Zbiór zadań z informatyki. Nie tylko dla maturzystów

Autor: Monika Niedziela
ISBN: 83-7361-991-7
Format: B5, stron: 336

Zestaw zadań przydatny każdemu uczniowi i maturzyście

- Dokładnie omówienie zagadnień związanych z algorytmami, programowaniem i bazami danych
- Przykłady rozwiązań
- Zadania

Znacznie łatwiej i szybciej opanuje się poznawane zagadnienie, gdy jest to powiązane z rozwiązywaniem praktycznych przykładów. O wiele prościej jest zrozumieć znaczenie symboli i wzorów, gdy zastosuje się je w zadaniu – wtedy okaże się, że „nie taki diabeł straszny". Opanowanie rzeczywistego zastosowania wiadomości teoretycznych jest szczególnie istotne w przypadku przedmiotów mocno powiązanych z wiedzą praktyczną. Takim przedmiotem niewątpliwie jest informatyka.

„Zbiór zadań z informatyki. Nie tylko dla maturzystów" to książka, dzięki której poznasz praktyczne zastosowanie wiadomości dotyczących systemów liczbowych, algorytmów, zasad programowania i możliwości wykorzystania baz danych. Każde zagadnienie jest przedstawione od strony teoretycznej i zilustrowane przykładami. Rozwiązując kolejne zadania, nauczysz się tego, co w informatyce najistotniejsze – myślenia algorytmicznego i obiektowego. Opanujesz metody przeliczania wartości pomiędzy systemem binarnym, ósemkowym, szesnastkowym i dziesiętnym, poznasz zastosowanie różnych typów danych, przedstawisz rzeczywiste problemy w sposób algorytmiczny, napiszesz proste programy i przekonasz się, jakie możliwości oferuje baza danych Access.

- Systemy liczbowe
- Typy i struktury danych
- Tworzenie algorytmów
- Podstawowe zasady programowania w języku Pascal
- Programowanie strukturalne i obiektowe
- Projektowanie relacyjnych baz danych
- Wprowadzanie i modyfikowanie danych
- Manipulowanie danymi za pomocą języka SQL

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Rozdział 6.

Wprowadzenie do algorytmiki

Pojęcia podstawowe

Powszechnie przez algorytm rozumie się schemat mechanicznego rozwiązania określonego problemu. Posługując się bardziej naukowymi pojęciami, można powiedzieć, że **algorytm to skończony ciąg operacji na obiektach, ze ściśle ustalonym porządkiem wykonania, dający możliwość realizacji zadania z określonej klasy**.

Algorytm w sensie informatycznym opiera się na określonych **danych** i daje oczekiwane **wyniki**. Ponadto powinien mieć następujące cechy:

- **Poprawność** — dla każdego zestawu poprawnych danych wejściowych algorytm powinien dawać poprawne wyniki.
- **Skończoność** — dla każdego zestawu poprawnych danych wejściowych algorytm powinien dawać wyniki w skończonej liczbie kroków.
- **Określoność** — algorytm nie może pozostawiać wątpliwości co do wyboru kolejnej operacji (najlepszym sprawdzianem tej cechy jest łatwość implementacji algorytmu).
- **Efektywność** (**sprawność**) — algorytm powinien prowadzić do rozwiązania jak najniższym kosztem, czyli w jak najmniejszej liczbie kroków. W informatyce cecha ta sprowadza się do optymalizacji zajętości pamięci przez struktury danych wykorzystywane w algorytmie oraz do optymalizacji złożoności obliczeniowej (liczby wykonanych operacji).
- **Ogólność** (**uniwersalność**) — algorytm powinien rozwiązywać nie tylko jedno szczególne zadanie, ale całą klasę zadań.

Sposoby zapisu algorytmów

Algorytm można zapisać w postaci:

- ❖ listy kroków,
- ❖ schematu blokowego,
- ❖ pseudokodu.

