Algorytmy + struktury danych = programy

Niklaus Wirth

Algorytm = logika + sterowanie

Robert Kowalski

1 Algorytm – komputer – program

1.1 Algorytm

Algorytm to opis sposobu postępowania prowadzącego do rozwiązania problemu.

Nazwa *algorytm* pochodzi od przydomka matematyka perskiego Abu Abdullaha Muhammada ibn Musy **al-Chuwarizmiego** (żył na przełomie VIII i IX wieku).

Przykładem algorytmu przepis z książki kucharskiej. Opisuje on jakie składniki są potrzebne do przygotowania dania oraz omawia krok po kroku jakie czynności należy wykonać aby uzyskać oczekiwany rezultat.

Innym przykładem algorytmu jest algorytm obliczania największego wspólnego dzielnika dwóch liczb dodatnich.

Możemy zapisać go na wiele sposobów, np. jednym zdaniem: *Dopóki obie liczby są różne odejmuj od większej mniejszą*.[1]

Przykładowe obliczenia:

$$(8, 11) \to (8, 3) \to (5, 3) \to (2, 3) \to (2, 1) \to (1, 1).$$

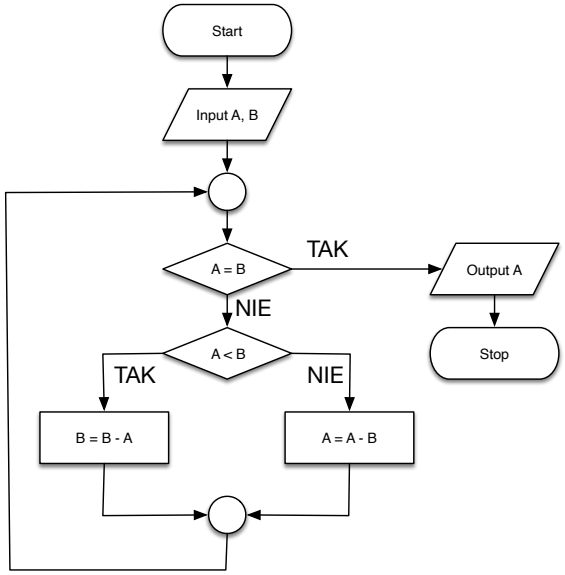
Możemy również wyrazić go schematem blokowym przedstawionym na rysunku 1.1.

Możemy zapisać go w tzw. pseudokodzie:

while $A \neq B$ **do**
  if $A < B$ **then**
    $B \Leftarrow B - A$
  else
    $A \Leftarrow A - B$
  end if
end while

Albo w języku programowania C:

[1] Można wyliczyć to efektywniej stosując operację reszty z dzielenia zamiast odejmowania.



Rysunek 1.1: Algorytm obliczania największego wspólnego dzielnika.


```
while(a != b)
  if(a < b)
    b = b - a;
  else
    a = a - b;
```

Na naszym kursie będziemy do zapisywania algorytmów używać języka C.

1.2 Komputer

Komputer jest urządzeniem, które automatycznie przetwarza dane. Przetwarzanie danych odbywa się zgodnie z algorytmem. Aby przetwarzać dane, komputer musi być wyposażony w jednostkę centralną, która wykonuje operacje. Same operacje i dane na których one działają są przechowywane w pamięci komputera.

1.3 Język programowania

Aby było możliwe wykonanie algorytmu przez komputer, musi być on zapisany w języku programowania.

Język programowania jest językiem formalnym o ściśle ustalonej składni oraz semantyce (znaczeniu) instrukcji w nim wyrażanych.

Składnię zapisuje się często w postaci gramatyki bezkontekstowej. Składa się ona z tzw. produkcji (reguł) mających postać:

```
<SYMBOL_DEFINIOWANY> ::= DEFINICJA
```

albo w postaci alternatywnych definicji:

```
<SYMBOL_DEFINIOWANY> ::=   DEFINICJA_1
                         | DEFINICJA_2
                         ...
                         | DEFINICJA_n
```

Oto przykładowa produkcja z gramatyki języka programowania C:

```
<parameter-list> ::=   <parameter-declaration>
                     | <parameter-list> , <parameter-declaration>
```

Powyższa reguła wyraża, że lista parametrów jest albo pojedynczą deklaracją parametru albo ciągiem deklaracji parametrów oddzielonych przecinkami.

Z kolei semantyka instrukcji, w mniej lub bardziej formalny sposób, opisuje jak wykonanie instrukcji zmienia stan obliczeń, tj. wartości wszystkich danych w programie.

Dla przykładu semantyka instrukcji:

```
zmienna = wyrażenie
```

stwierdza, że należy wyliczyć wartość wyrażenia po prawej stronie znaku przypisania a następnie zmienić wartość zmiennej po lewej stronie znaku przypisania na wartość przed chwilą wyliczoną.

