

7. INSTRUKCJE DECYZYJNE

7.1 Wyrażenia relacyjne i logiczne

Wyrażenia relacyjne są koniecznym składnikiem instrukcji warunkowych i powtarzających. Powstają one w wyniku łączenia zmiennych i stałych lub wyrażeń operatorami relacyjnymi. Poniżej podajemy przykłady takich wyrażeń:

$A+1 > B$ { Aktualna wartość $A+1$ jest większa od aktualnej wartości B . }
 $X=0$ { Aktualna wartość $X=0$. }
 $X \bmod 2 = 0$ { Reszta z dzielenia aktualnej wartości X przez 2 jest zerowa. }
 $K \text{ in } [1..6]$ { Aktualna wartość K jest elementem zbioru liczb $[1 .. 6]$. }

Po napotkaniu wyrażenia relacyjnego, program sprawdza, czy przy aktualnych wartościach zmiennych i zdefiniowanych stałych relacja jest spełniona. Jeżeli tak, całej relacji nadawana jest wartość logiczna *True*; w przeciwnym przypadku relacji nadawana jest wartość logiczna *False*. Należy zapamiętać, że:

- Wartość wyrażenia relacyjnego, jako całości, jest typu *Boolean*.
- Jeżeli wyrażenie relacyjne jest prawdziwe, ma ono wartość *True*.
- Jeżeli wyrażenie relacyjne jest nieprawdziwe, ma ono wartość *False*

Ponieważ relacja ma wartość logiczną, można ją w całości przypisać do zmiennej typu *Boolean*, by przechować aktualną wartość tej relacji, na przykład:

```
var B:Boolean;  
begin  
  B:=(X mod 2 = 0);  
end.
```

Skoro relacje mają wartości logiczne, to mogą być argumentami pokazanych w tabeli 7.1 *funkcji logicznych* Turbo Pascala, na przykład wyrażenie:

$(A > B) \text{ and } (A > C)$

przyjmie wartość *True* wtedy i tylko wtedy, gdy obie relacje składowe, połączone operatorem *and*, są prawdziwe. Natomiast wyrażenie:

$(A <= B) \text{ or } (A <= C)$

przyjmie wartość *True* wtedy, gdy chociaż jedna z dwu relacji łączonych operatorem *or* jest prawdziwa, a *False* wtedy i tylko wtedy, gdy obie są nieprawdziwe.

Oczywiście również takie wielocłonowe wyrażenie relacyjne można przypisać zmiennej logicznej, na przykład:

$B := (A > B) \text{ and } (A > C) ;$

Relacje łączone operatorami logicznymi piszemy w nawiasach. Jest to konieczne ze względu na obowiązujący w Turbo Pascalu priorytet operatorów.

Tabela 7.1. Definicje operacji logicznych na zmiennych: var A, B: Boolean;

| Kombinacje wartości argumentów | | Wyniki operacji logicznych | | | | |
|--------------------------------|-------|----------------------------|-------|---------|--------|---------|
| A | B | not A | not B | A and B | A or B | A xor B |
| FALSE | FALSE | TRUE | TRUE | FALSE | FALSE | FALSE |
| FALSE | TRUE | TRUE | FALSE | FALSE | TRUE | TRUE |
| TRUE | FALSE | FALSE | TRUE | FALSE | TRUE | TRUE |
| TRUE | TRUE | FALSE | FALSE | TRUE | TRUE | FALSE |