Każdy algorytm powinien posiadać **specyfikację**, w której określamy **dane**, z jakich algorytm korzysta, **wyniki**, które powinien dawać, oraz **zmienne pomocnicze** niezbędne do realizacji algorytmu. Uniwersalny algorytm operuje nie na stałych (liczbowych, tekstowych), lecz na **zmiennych**. Zmienna to po prostu pojemnik na dane. Może być oznaczona dowolną literą lub łańcuchem znaków (np. `a`, `bok_a`).

W specyfikacji poza wypisaniem nazw zmiennych należy określić ich funkcję w algorytmie oraz typ (liczba całkowita, liczba rzeczywista, łańcuch znaków, wartość logiczna). Czasem należy także określić warunki, które muszą spełniać dane wejściowe.

W algorytmach spotykamy się także z **wyrażeniami**, w skład których poza **zmiennymi** i **stałymi** wchodzą operatory (`+`, `-`, `*`, `/`) i funkcje matematyczne.

Algorytm w postaci listy kroków jest intuicyjny i czytelnik najlepiej go opanuje, analizując kolejne przykłady. Algorytm w postaci **pseudokodu** ma strukturę bardziej złożoną, jego budowę czytelnik pozna czytając komentarze. Każdy z algorytmów zastanie zapisany także w postaci **schematu blokowego**, którego elementy zostaną omówione pod przykładami (rysunek 6.4).

Należy zwrócić uwagę, że budując wyrażenie warunkowe, korzystamy z **operatorów relacyjnych** (`=`, `<`, `>`, `<=`, `>=`, `<>`). W algorytmach stosuje się także **instrukcję przypisania** o postaci `:=`. Polega ona na podstawieniu wartości znajdującej się po prawej stronie pod zmienną z lewej strony, np. `a:=b+5` oznacza podstawienie pod zmienną *a* wartości zmiennej *b* zwiększonej o 5, tzn. jeśli $b = 7$, to $a = 12$.

Zadania znajdujące się pod przykładami najlepiej rozwiązać na zasadzie analogii.

Przykład 6.1. Algorytm liniowy

Sformułuj algorytm wyliczający pole prostokąta o bokach podanych przez użytkownika.

Sposób rozwiązania

Długości boków podane przez użytkownika zapamiętamy w zmiennych *a* i *b*. Następnie wyliczymy *p* jako iloczyn *a* i *b*. Wyliczone *p* wyświetlimy na ekranie. Będzie to prosty **algorytm liniowy**.

Rozwiązanie

Poniżej przedstawiono ten sam algorytm zapisany w postaci listy kroków, pseudokodu i schematu blokowego (rysunek 6.1).

Specyfikacja algorytmu:

Dane:

a — pierwszy bok prostokąta, liczba rzeczywista większa od 0;

b — drugi bok prostokąta, liczba rzeczywista większa od 0.

Wynik:

p — pole prostokąta, liczba rzeczywista.

Lista kroków:

Krok 1. Wczytaj wartość pierwszego boku. Zapisz ją pod zmienną *a*.

Krok 2. Wczytaj wartość drugiego boku. Zapisz ją pod zmienną *b*.

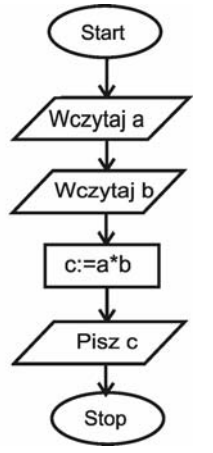
Krok 3. Oblicz *p* jako iloczyn *a* i *b*.

Krok 4. Wyświetl *p*.

Pseudokod:

```
Program kwadrat              {nagłówek programu}
zmienne
   a, b, c: rzeczywiste      {deklaracje zmiennych}
początek                     {początek programu właściwego}
  czytaj(a)                  {wczytanie danych}
  czytaj(b)
  c:=a*b                     {instrukcja przypisania}
  pisz(c)                    {wyświetlenie wyniku}
koniec                       {koniec programu właściwego}
```

Schemat blokowy:



Rysunek 6.1.
Schemat blokowy do przykładu 6.1

Przykład 6.2. Algorytm z warunkiem

Sformułuj algorytm zwracający mniejszą z dwóch liczb podanych przez użytkownika.

