyle samo kawałków pizzy (pozostałe kawałki zostają dla organizatora).

Zadanie 2.0* - rozwiązanie

```
int goscie;  
int pizza;  
int dlaGoscia;  
int dlaGospodarza;  
  
int main()  
{  
    cout << "Ilu gosci jest na imprezie: ";  
    cin >> goscie;  
  
    cout << "Ile kawalkow pizy dostarczono: ";  
    cin >> pizza;  
  
    dlaGoscia = pizza / (goscie);  
    cout << "Kawalkow pizy dla kazdego goscia: " << dlaGoscia;  
  
    dlaGospodarza = pizza - dlaGoscia * (goscie);  
    cout << endl << "Dla gospodarzy: " << dlaGospodarza;  
  
    return 0;  
}
```



Konsola debugowania programu Micr...

Ilu gosci jest na imprezie: 26
Ile kawalkow pizy dostarczono: 55
Kawalkow pizy dla kazdego goscia: 2
Dla gospodarzy: 3
F:\PJ_PRG\LABORATORIA\Lab 2\Lab2_Zadania\Zadanie2.0\Debug\Zadanie2.0.exe (proces 18296) zakończono z kodem 0.
Naciśnij dowolny klawisz, aby zamknąć to okno...

Pętle (ang. loop)

Definicja

Czym i po co są pętle?

Prawie każdy algorytm (a więc i program komputerowy) wykonuje operacje, które powtarzają się wielokrotnie.

Najbardziej popularnym zastosowaniem pętli jest wykonanie operacji na zbiorze danych np. wszystkim studentom danej grupy można przyporządkować kolejne numery; wszystkim uczniom w klasie rozdać cukierki z okazji urodzin.

Częstą praktyką jest stosowanie pętli w pętli.

Czym i po co są pętle? Definicje.

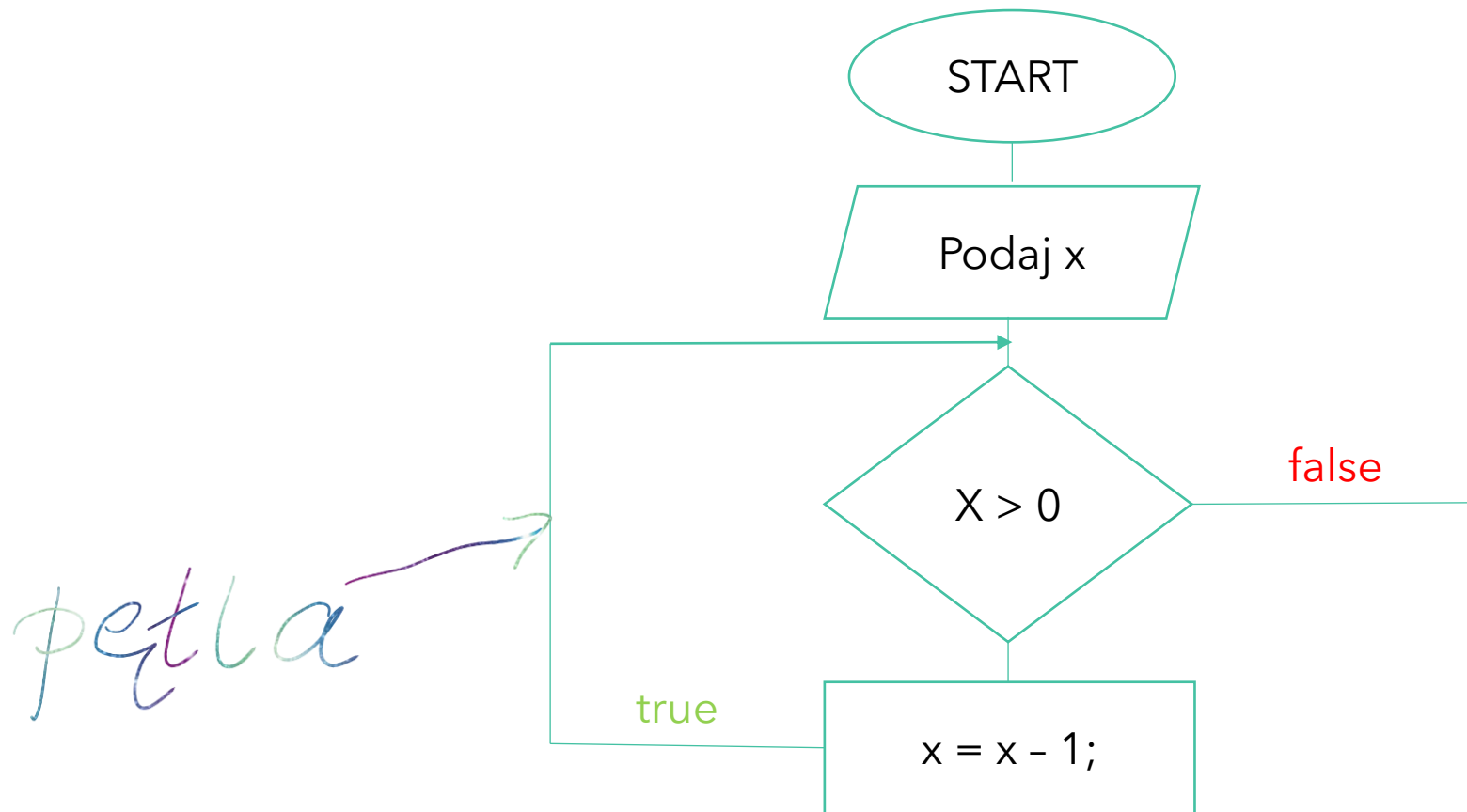
Pętla - specjalna konstrukcja językowa, która ma za zadanie powtarzać pewien zestaw instrukcji określoną liczbę razy.

Pętla to instrukcja iteracyjna (łac. Iteratio - powtarzanie).

Iteracja - pojedyncze wykonanie pętli np. jeśli pętla wykona się 10 razy tzn., że składa się z 10 iteracji.

Iterator (licznik) - przechowuje numer właśnie wykonywanej iteracji. Zmienna ta tradycyjnie nazywa się *i*. Iterator musi być liczbą całkowitą.

