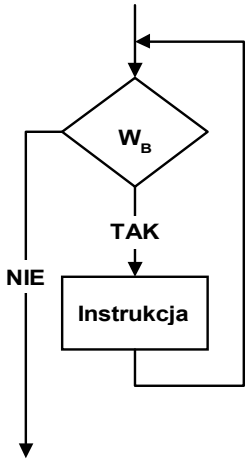
1. Instrukcja `while`.

Postać ogólna instrukcji **while**: `while` W_B `do Instrukcja;`

Działanie instrukcji **while** polega na powtarzaniu wykonywania instrukcji stanowiącej wnętrze iteracji (Rys. 1) dopóki spełniony jest warunek logiczny $\mathbf{W_B}$ (posiada wartość **true**). Wartość ta jest sprawdzana każdorazowo **przed wykonaniem instrukcji**. Wnętrze iteracji jest praktycznie zawsze instrukcją sekwencji **begin ... end**, która oprócz powtarzanych obliczeń **powinna także zawierać** operację wpływającą na wartość wyrażenia logicznego $\mathbf{W_B}$, aby w ten sposób doprowadzić do zakończenia powtarzania iteracji.



Rys. 1. Ilustracja działania instrukcji **while**.

2. Sprawdź działanie przykładowego programu wykorzystującego instrukcję `while` do obliczenia sumy liczb parzystych od 2 do 100.

```
program p1;
var i, suma: Integer;
begin
    i := 2;
    suma := 0;
    while i <= 100 do
    begin
          suma := suma + i;
          i := i + 2;
    end;
    writeln('Suma liczb: ', suma);
end.
```

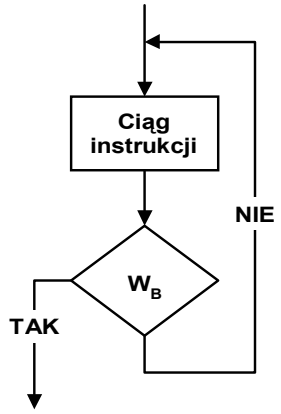
3. Instrukcja `repeat`.

Postać ogólna instrukcji **repeat .. until**:

```
repeat
    Instrukcja_1;
    Instrukcja_1;
    ...
until WB;
```

Ciąg instrukcji wewnętrznych jest powtarzany dopóty, dopóki wyrażenie logiczne $\mathbf{W_B}$ posiada wartość **false**, zmiana na **true** powoduje zakończenie pętli (Rys. 2). Sprawdzanie warunku odbywa się **na końcu pętli**, stąd jest ona **conajmniej raz wykonywana**. Podobnie jak w iteracji **while** jedna z instrukcji **powinna mieć wpływ** na wartość wyrażenia logicznego.

Instrukcję **repeat .. until** można wykorzystać do chwilowego zatrzymania programu w celu oczekiwania na naciśnięcie dowolnego klawisza (określonej litery lub konkretnego klawisza).



Rys. 2. Ilustracja działania instrukcji **repeat .. until**.

4. Sprawdź działanie przykładowego programu wykorzystującego instrukcję `repeat`.

```
program p2;
uses crt;
var  suma: Integer;
     Tak_Nie: Char;
begin
     repeat
           suma := suma + 1;
           write(suma);
           writeln('   Jeszcze raz? (T)ak');
           Tak_Nie := upcase(readkey);
     until not(Tak Nie = 'T');
end.
```

5. Wykonaj zadania 2, 5 i 6.

Zadania

1. Napisz program, który dla pobranego od użytkownika ciągu liczb $a_1, a_2, ..., a_n$, zakończonego liczbą **0** (zero kończy wprowadzanie danych), wyznaczy i wyświetli element minimalny oraz jego indeks (nie uwzględniać końcowego zera).

2. Napisz program, który wczytuje z klawiatury dane (liczby) rzeczywiste do momentu, gdy ich suma przekroczy wartość **1000**, a następnie wyświetla stosowną informację oraz liczbę wprowadzonych danych.

3. Napisz program wyznaczający **największy wspólny podzielnik** 2 liczb.

Wskazówka: Szukając **NWD (a, b)** należy podzielić większą z liczb (**a**) przez mniejszą (**b**) i wyznaczyć iloraz i resztę: $a = t \times b + r$. Przy dzieleniu skorzystać z operatorów **div** (dzielenie całkowite) i **mod** (reszta z dzielenia). Następnie należy podzielić liczbę **b** przez resztę **r** w wyniku czego otrzymuje się nowy iloraz i nową resztę: $b = t_1 \times a_1 + r_1$.
Poniższy przykład ilustruje zastosowanie tego algorytmu (algorytm Euklidesa) do wyznaczenia największego wspólnego dzielnika liczb **252** i **70**:

252 : 70 = 3, reszta 42;
70 : 42 = 1, reszta 28;
42 : 28 = 1, reszta 14;
28 : 14 = 2, reszta 0.

Największym wspólnym dzielnikiem tych liczb jest **14** (ostatnia reszta, która jest różna od zera).

4. Napisz program wyznaczający **najmniejszą wspólną wielokrotność** 2 liczb.

Wskazówka: W programie możesz wykorzystać następującą zależność: **a × b = NWW (a, b) × NWD (a,**

5. Zmodyfikuj zadanie 2, aby naciśnięcie klawisza, np. **N** kończyło wprowadzanie danych.

6. Napisz program obliczający średnią arytmetyczną wprowadzanych liczb z chwilą, gdy ich suma przekroczy wartość **150**.

7. Napisz program przekształcający wprowadzoną liczbę całkowitą na wyrażenie typu string o żądanym formacie, np. po wprowadzeniu liczby w postaci **123456789** zostanie ona wyświetlona następująco: **123 456 789**.