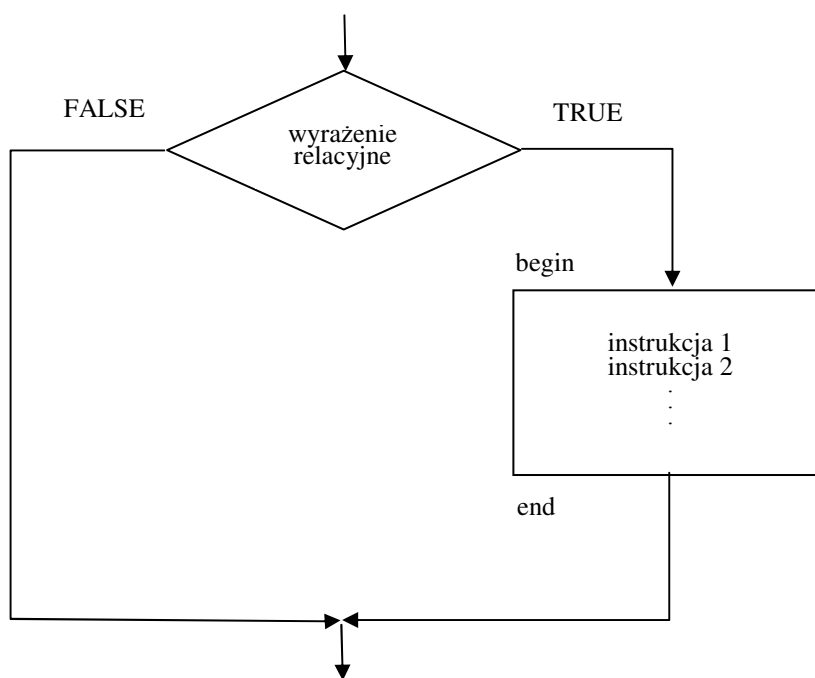
7.2. Instrukcja warunkowa if-then

Instrukcja *if-then* (po polsku: jeżeli-to) ma następującą składnię:

```
if wyrażenie_relacyjne then begin
    instrukcja_1;
    instrukcja_2;
    {-----}
    instrukcja_n;
end;
```

W klamrze utworzonej przez słowa kluczowe *begin* i *end* może znajdować się dowolnie dużo instrukcji; mogą to być także inne instrukcje warunkowe. Gdy jest tam tylko jedna instrukcja, słowa *begin* i *end* można pominąć.

Działanie instrukcji *if-then* wyjaśnia graf na rysunku 7.1. Jak już powiedziano w p. 7.2, wyrażenie relacyjne po *if* może przyjąć jedynie jedną z dwóch wartości logicznych: *True*, gdy jest prawdziwe, lub *False*, gdy nie jest prawdziwe. W przypadku, gdy przyjmie wartość *True*, zostaną wykonane wewnętrzne instrukcje znajdujące się w klamrze *begin-end* po słowie *then*. W przeciwnym przypadku, gdy wyrażenie warunkowe ma wartość *False*, żadna instrukcja nie zostanie wykonana.



Rys. 7.1. Graf działania instrukcji warunkowej *if-then*

W przykładzie 7.1 pokazano prosty program wykorzystujący instrukcję *if-then*. Program ma zmienić znak liczby *X*, ale tylko wtedy, gdy jest ona ujemna. Warunek $X < 0$ po słowie *if* sprawdza, czy liczba jest ujemna. Jeżeli to prawda, wewnętrzna instrukcja przypisania wpisana po słowie *then* zmieni znak *X* na dodatni. Brak tutaj słów *begin-end*, bo po słowie *then* jest tylko jedna instrukcja.

Przykład 7.1. Program zmieniający znak liczby ujemnej

```
program Ex7_1;
uses Crt;
var X:Integer;
begin
  Clrscr;
  Write('Podaj liczbe typu Integer: ');
  Readln(X);
  Writeln('Czytana wartosc X: ',X:6);
  if X<0 then X:=-X;           {instrukcja warunkowa}
  Writeln('Koncowa wartosc X: ',X:6);
  Readln;
end.
```

7.3. Instrukcja warunkowa if-then-else

Ta postać instrukcji warunkowej ma następującą składnię:

```
if wyrażenie_relacyjne then begin
    instrukcja_1;
    instrukcja_2;
    {-----}
    instrukcja_n;
end
else begin
    instrukcja_1;
    instrukcja_2;
    {-----}
    instrukcja_n;
end;
```