Pętla w schemacie blokowym



Zalety stosowania pętli

Pętle pozwalają na:

- wykonywanie operacji taką liczbę razy, której nie da się z góry określić;
- skrócenie kodu – zamiast n-powtórzeń kodu jest 1 wystąpienie;
- łatwiejszy refaktoring – wprowadzamy zmiany w jednym miejscu;
- unikanie błędów – mniej kodu to mniej szans na pomyłki.



Pętla *while*



Pętle

Na czym polega instrukcja *while*

while (wyrażenie / warunek końcowy) instrukcja

Oznacza to, że instrukcja (instrukcje) są powtarzane, tak długo, jak warunek jest spełniony np.

Podczas pandemii stosujemy ograniczenia.

(Domyślnie: Ograniczenia stosowane są tylko w czasie pandemii. Jeśli ta się skończy, ograniczenia nie będą stosowane.)

Dopóki będzie padał deszcz, zostanę w domu.

(Domyślnie: Wyjdę z domu, jak przestanie padać.)

Działanie instrukcji *while*

while (wyrażenie / warunek końcowy) instrukcja

Najpierw oblicza się wyrażenie / warunek końcowy. Jeśli wynik jest zerowy (fałsz), wówczas instrukcja nie jest wykonywana np.

```
int liczba = 10
```

```
(while liczba < 5){  
    cout << „Liczba jest mniejsza od 5.”;  
}
```

„Liczba jest mniejsza od 5.” nigdy nie zostanie wypisane, ponieważ 10 nie jest < 5.

Działanie instrukcji *while*

while (wyrażenie / warunek końcowy) instrukcja

Jeśli po obliczeniu wartości wyrażenia wynik jest niezerowy (prawda), wtedy wykonywana jest instrukcja, po czym ponownie oblicza się wartość wyrażenia. Powtarza się to do momentu, gdy wartość wyrażenia będzie zerowa (fałsz).

```
int liczba = 1
```

```
    (while liczba < 5) {
```

```
        cout << „Liczba jest mniejsza od 5.”; }
```

„Liczba jest mniejsza od 5.” zostanie wypisana.

Przykład 2a

Użycie pętli
while

Przykład 2a: Program, który wypisuje liczby od 1 do 10

```
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8      int number = 1;
9
10     // dla pętli od 1 do 10
11     while (number <= 10) {
12
13         cout << number << " ";
14
15         ++number;
16     }
17     return 0;
18 }
19
```

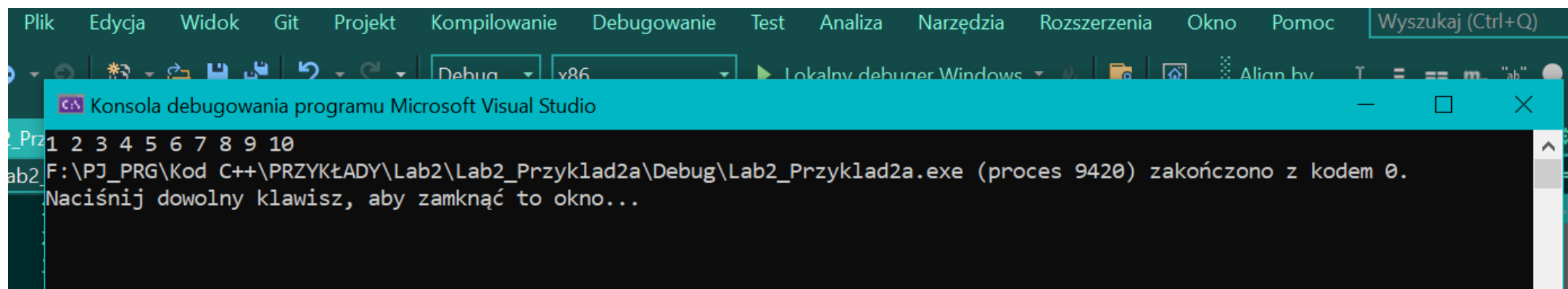
Zmiennej number typu int przypisujemy wartość 1

Warunek: tak długo, jak zmienna number będzie mniejsza lub równa 10

Jeśli warunek jest spełniony, to wypisujemy zmienną, a następnie ją inkrementujemy (zwiększamy jej wartość o 1)

Program wykonał się pomyślnie i zrobił to, co miał zrobić

Przykład 2a: Program, który wypisuje liczby od 1 do 10 - output



The screenshot shows the Visual Studio interface with the 'Debug' menu open. The 'Konsola debugowania programu Microsoft Visual Studio' window is active, displaying the output of the program. The output shows the numbers 1 through 10 on a single line, followed by a message indicating the program has finished execution with code 0. The message also prompts the user to press any key to close the window.

```
Prz1 2 3 4 5 6 7 8 9 10  
ab2 F:\PJ_PRG\Kod C++\PRZYKŁADY\Lab2\Lab2_Przyklad2a\Debug\Lab2_Przyklad2a.exe (proces 9420) zakończono z kodem 0.  
Naciśnij dowolny klawisz, aby zamknąć to okno...
```

Zostały wypisane wszystkie liczby od 1 do 10 włącznie (wszystkie spełniają warunek).

Pętla *do... while*

Pętle

Na czym polega pętla *do... while*

do instrukcjaY while (wyrażenie)

Pętla do/while jest odmianą pętli while. Ta pętla wykona blok kodu raz, przed sprawdzeniem, czy warunek jest spełniony, a następnie powtórzy pętlę, dopóki warunek będzie spełniony.

Różnica pomiędzy pętlą **do... while** a **while** polega na tym, że w przypadku pętli do... while wartość wyrażenia obliczana jest po wykonaniu instrukcjiY. Wynika z tego, że instrukcjaY zostanie wykonana co najmniej jeden raz (nawet wtedy, gdy wyrażenie nie będzie nigdy prawdziwe).

Na czym polega pętla *do... while*

Rób czynność A, dopóki nie zdarzy się B.

Np.

Pisz wypracowanie na maturze, dopóki nie skończy ci się kartka.

Startuj w wyborach, dopóki chcesz zostać politykiem.

Ważne! Zarówno czynność pisania i start w wyborach zdarzą się co najmniej jeden raz – wynika to ze specyfiki pętli *do... while*.