1.3.1 Język maszynowy

Należy sobie zdawać sprawę, że komputer zna tylko jeden język programowania. Językiem tym jest jego język maszynowy. Zatem jednostka centralna komputera wykonuje jedynie programy w języku maszynowym tego komputera. Inny komputer (o innej jednostce centralnej) może mieć inny język maszynowy i dlatego nie będzie mógł wykonać programu przeniesionego z tego komputera.

Dla przykładu programy w języku maszynowym uruchamiane na komputerze PowerBook wyposażonym w procesor PowerPC G4 nie uruchomią się na komputerze MacBook wyposażonym w procesor Intel Core i5 (języki maszynowe tych procesorów są różne).

Język maszynowy składa się z podstawowych instrukcji jakie potrafi wykonać jednostka centralna.

Przykładem takie instrukcji jest dwubajtowa (szesnastobitowa) instrukcja `0xA90D`[2] mikroprocesora 6502 (można go znaleźć w ośmiobitowych komputerach Atari, Commodore albo Apple i konsoli Nintendo).

Instrukcja taka jest zupełnie nieczytelna dla człowieka. Z tego powodu przyjęło się zapisywać instrukcje w postaci mnemoników, tj. krótkich nazw zrozumiałych dla człowieka. Język programowania posługujący się mnemonikami zamiast liczbowymi kodami instrukcji maszynowych to język assemblera.

[2]Prefiks 0x oznacza, że po nim znajduje się ciąg cyfr szesnastkowych tj. cyfry od 0 do 9 i od A (10) do F (15). Zatem liczba ta jest zapisana w systemie szesnastkowym a jej dziesiętna wartość jest równa $10 \cdot 16^3 + 9 \cdot 16^2 + 0 \cdot 16^1 + 13 \cdot 16^0 = 43\,277$.

Instrukcję `0xA90D` można podzielić na dwa ośmiobitowe bajty `0xA9` i `0x0D`. Pierwszy bajt odpowiada instrukcji pobierania ośmiobitowej wartości liczbowej do ośmiobitowego rejestru akumulatora. Drugi bajt to argument dla instrukcji zapisanej pierwszym bajtem. Argumentem tym jest liczba `0x0D` czyli 13.

Instrukcję `0xA90D` zapisać można w postaci następującej instrukcji assemblera:

```
LDA #13
```

Mnemonik LDA jest skrótem od *LoaD Accumulator* (załaduj do akumulatora), natomiast znak `#` oznacza, że po nim znajduje się nie adres, spod którego należy pobrać ładowaną daną, ale ta dana (natychmiastowy tryb adresowania).

W wyniku wykonania tej instrukcji w rejestrze akumulatora zostanie umieszczona liczba 13.

Język maszynowy nazywa się językiem niskiego poziomu.

O architekturze komputerów i programowaniu w języku niskiego poziomu będziesz się uczył na trzecim semestrze na obowiązkowym kursie **Architektura komputerów i systemy operacyjne**.

1.3.2 Języki wysokiego poziomu

Aby móc przenosić programy między komputerami (uruchamiać ten sam program na różnych komputerach o różnych jednostkach centralnych) muszą one być napisane w językach programowaniach wysokiego poziomu.

Przykładem języka wysokiego poziomu jest język C, który będziemy poznawać na kolejnych wykładach i używać na laboratorium i ćwiczeniach. Podczas zajęć na naszym kierunku poznasz między innymi następujące języki programowania wysokiego poziomu: Ada, C, C++, Go, Haskell, Java, Javascript, Julia, Prolog, Python, SPARK, SQL.

1.4 Interpretacja i kompilacja

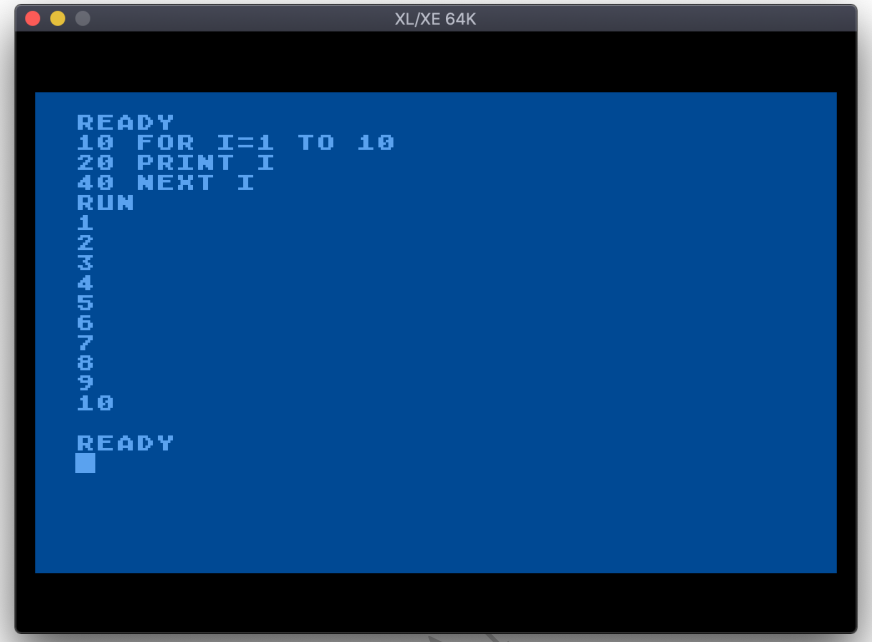
Komputer rozumie i potrafi wykonać tylko swój język maszynowy. Każdy inny język programowania jest dla niego niezrozumiały. Jak więc możliwe jest wykonywanie na komputerze programów napisanych w innych językach niż jego maszynowy?