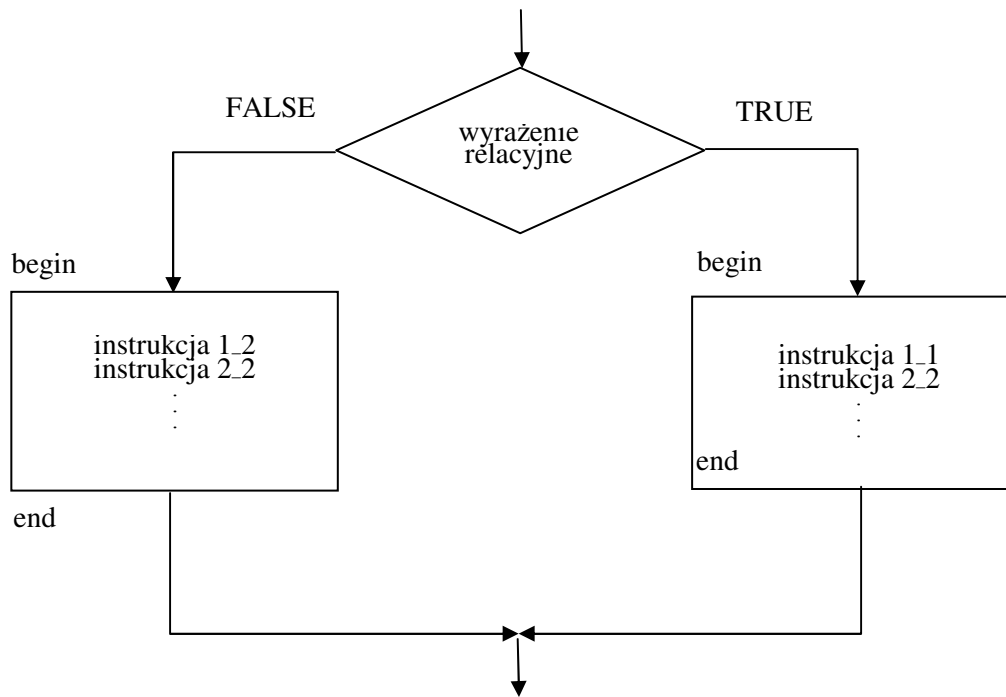
Zwróćmy uwagę, że przed słowem *else* nie można postawić średnika! Gdy to się zdarzy, podczas kompilacji wystąpi błąd z komunikatem:

```
Error 113: Error in statement.
```

Jak można się domyśleć ze znaczenia słów *if-then-else* (po polsku: jeżeli-to-w przeciwnym przypadku), instrukcja ta uwzględnia także alternatywne działanie, wykonywane wtedy, gdy warunek po *if* przyjmuje wartość *False*. Ilustruje to graf działania na rysunku 7.2.

Jako ilustracja działania instrukcji *if-then-else* może służyć przykład 7.2. Pokazano tam program do obliczania rozwiązań równania kwadratowego typu: $ax^2 + bx + c = 0$.

Po wprowadzeniu danych, którymi są parametry *A*, *B*, *C*, program oblicza wartość wyrażenia *Delta*, równego $b^2 - 4ac$. (Wyrażenie to w tekście programu zapisano jako $B*B - 4*A*C$, ponieważ nie ma w Turbo Pascalu operatora potęgowania; poza tym nie wolno w iloczynie pominąć symbolu mnożenia). Uruchamiana następnie instrukcja warunkowa sprawdza znak zmiennej *Delta*. Jeżeli jest ona nieujemna, następuje obliczenie pierwiastków równania i wyniki zostają zapisane w zmiennych *X1*, *X2*; w przeciwnym przypadku, gdy *Delta* jest ujemna, na ekranie pojawia się tylko komunikat o braku pierwiastków rzeczywistych. Zauważmy, że słowa *begin-end* po *else* pominięto, ponieważ występuje tam tylko jedna instrukcja.