Przykład 2b

Użycie pętli
do... while

Przykład 2b: Program, który wypisuje liczby od 1 do 10 (z wykorzystaniem pętli *do... while*)

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

int main()
{
    int number = 1;

    // pętla do... while od 1 do 10
    do {
        cout << number << " ";
        ++number;
    } while (number <= 10);

    return 0;
}
```

Rozpoczynamy od zmiennej typu `int` o wartości 1

Czynność, która będzie wykonywana („rób”)

Wypisanie na ekran wartości zmiennej, a następnie zwiększenie jej o 1

Czynność będzie powtarzana do momentu, gdy zmienna osiągnie wartość 10 (z 10 włącznie). Gdy `number = 11`, wykonanie pętli zakończy się

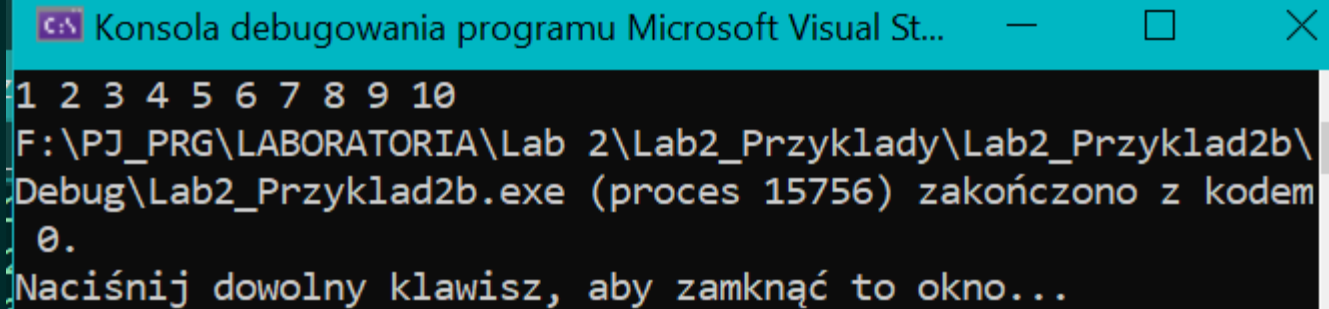
Przykład 2b: Program, który wypisuje liczby od 1 do 10 (z wykorzystaniem pętli *do... while*) - output

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

int main()
{
    int number = 1;

    // pętla do... while od 1 do 10
    do {
        cout << number << " ";
        ++number;
    } while (number <= 10);

    return 0;
}
```



Konsola debugowania programu Microsoft Visual St...

1 2 3 4 5 6 7 8 9 10
F:\PJ_PRG\LABORATORIA\Lab 2\Lab2_Przyklady\Lab2_Przyklad2b\Debug\Lab2_Przyklad2b.exe (proces 15756) zakończono z kodem 0.
Naciśnij dowolny klawisz, aby zamknąć to okno...

Pętla *while* vs. *do... while*

W pętli *while* najpierw sprawdzany jest warunek i dopiero, kiedy on jest spełniony, wykonywane są instrukcje.

W pętli *do... while* najpierw wykonywane są instrukcje, a na koniec sprawdzany jest warunek



Pętla *while* może nie wykonać się ani razu, a pętla *do... while* wykona się przynajmniej 1 raz (są przed sprawdzeniem warunku).



Pętle *for*



Pętle

Na czym polega pętla *for*

`for (instrukcja_inicjalizująca ; wyraz_warunku ; instrukcja_kroku) treść pętli`

Za pomocą pętli *for* można wykonywać te same rzeczy, co za pomocą pętli *do... while*.

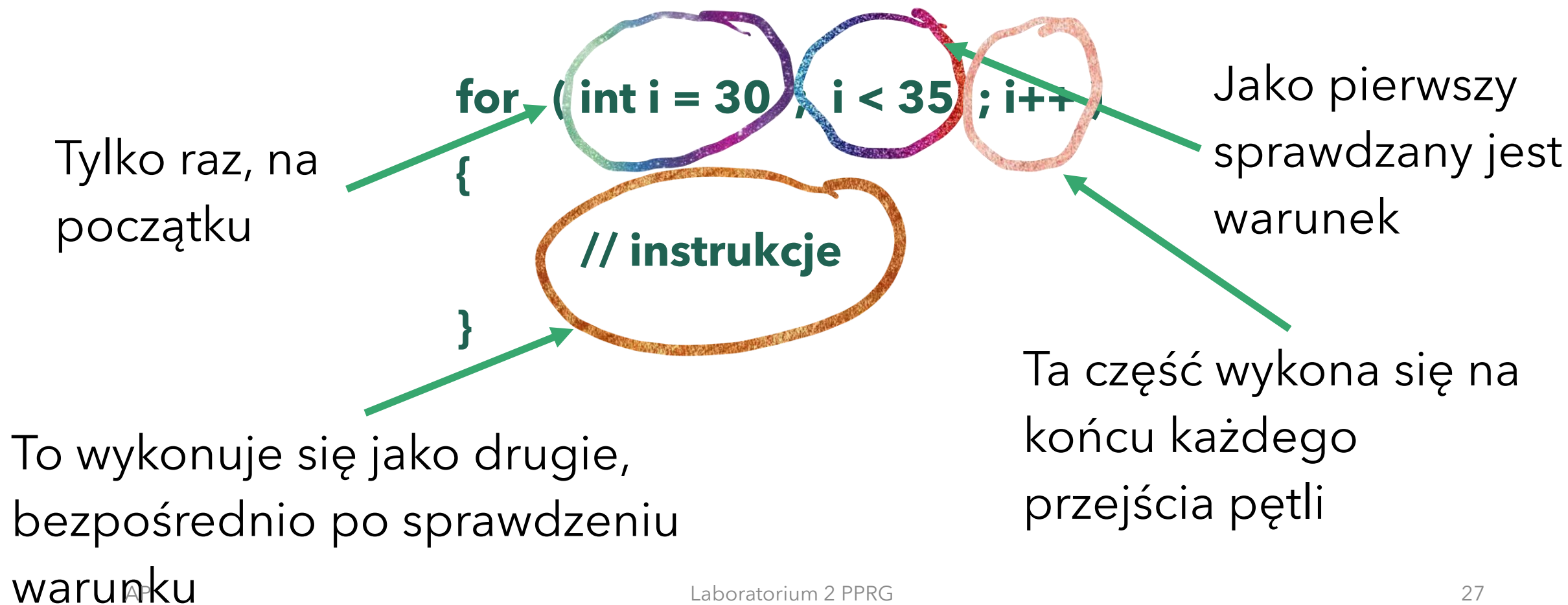
Pętli *for* używamy najczęściej, gdy znamy ilość danych do wczytania, wypisania lub zmiany np. jeśli chcemy wypisać na ekran wszystkie pozycje leków z recepty, wypisanej przez lekarza, daty i miejsca trasy koncertowej ulubionego wykonawcy lub wczytać do systemu oceny z kolokwium wszystkim studentom z grupy.