Sposób rozwiązania:

Wartości podane przez użytkownika zapamiętamy w zmiennych *a* i *b*. Następnie sprawdzimy, czy $a < b$ — jeśli tak, to w zmiennej *c* zapamiętamy wartość *a*; jeśli nie, to w zmiennej *c* zapamiętamy wartość *b*. Następnie wyświetlimy *c* na ekranie. Będzie to **algorytm z warunkami** (rozgałęzieniami), gdyż w zależności od spełnienia bądź niespełnienia warunku ($a < b$) różne jest jego działanie.

Rozwiązanie

Podobnie jak w przykładzie 6.1, poniżej przedstawiono ten sam algorytm zapisany w postaci listy kroków, pseudokodu i schematu blokowego (rysunek 6.2).

Specyfikacja algorytmu:

Dane:

a, *b* — liczby podane przez użytkownika, liczby całkowite.

Wynik:

c — mniejsza z liczb *a* i *b*, liczba całkowita.

Lista kroków:

Krok 1. Wczytaj wartość pierwszej liczby. Zapisz ją pod zmienną *a*.

Krok 2. Wczytaj wartość drugiej liczby. Zapisz ją pod zmienną *b*.

Krok 3. Jeśli $a < b$, to do *c* przypisz wartość *a* i przejdź do kroku 5.

Krok 4. Do *c* przypisz wartość *b*.

Krok 5. Wypisz wartość *c*.

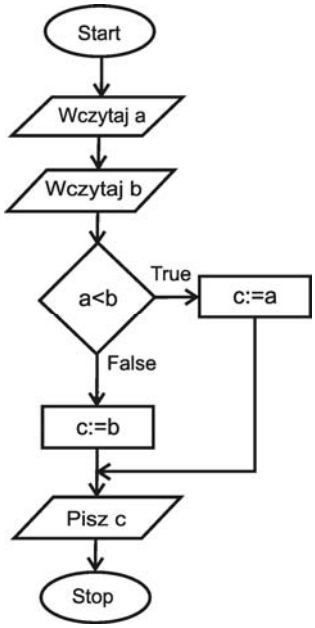
Krok 6. Koniec.

Pseudokod:

```
Program mniejsza
zmienne
   a, b, c: całkowite
początek
  czytaj(a,b)
  jeżeli a<b to c:=a   {sprawdzenie warunku oraz działanie wykonywane w przypadku
                        gdy warunek jest prawdziwy}
     w przeciwnym przypadku c:=b    {działanie wykonywane, gdy warunek nie jest prawdziwy}
  pisz(c)
koniec
```

Schemat blokowy:

Rysunek 6.2.
Schemat blokowy do przykładu 6.2



Przykład 6.3. Algorytm z zagnieżdżonymi warunkami

Sformułuj algorytm sprawdzający, czy z boków o długościach podanych przez użytkownika można utworzyć trójkąt.

Sposób rozwiązania:

Długości boków podane przez użytkownika zapamiętamy w zmiennych *a*, *b*, *c*. Aby z trzech odcinków można było zbudować trójkąt, żadna z sum dwóch dowolnych boków nie może być mniejsza od trzeciego boku. Posłużymy się zmienną logiczną `trojkat`, której na początku przypiszemy wartość `fałsz` (`false`). Jeśli będzie zachodziła każda z zależności: $a+b > c$, $b+c > a$, $c+a > b$, to zmiennej `trojkat` przypiszemy wartość `prawda` (`true`) i wyświetlimy komunikat, że z podanych boków da się zbudować trójkąt. Algorytm w postaci pseudokodu i schematu blokowego będzie **algorytmem z zagnieżdżonymi warunkami**.

Rozwiązanie

Podobnie jak w poprzednich przykładach, przedstawiono ten sam algorytm zapisany w postaci listy kroków, pseudokodu i schematu blokowego (rysunek 6.3).