Rys. 7.2. Graf działania instrukcji if-then-else

Przykład 7.2. Program do znajdowania rozwiązań równania kwadratowego

```

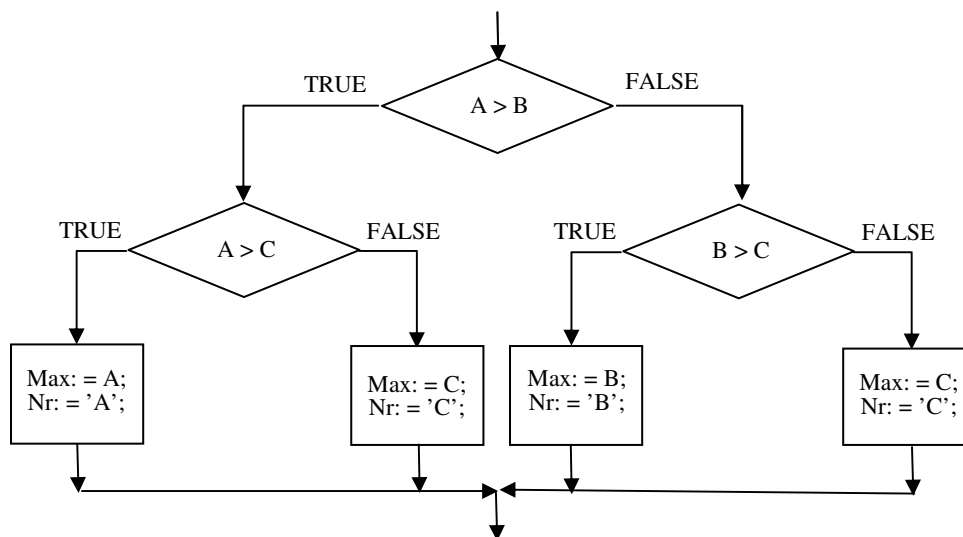
Program Ex7_2;
{Znajduje rozwiązania równania kwadratowego.}
uses Crt;
var A,B,C,Delta,X1,X2:Real;
begin
  Write('Podaj A,B,C: ');
  Readln(A,B,C);
  Delta:=B*B-4*A*C;
  if Delta>=0 then begin
    X1:=(-B+Sqrt(Delta))/(2*A);
    X2:=(-B-Sqrt(Delta))/(2*A);
    Writeln('X1=',X1);
    Writeln('X2=',X2);
  end
  else Write('Nie ma pierwiastka rzeczywistego. ');
  Readln;
end.

```

7.4. Instrukcje warunkowe zagnieżdżone

Zagnieżdżone instrukcje warunkowe to takie, gdzie wewnątrz jednej instrukcji warunkowej występuje inna instrukcja warunkowa. Te zagnieżdżone struktury mogą być bardzo złożone i zawierać wiele słów *if*, *then*, oraz *else*. Analiza ich działania jest trudna. Pomaga w tym stosowanie odpowiednich wcięć w tekście programu; pomocne są też grafy działania.

Dla ilustracji zastosowania instrukcji warunkowej zagnieżdżonej, pokażemy przykład procesu znajdowania tej spośród trzech zmiennych, która aktualnie przechowuje największą wartość. Jedną z metod rozwiązania tego zadania przedstawia graf na rysunku 7.3. Metodę zaimplementowano w programie pokazanym w przykładzie 7.3, posługując się zagnieżdżoną instrukcją warunkową.



Rys. 7.3. Graf działania metody poszukiwania największej z trzech zmiennych

Przykład 7.3. Program do wykrywania największej z trzech zmiennych

```

Program Ex7_3;
uses Crt;
var Max,A,B,C:Shortint;
    Nr:Char;
begin
  Clrscr;
  A:=60;
  B:=10;
  C:=125;
  if A>B then if A>C then begin
    Max:=A;
    Nr:='A';
  end
  else begin
    Max:=C;
    Nr:='C';
  end
  else if B>C then begin
    Max:=B;
    Nr:='B';
  end
  else begin
    Max:=C;
    Nr:='C';
  end;
  Write('Szukana zmienna: ',Nr,'=',Max);
  Readln;
end.
  
```

7.5. Wyrażenia logiczne jako alternatywa zagnieżdżenia

Jak już powiedziano w p. 7.1, wyrażenia relacyjne mogą mieć jako całość jedynie jedną z dwóch wartości: *True* lub *False*. Dlatego można je łączyć operatorami logicznymi *and*, *or*, *xor*, lub negować operatorem *not*. Często pozwala to na uniknięcie zagnieżdżenia instrukcji warunkowych i uzyskanie bardziej czytelnej postaci programu.