Kolejność wykonywania się poszczególnych części pętli *for*

for (instrukcja_inicjalizująca ; wyraz_warunku ; instrukcja_kroku) treść pętli

for	dla takich warunków rób
instrukcja_inicjalizująca	instrukcja wykonywana, zanim pętla zostanie uruchomiona po raz pierwszy
wyraz_warunku	wyrażenie, obliczane, przed każdym obiegiem pętli; jeśli jest ono różne od zera, to wykonywane zostają instrukcje będące treścią pętli np. $i < 0$ – jeśli i będzie mniejsze od zera, to wykonywana zostaje instrukcja będąca treścią pętli
instrukcja_kroku	instrukcja wykonywana na zakończenie każdego obiegu pętli; ostatnia instrukcja, wykonywana bezpośrednio przed obliczeniem wyrażenia <code>wyraz_warunku</code>

Kolejność wykonywania się poszczególnych części pętli *for*



Kolejność wykonywania się poszczególnych części pętli *for*

for (instrukcja_inicjalizująca ; wyraz_warunku ; instrukcja_kroku) treść pętli

Ważne:

- może być kilka instrukcji_inicjalizujących lub instrukcji_kroku (oddzielonych przecinkami);
- każdy element z nawiasu można opuścić, zachowując średnik oddzielający od sąsiada; opuszczenie wyrażenia warunkowego traktowane jest tak, jakby stało tam wyrażenie zawsze prawdziwe;
- pętle nieskończone np.

Inkrementacja a dekrementacja

Inkrementacja - zwiększanie wartości zmiennej dokładnie o 1, np.
`i++` lub `i = i + 1`

(ang. to increment - zwiększyć).

Dekrementacja - zmniejszanie wartości zmiennej dokładnie o 1, np.
`i--` lub `i = i - 1`

(ang. to decrement - zmniejszyć).

Przykład 2c

Użycie pętli *for*

Przykład 2c: Program przedstawiający wykorzystanie pętli for - wypisywanie zdania na ekran n-razy.

```
#include <iostream>

using namespace std;

int main()
{
    int licznik_petli;

    for (licznik_petli = 1; licznik_petli < 10; licznik_petli++) {
        cout << licznik_petli << " przebieg petli" << endl;
    }

    return 0;
}
```

Zaczynamy od 1

Pętla będzie się wykonywać dopóki wartość zmiennej licznik_petli będzie mniejsza niż 10

Program wypisuje wartość zmiennej licznik_petli oraz zmienną tekstową „przebieg petli”

Po wypisaniu na ekran wartość zmiennej licznik_petli zostaje zwiększona o 1

Przykład 2c: Program przedstawiający wykorzystanie pętli for - wypisywanie zdania na ekran n-razy - Output.

```
#include <iostream>

using namespace std;

int main()
{
    int licznik_petli;

    for (licznik_petli = 1; licznik_petli < 10; licznik_petli++) {
        cout << licznik_petli << " przebieg petli" << endl;
    }

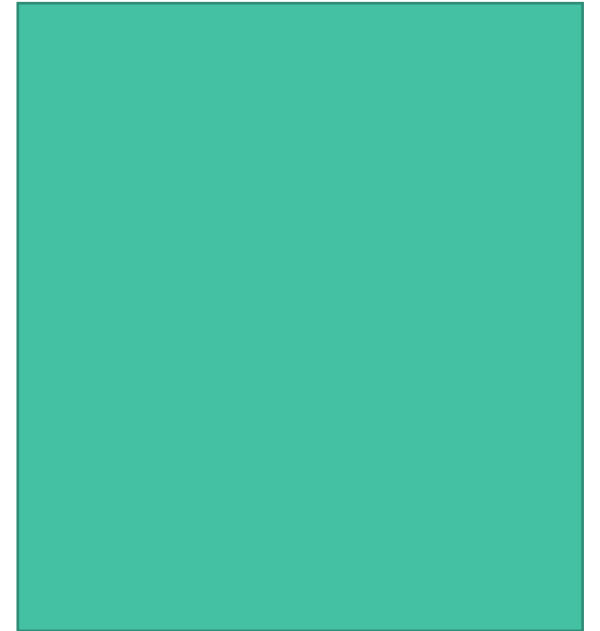
    return 0;
}
```

C:\> Konsola debugowania programu Micro

```
1 przebieg petli
2 przebieg petli
3 przebieg petli
4 przebieg petli
5 przebieg petli
6 przebieg petli
7 przebieg petli
8 przebieg petli
9 przebieg petli
```

```
F:\PJ_PRG\LABORATORIA\Lab 2\Lab
Debug\Lab2_Przyklad2c.exe (proc
0.
Naciśnij dowolny klawisz, aby z
```


Instrukcja *switch*



Instrukcja *switch*

Switch (ang. przełącznik) – instrukcja wyboru.

Instrukcja *switch* służy do podejmowania wielowariantowych decyzji. Obliczane jest wyrażenie umieszczone w nawiasie przy słowie *switch*.

Instrukcja *switch*

```
switch(wyrażenie) {  
    case wartosc1:  
        instrukcjaA;  
        break;  
    case wartosc2:  
        instrukcjaB;  
        break;  
    case wartosc3:  
        instrukcjaC;  
        break;  
}
```

Instrukcja *switch*

Jeśli wartość wyrażenia odpowiada którejś z wartości podanej w jednej z etykiet case, wówczas wykonywane są instrukcje począwszy od tej etykiety. Wykonywanie ich skończy się po napotkaniu instrukcji break. Powoduje to wyskok z instrukcji switch.