Specyfikacja algorytmu:

Dane:

a, *b*, *c* — liczby rzeczywiste podane przez użytkownika, długości odcinków.

Wynik:

`trojkat` — informacja, czy z odcinków o podanych długościach można utworzyć trójkąt, wartość logiczna.

Lista kroków:

Krok 1. Wczytaj wartości zmiennych *a*, *b*, *c*.

Krok 2. Zmiennej `trojkat` przypisz wartość `false`.

Krok 3. Jeśli `a+b>c`, to jeśli `a+c>b`, to jeśli `c+b>a`, to zmiennej `trojkat` przypisz wartość `true`.

Krok 4. Jeżeli zmienna `trojkat` ma wartość true, to wypisz `'Z podanych boków można zbudować trójkąt'`; w przeciwnym przypadku wypisz `'Z podanych boków nie można zbudować trójkąta'`.

Krok 5. Koniec.

Komentarz:

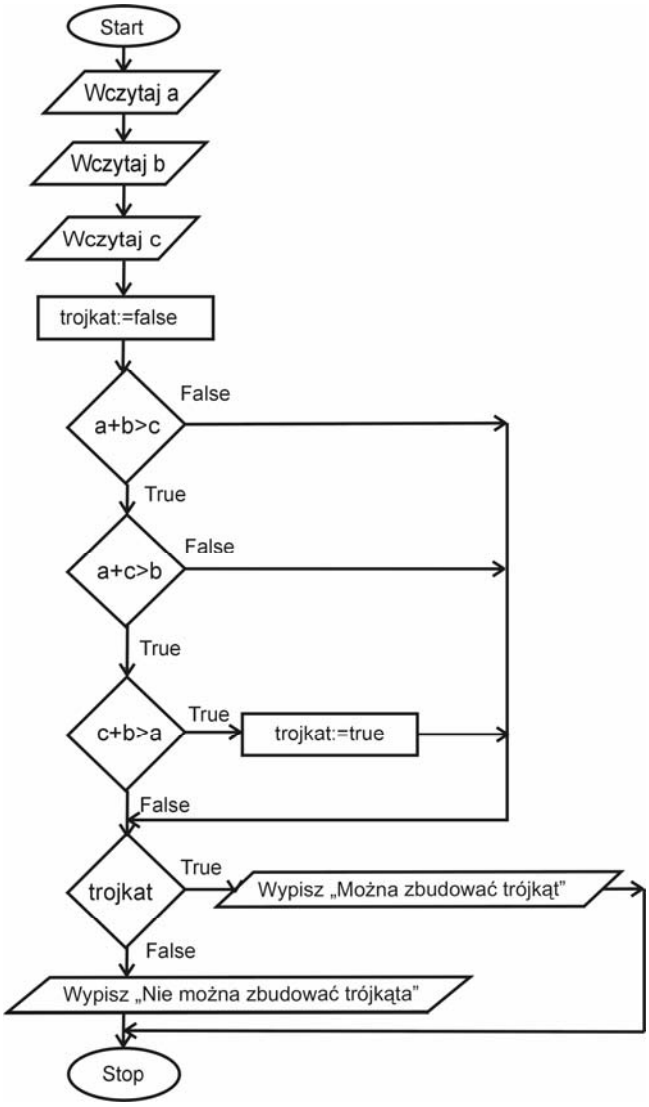
Najpierw sprawdzamy warunek `a+b>c` — jeśli zachodzi, to sprawdzamy, czy `a+c>b` — jeśli tak, to sprawdzamy, czy `c+b>a` — jeśli tak, to `trojkat :=true`. Jak widać, w przypadku gdy z podanych boków można zbudować trójkąt, sprawdzane są wszystkie 3 warunki i wszystkie są prawdziwe.

Schemat blokowy (rysunek 6.3).