Jeżeli na przykład zmienna *A* jest większa od *B* i jednocześnie większa od *C*, to mamy pewność, że właśnie ta zmienna aktualnie przechowuje największą wartość. Można to zapisać w postaci prostej instrukcji *if-then*, w której obie relacje większości będą połączone operatorem *and*. Podobne rozumowanie można przeprowadzić w stosunku do pozostałych zmiennych, tj. *B* i *C*. W ten sposób uzyskujemy metodę rozwiązania zadania, w której wykorzystuje się sekwencję trzech prostych instrukcji warunkowych:

```
if (A>B) and (A>C) then Max:=A;
if (B>A) and (B>C) then Max:=B;
if (C>A) and (C>B) then Max:=C;
```

Tę metodę wykorzystuje zmodyfikowana wersja programu, pokazana w przykładzie 7.4. Dla uproszczenia zapisu instrukcji *if* zastosowano tutaj pomocniczą zmienną logiczną *B*, której przypisuje się aktualną wartość odpowiedniego warunku. Po słowie *then* wykonywane są dwie instrukcje. Oprócz zapamiętania w *Max* wartości maksymalnej, w *Z* przechowuje się nazwę zmiennej, w której wartość tę wykryto.

Przykład 7.4. Program do wykrywania największej z trzech zmiennych – wersja druga

```
program Ex7_4;
uses Crt;
var Max,A,B,C:Shortint;
    Z:Char;
    W:Boolean;
begin
  Clrscr;
  A:=60;
  B:=100;
  C:=15;
  W:=(A>B) and (A>C);
  if W then begin
    Max:=A;
    Z:='A';
  end;
  W:=(B>A) and (B>C);
  if W then begin
    Max:=B;
    Z:='B';
  end;
  W:=(C>A) and (C>B);
  if W then begin
    Max:=C;
    Z:='C';
  end;
  Write('Szukana zmienna: ',Z,'=',Max);
  Readln;
end.
```

7.6. Instrukcja wyboru *case*

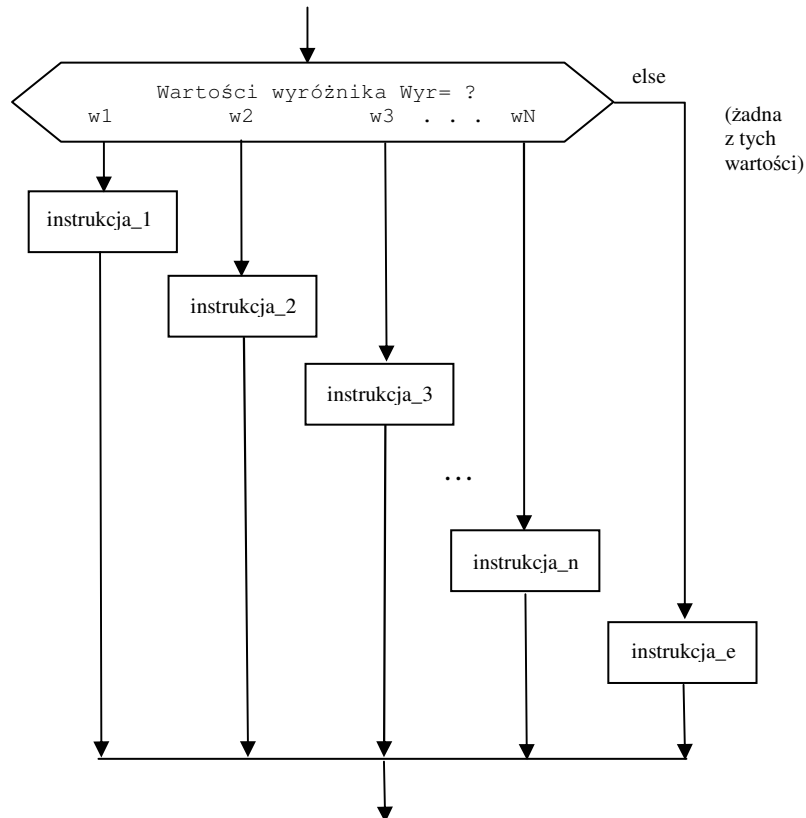
Instrukcja ta ma następującą ogólną postać:

```
case Wyróżnik of  
  ww_1: begin sekwencja_instrukcji_1 end;  
  ww_2: begin sekwencja_instrukcji_1 end;  
  ww_3: begin sekwencja_instrukcji_1 end;  
  {-----}  
  ww_N: begin sekwencja_instrukcji_1 end;  
  else begin sekwencja_instrukcji_e end;  
end;
```