Jeśli wartość wyrażenia nie zgadza się z żadną z wartości podanych przy etykietach case, wówczas wykonują się instrukcje umieszczone po etykiecie default (etykieta default może być w dowolnym miejscu lub może jej nie być wcale).

Jeśli etykiety default nie ma, a wartość wyrażenia nie zgadza się z żadną z wartości przy etykietach case, wówczas opuszcza się instrukcję switch nie wykonując niczego.

Instrukcji następujących po etykiecie case nie musi kończyć instrukcja break. Jeśli jej nie będzie, to zaczną wykonywać się instrukcje umieszczone pod następną etykietą case.

Przykład 2d

Instrukcja
switch

Przykład 2d:

Napisz program, który wykorzystuje numer dnia tygodnia do obliczenia nazwy dnia tygodnia. Liczba do obliczenia dnia tygodnia powinna być pobrana od użytkownika.

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

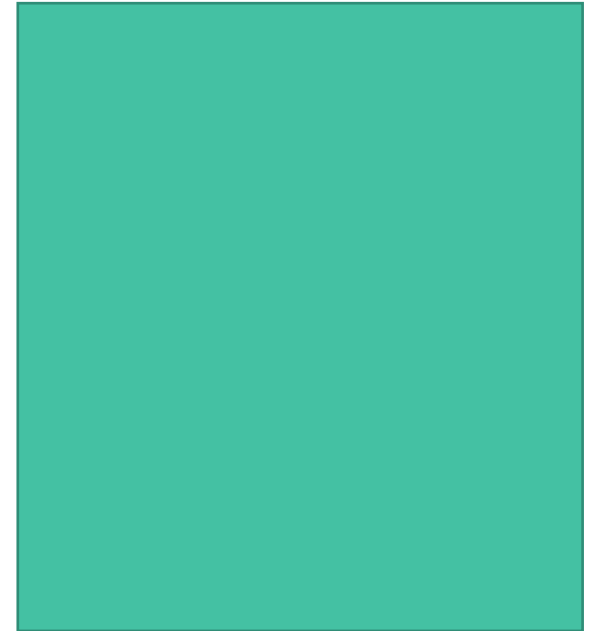
int main()
{
    int day;
    cout << "Podaj liczbe od 1 do 7.\n";
    cin >> day;
    switch (day)
    {
        case 1:
            cout << "Poniedzialek";
            break;
        case 2:
            cout << "Wtorek";
            break;
        case 3:
            cout << "Sroda";
            break;
        case 4:
            cout << "Czwartek";
            break;
        case 5:
            cout << "Piatek";
            break;
        case 6:
            cout << "Sobota";
            break;
        case 7:
            cout << "Niedziela";
            break;
        default:
            cout << "Wybrales liczbe spoza zakresu";
    }
    return 0;
}
```

Przykład 2d: Przykładowy output

```
Konsola debugowania programu Microsoft Visual Stud...  
Podaj liczbe od 1 do 7.  
6  
Sobota  
  
F:\PJ_PRG\LABORATORIA\Lab 2\Lab2_Przyklady\Lab2_Przyklad2d\De  
bug\Lab2_Przyklad2d.exe (proces 16868) zakończono z kodem 0.  
Naciśnij dowolny klawisz, aby zamknąć to okno...
```

```
#include <iostream>  
using namespace std;  
  
int main()  
{  
    int day;  
    cout << "Podaj liczbe od 1 do 7.\n";  
    cin >> day;  
    switch (day)  
    {  
        case 1:  
            cout << "Poniedzialek";  
            break;  
        case 2:  
            cout << "Wtorek";  
            break;  
        case 3:  
            cout << "Sroda";  
            break;  
        case 4:  
            cout << "Czwartek";  
            break;  
        case 5:  
            cout << "Piatek";  
            break;  
        case 6:  
            cout << "Sobota";  
            break;  
        case 7:  
            cout << "Niedziela";  
            break;  
        default:  
            cout << "Wybrales liczbe spoza zakresu";  
    }  
    return 0;  
}
```

Instrukcja *break*
przerywająca
wykonanie instrukcji
pętli *for*, *while*, *do...*
while



Instrukcja *break* przerywająca wykonanie instrukcji pętli *for*, *while*, *do... while*

W przypadku kilku zagnieżdżonych pętli instrukcja *break* powoduje przerwanie tylko tej pętli, w której bezpośrednio tkwi (przerwanie z wyjściem o jeden poziom wyżej).

Instrukcja *break* kończy wykonywanie najbliższej otaczającej pętli lub instrukcji warunkowej, w której się pojawia. Jeżeli po końcu przerwanej instrukcji występuje kolejna, sterowanie przechodzi do niej.

Przykład 2e

Break

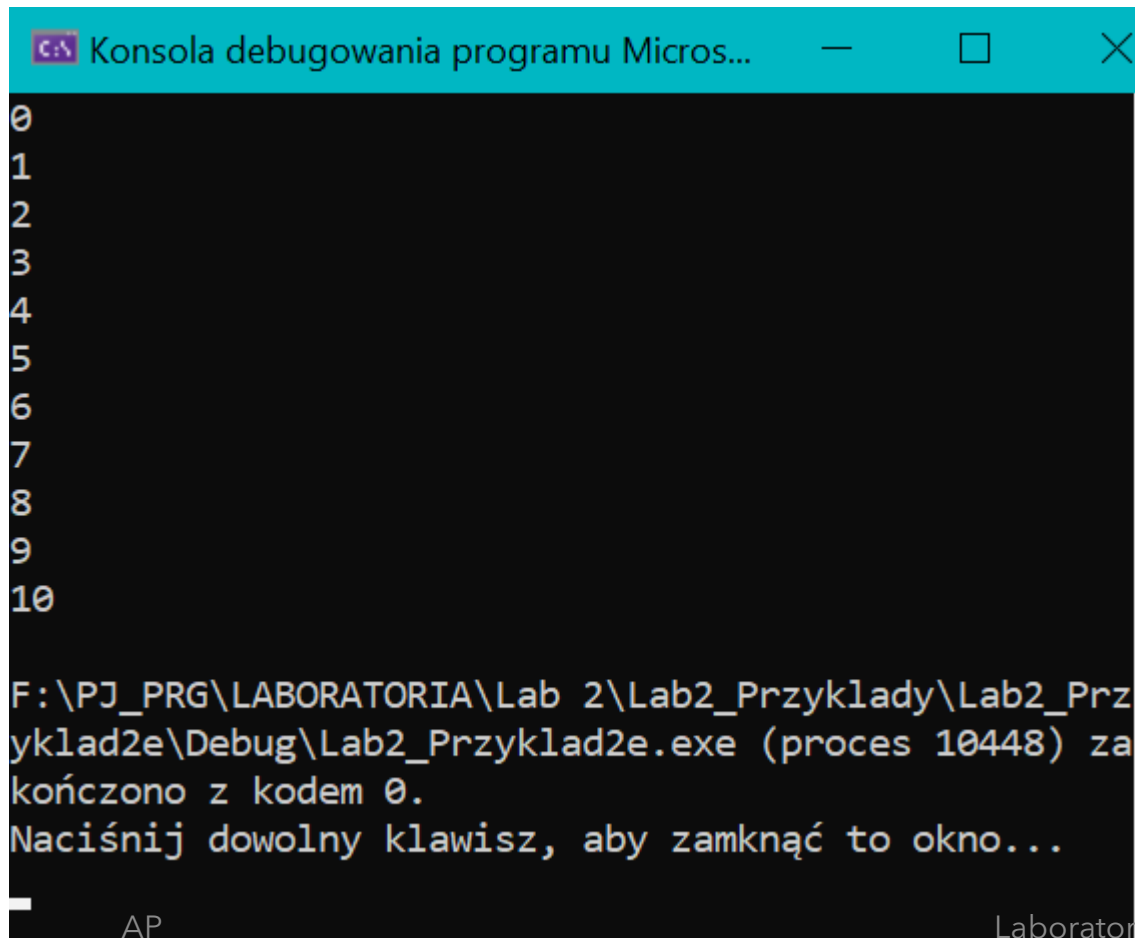
Przykład 2e:

Napisz program, który z liczb od 0 do 16 wypisuje kolejne, do momentu spełnienia instrukcji *break*.

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

int main()
{
    int i = 0;
    while (i < 16) {
        cout << i << "\n";
        i++;
        if (i == 11) {
            break;
        }
    }
    return 0;
}
```