Pseudokod:

```
Program czytrojkat
zmienne
   a, b, c: rzeczywiste
   trojkat: logiczna
początek
   czytaj(a,b,c)
   trojkat:=false
   jeżeli a+b>c to
      początek
      jeżeli a+c>b to
             jeżeli b+c>a to trojkat:=true
      koniec
   jeżeli trojkat to pisz('Można zbudować trójkąt')
     w przeciwnym wypadku pisz('Nie można zbudować trójkąta')
koniec
```

Rysunek 6.3.
Schemat blokowy do przykładu 6.3

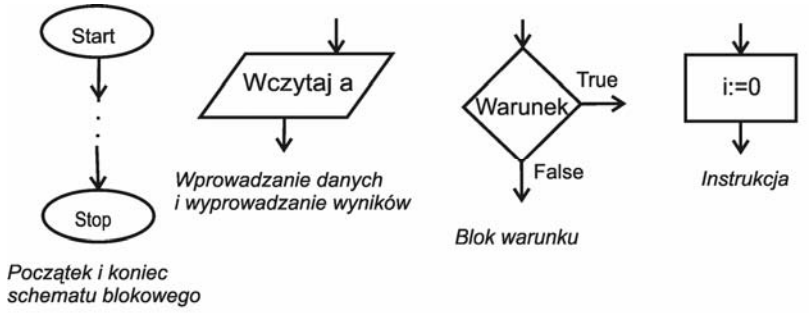


Elementy schematu blokowego

Reguły, według których należy budować schematy blokowe (rysunek 6.4):

- ❖ Każdy algorytm ma tylko jeden początek.
- ❖ Wszystkie drogi muszą się kończyć na bloku *Stop*.
- ❖ Wszystkie bloki muszą mieć zgodną z ich budową liczbę wejść i wyjść.
- ❖ Każda linia musi być opatrzona strzałką informującą o kierunku przepływu informacji.

Rysunek 6.4.
Elementy schematu blokowego



Zadania do samodzielnego rozwiązania

Zadanie 6.1.

Napisz algorytm w postaci listy kroków i pseudokodu, wyliczający pole kwadratu o boku podanym przez użytkownika.

Zadanie 6.2.

Napisz algorytm w postaci listy kroków, według którego sporządzasz swoją ulubioną potrawę.

Zadanie 6.3.

Napisz algorytm w postaci schematu blokowego i pseudokodu, zwracający wartość bezwzględną liczby podanej przez użytkownika.

Zadanie 6.4.

Napisz algorytm w postaci listy kroków i schematu blokowego, informujący, czy liczba podana przez użytkownika jest większa, mniejsza czy równa zero.

Zadanie 6.5.

Zmodyfikuj algorytm z przykładu 6.1 tak, by w przypadku wprowadzenia danych mniejszych lub równych zero nie było wyliczane pole, lecz by był wyświetlany odpowiedni komunikat.

Zadanie 6.6.

Napisz algorytm w postaci listy kroków, pseudokodu i schematu blokowego, informujący, czy liczba podana przez użytkownika jest liczbą parzystą czy nieparzystą. Użyj operatora `mod`, który zwraca resztę z dzielenia całkowitego. Np. `5 mod 2 = 1`, `4 mod 2 = 0`.

Zadanie 6.7.

Utwórz w postaci listy kroków wraz ze specyfikacją algorytm wyliczający pierwiastek równania liniowego o postaci $ax+b = 0$; gdzie a, b podaje użytkownik. Weź pod uwagę fakt, że użytkownik może podać $a = 0$ lub $b = 0$.

Zadanie 6.8.

Narysuj w postaci schematu blokowego wraz ze specyfikacją algorytm wyliczający pierwiastki równania kwadratowego o postaci $ax^2+bx+c = 0$; gdzie a, b, c podaje użytkownik.

Zadanie 6.9.

Rozbuduj algorytm z przykładu 6.3. Nowy algorytm, w przypadku odcinków, z których można utworzyć trójkąt, ma liczyć jego pole według wzoru Herona:

$$S = \sqrt{p(p-a)(p-b)(p-c)}\text{, gdzie: } p = \frac{a+b+c}{2}.$$