Wyróżnik jest wyrażeniem typu *porządkowego* (np całkowitego, znakowego), które steruje wykonaniem instrukcji, natomiast symbole *ww_1..ww_N* oznaczają konkretne wartości wyróżnika (wolno tu wpisać tylko jawne stałe liczbowe lub znakowe, lub też nazwy stałych definiowanych). Do wykonania zostaje wybrana tylko jedna sekwencja instrukcji – ta, która odpowiada aktualnej wartości wyróżnika. Na przykład, gdy *Wyróżnik=ww_3*, zostanie wykonany ciąg instrukcji:

```
ww_3: begin sekwencja_instrukcji_3 end;
```

Jeżeli wyróżnik nie przyjął żadnej z wpisanych w instrukcji wartości *ww_1..ww_N*, to zostanie wykonana sekwencja instrukcji po słowie *else*. Trzeba pamiętać, że instrukcja kończy się słowem *end*, które jest niezbędne. Natomiast słowo *else* i występującą po nim sekwencję instrukcji można pominąć, jeżeli nie jest potrzebna w programie. Można też pominąć te klamry *begin–end*, w których występują pojedyncze instrukcje. Działanie instrukcji *case* przedstawia rysunek 7.4.



Rys. 7.4. Graf działania instrukcji wyboru *case*

W przykładzie 7.4 pokazano program, pełniący funkcję sześciodziałaniowego kalkulatora, umożliwiającego wykonywanie wybranej operacji dwuargumentowej na liczbach całkowitych. Mamy do wyboru sześć działań: dodawanie, odejmowanie, mnożenie, dzielenie, dzielenie całkowite oraz znajdowanie reszty z dzielenia całkowitego. Do wybierania rodzaju działania zastosowano instrukcję wyboru *case*. Jej wyróżnikiem jest zmienna całkowita *Operacja*.. Każdemu z sześciu dostępnych działań jest przypisana odpowiednia wartość wyróżnika – od 1 do 6. Po wprowadzeniu wartości spoza tego zakresu, wykona się instrukcja po słowie *else*, informując użytkownika, że nie ma takiej operacji.

Przykład 7.4. Programowa realizacja sześciodziałaniowego kalkulatora dla liczb całkowitych

```
program Ex7_4;
{Wykonuje wybrany rodzaj operacji
 na dwóch liczbach całkowitych.}
uses Crt;
var X1,X2,Operacja:Integer;
begin
  Clrscr;
  Write('Podaj 1. argument: ');
  Readln(X1);
  Write('Podaj 2. argument: ');
  Readln(X2);
  Writeln('Wybierz rodzaj operacji:');
  Writeln('<1>Suma      <2>Roznica      <3>Iloczyn');
  Writeln('<4>Iloraz   <5>Iloraz całkowity <6>Reszta z dzielenia');
  Readln(Operacja);
  Clrscr;
  Gotoxy(30,12);
  case Operacja of
    1: Write(X1, ' + ', X2, ' = ', X1+X2);
    2: Write(X1, ' - ', X2, ' = ', X1-X2);
    3: Write(X1, ' * ', X2, ' = ', X1*X2);
    4: if X2<>0 then Write(X1, ' / ', X2, ' = ', X1/X2:0:3)
        else Write('Nie dziel przez 0. ');
    5: Write(X1, ' div ', X2, ' = ', X1 div X2);
    6: Write(X1, ' mod ', X2, ' = ', X1 mod X2);
    else Write('Nie ma takiej operacji!');
  end;
  Readln;
end.
```