Przykład 2e: Przykładowy output



```
C:\> Konsola debugowania programu Micros...  
0  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
  
F:\PJ_PRG\LABORATORIA\Lab 2\Lab2_Przyklady\Lab2_Przyklad2e\Debug\Lab2_Przyklad2e.exe (proces 10448) zakończono z kodem 0.  
Naciśnij dowolny klawisz, aby zamknąć to okno...
```

```
#include <iostream>  
using namespace std;  
  
int main()  
{  
    int i = 0;  
    while (i < 16) {  
        cout << i << "\n";  
        i++;  
        if (i == 11) {  
            break;  
        }  
    }  
    return 0;  
}
```

Instrukcja *continue*



Instrukcja *continue*

Jest przydatna wewnątrz pętli `for`, `while`, `do... while`. Powoduje zaniechanie instrukcji będących treścią pętli, jednak w przeciwieństwie do instrukcji `break`, sama pętla nie zostaje przzerwana. *Continue* przerywa tylko ten obieg pętli i zaczyna następny, kontynuujący pracę pętli.

Instrukcja *continue* powoduje przerwanie wykonania bieżącego kroku pętli i przejście do następnego kroku. Instrukcja *continue* działa tylko na pętlę, w której się bezpośrednio znajduje – nie da się spowodować przerwania kroku pętli zewnętrznej.

Przykład 2f

Continue

Przykład 2f:

Instrukcja `continue` w zagnieżdżonej pętli

Instrukcja `continue` powoduje przerwanie wykonania bieżącego kroku pętli i przejście do następnego kroku. Instrukcja `continue` działa tylko **na pętli**, w której się bezpośrednio znajduje – nie da się spowodować przerwania kroku **pętli zewnętrznej**.

AP

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

int main()
{
    int number;
    int sum = 0;

    // zagnieżdżenie pętli

    // pierwsza pętla
    for (int i = 1; i <= 3; i++) {
        // druga pętla
        for (int j = 1; j <= 3; j++) {
            if (j == 2) {
                continue;
            }
            cout << "i = " << i << ", j = " << j << endl;
        }
    }
    return 0;
}
```

Laboratorium 2 PPRG

Przykład 2f:

Przykładowy output

Instrukcja *continue* powoduje przerwanie wykonania bieżącego kroku pętli i przejście do następnego kroku. Instrukcja *continue* działa tylko **na pętlę**, w której się bezpośrednio znajduje – nie da się spowodować przerwania kroku **pętli zewnętrznej**.

AP

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

int main()
{
    int number;
    int sum = 0;

    // zagnieżdżenie pętli

    // pierwsza pętla
    for (int i = 1; i <= 3; i++) {

        // druga pętla
        for (int j = 1; j <= 3; j++) {
            if (j == 2) {
                continue;
            }
            cout << "i = " << i << ", j = " << j << endl;
        }
    }

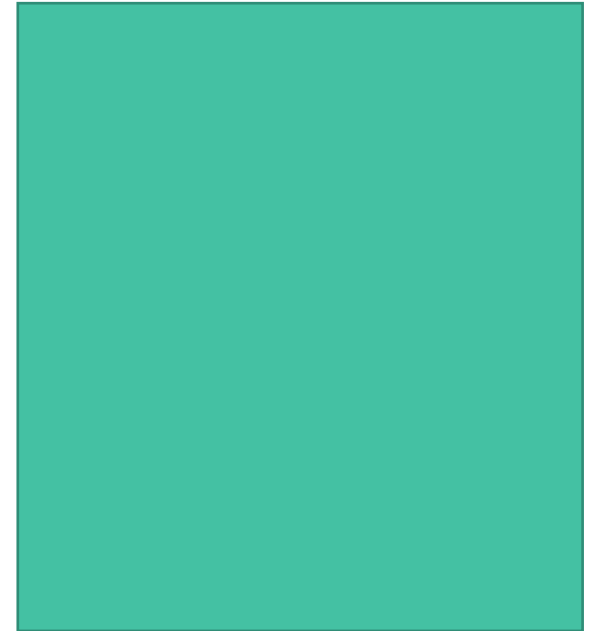
    return 0;
}
```

C:\N Konsola debugowa...

```
i = 1, j = 1
i = 1, j = 3
i = 2, j = 1
i = 2, j = 3
i = 3, j = 1
i = 3, j = 3
```

Laboratorium 2 PPRG

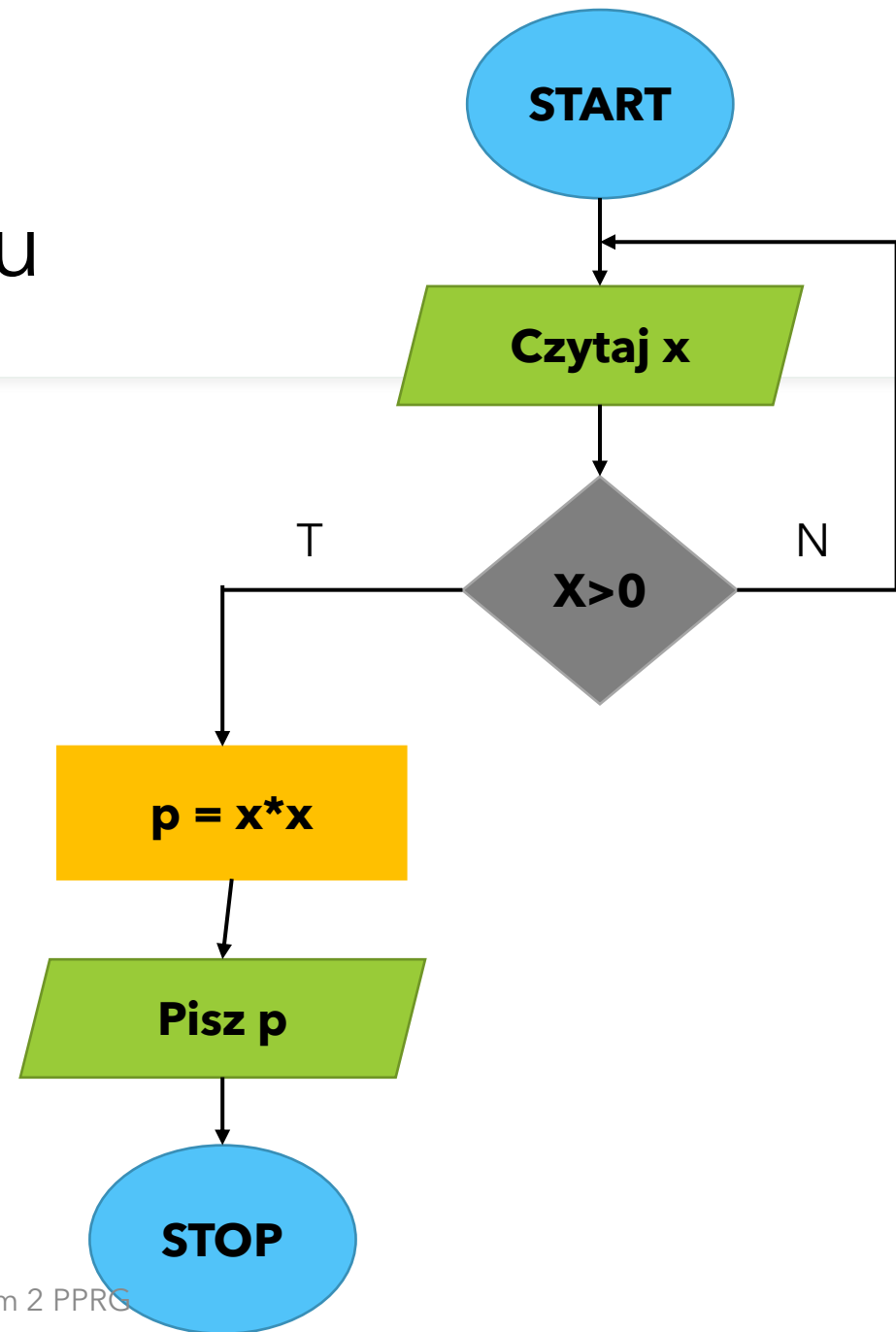
Schematy blokowe



Czym jest schemat blokowy?

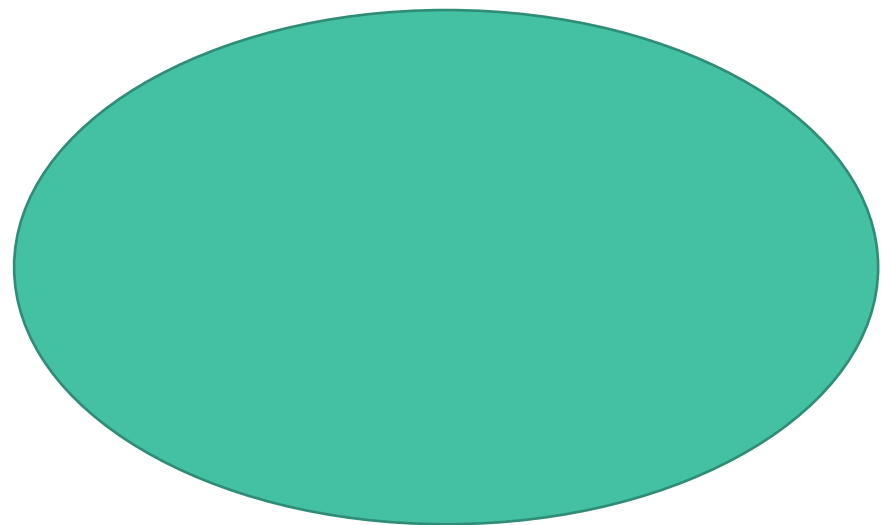
Schematy blokowe to diagramy, przedstawiające etapy procesu. Podstawowe schematy blokowe są łatwe do utworzenia oraz zrozumienia dzięki prostocie i obrazowości kształtów.

Przykład: Algorytm na obliczanie pola kwadratu



Bloki stosowane w schematach blokowych





START/STOP

- początek i koniec algorytmu; tego kształtu należy użyć w pierwszym i ostatnim etapie procesu



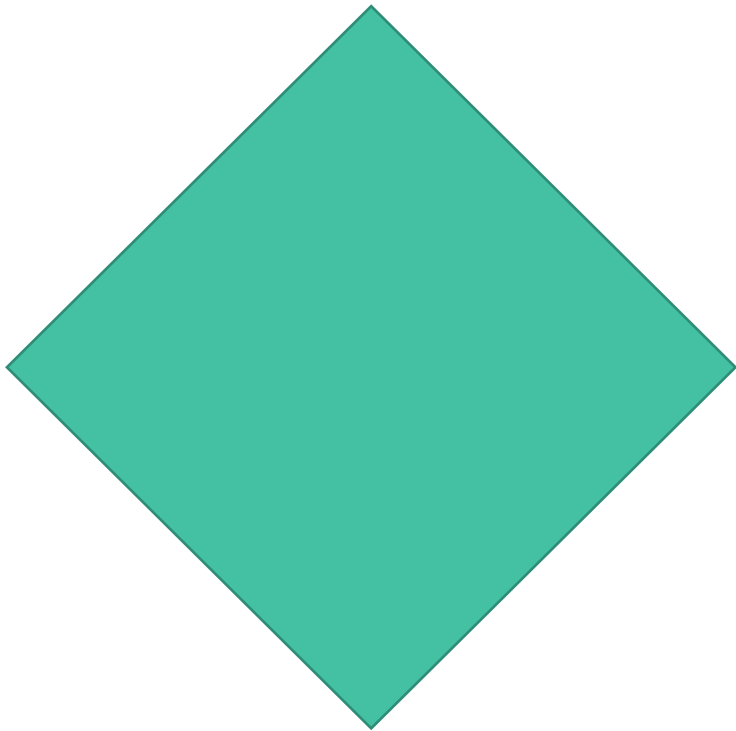
WPROWADZANIE DANYCH

- kształt ten wskazuje, że informacje przychodzą do procesu z zewnątrz lub opuszczają proces. Kształt ten nazywany bywa kształtem wejścia/wyjścia



WYKONYWANIE DZIAŁAŃ

- kształt ten odpowiada typowemu etapowi procesu; Ttzw. „blok operacyjny”



DECYZJA (SPRAWDZENIE WARUNKU)

- blok warunkowy albo decyzyjny; kształt ten wskazuje punkt, w którym wybór kolejnego etapu zależy od podjęcia decyzji; wyników może być wiele, ale zwykle są tylko dwa, odpowiadające decyzjom „TAK” lub „NIE”



ŁĄCZNIK (ODWOŁANIE NA STRONIE)

- kształt ten wskazuje, że następny lub poprzedni etap znajduje się w innym miejscu na rysunku; umieszczony wewnątrz numer powinien być taki sam w obu częściach



POŁĄCZENIE

- 
- kształt ten łączy bloki lub dochodzi do innego połączenia

Zadania

*Do
samodzielnego
wykonania*

Zadanie 2.1

Napisz program do obliczania szeregu wg podanego wzoru:
 $(1) + (1+2) + (1+2+3) + (1+2+3+4) + \dots + (1+2+3+4+\dots+n)$.
(1 pkt)

Reprezentacja graficzna programu dla $n = 5$:

$$1 = 1$$

$$1 + 2 = 3$$

$$1 + 2 + 3 = 6$$

$$1 + 2 + 3 + 4 = 10$$

$$1 + 2 + 3 + 4 + 5 = 15$$

$$\text{Suma szeregu: } 1 + 3 + 6 + 10 + 15 = 35$$

Zadanie 2.2

Napisz program, który:

- A) Pobierze od Użytkownika liczb "a" oraz "b" oraz wypisze je na ekran.
- B) Wypisze wiersz gwiazdek o długości "a".
- C) Wypisze kolumnę gwiazdek o długości "b".
- D) Wypisze wypełniony prostokąt gwiazdek o wymiarach "a" na "b".
- E) Wypisze obwód (obramowanie) prostokąta gwiazdek o wymiarach "a" na "b". (Wskazówka: warto skorzystać z operatora logicznego || - OR).

=== Podpunkty bonusowe ===

F*) Wypisze trójkąt prostokątny równoramienny:

- o pionowej przyprostokątnej o długości "a" oraz poziomej przyprostokątnej o długości TAKŻE "a".
- liczba gwiazdek w kolejnych wierszach: 1, 2, 3, ..., a-1, a.
- kąt prosty w trójkącie: lewy dolny róg.

G*) Jak w podpunkcie F, ale:

- liczba gwiazdek w kolejnych wierszach NA ODWRÓT: a, a-1, ..., 3, 2, 1;
- kąt prosty w trójkącie: prawy górny róg.

Zadanie 2.2:

Przykładowy
output:

```
Pobrano liczby a == 4 oraz b == 2
```

```
Wiersz o dlugosci a:
```

```
****
```

```
Kolumna o dugosci b:
```

```
*
```

```
*
```

```
Prostokat gwiazdek o wymiarach 'a' na 'b':
```

```
**
```

```
**
```

```
**
```

```
**
```

```
Obwod prostokatu o wymiarach 'a' na 'b':
```

```
**
```

```
**
```

```
**
```

```
**
```

```
Trojkat prostokatny rownoramienny z katem prostym w lewym dolnym rogu :
```

```
*
```

```
**
```

```
***
```

```
****
```

```
Trojkat prostokatny rownoramienny z katem prostym w prawym gornym rogu :
```

```
****
```

```
***
```

```
*Laboratorium 2 PPRG
```

```
*
```

Zadanie 2.3

Użyj instrukcji *switch* i napisz program, który:

- a) Pobierze od Użytkownika liczbę z przedziału $[1,12]$, reprezentującą wybrany miesiąc w roku.
- b) Jeśli podana przez użytkownika liczba będzie spoza przedziału, program poinformuje o tym użytkownika i zakończy się.
- c) Wypisze polską nazwę miesiąca (ale bez polskich znaków).
- d) Wypisze ile dni ma wybrany miesiąc w roku nieprzestępnym.
- e) Wypisze czy w tym miesiącu jest słonecznie, czy pochmurno.
Słonecznie jest od kwietnia włącznie do września włącznie.

(2 pkt)