1.4.1 Interpreter

Jednym z możliwych podejść do wykonywania programu w języku programowania innym niż język maszynowy komputera, jest napisanie w jego języku maszynowym programu, który będzie analizował i wykonywał instrukcje zadanego programu. Taki program nazywa się interpreterem.

Na rysunku 1.2 przedstawiono okno emulatora komputera Atari z interpreterem języka Basic. Wprowadzono do niego krótki program:

```
10 FOR I=1 TO 10
20 PRINT I
30 NEXT I
```



Rysunek 1.2: Okno emulatora komputera Atari 800 XL.

a następnie wykonano go poleceniem `RUN`.

W wyniku wykonania programu zostały na ekranie wydrukowane kolejne liczby od 1 do 10.

Należy pamiętać, że interpreter wielokrotnie analizuje te same instrukcje co spowalnia działanie programu. Dla przykładu instrukcja `PRINT I` jest analizowana tyle razy ile razy powtarza się wykonanie pętli `FOR-NEXT`.

1.4.2 Kompilator

Innym podejściem jest napisanie w języku maszynowym programu, który analizowałby cały zadany program i tłumaczyłby go na język maszynowy. Taki program tłumaczący nazywa się kompilatorem.

Program jest analizowany tylko raz a potem za każdym uruchomieniem jego przekładu wykonywane są tylko instrukcje maszynowe wykonujące dokładnie to co zapisano w oryginalnym programie. Dzięki temu programy skompilowane wykonują się bardzo szybko w porównaniu do programów interpretowanych.

Załóżmy, że `suma` i `składnik` są szesnastobitowymi zmiennymi o wartościach całkowitych z zakresu od $-32\,768$ do $32\,767$.

Instrukcja powiększająca wartość sumy o składnik:

```
suma = suma + składnik
```

może zostać przetłumaczona na następujący ciąg instrukcji mikroprocesora 6502:

```
LDA suma
CLC
ADC składnik   ; bit przeniesienia został wyzerowany
STA suma
LDA suma+1
ADC składnik+1 ; bit przeniesienia nie jest zerowany
STA suma+1
```

Poza instrukcją `LDA` wykorzystano w nim jeszcze instrukcje:

CLC czyszczenie (zerowanie) bitu przeniesienia[3] (ang. *CLear Carry*);

STA zapisanie akumulatora w pamięci (ang. *STore Accumulator*).

O gramatykach i kompilatorach będziesz się uczyć na piątym semestrze na obowiązkowym kursie **Języki formalne i techniki translacji**. Podczas laboratorium do tego kursu napiszesz własny kompilator dla zadanego przez prowadzącego języka programowania.

1.5 Przykład programu w języku C

Na rysunku 1.3 przestawiono schemat blokowy programu który czyta kolejne liczby i oblicza sumę wprowadzonych liczb dodatnich oraz sumę wprowadzonych liczb ujemnych. Czytanie liczb kończy się gdy wprowadzi się liczbę zero. Na koniec działania programu drukowane są wartości wyliczonych sum.

Źródła programu w języku C:

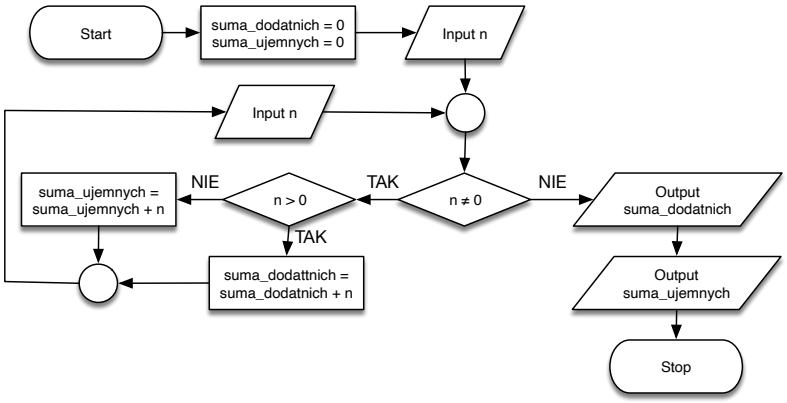
Listing 1.1: Suma dodatnich i suma ujemnych

```c
// sumy.c
//
// Program czyta liczby całkowite i liczy sumę liczb dodatnich
// oraz sumę liczb ujemnych. Aby zakończyć wpisywanie liczb
// należy wpisać liczbę zero.

#include <stdio.h>

int main(void)
{
  int suma_dodatnich = 0;
  int suma_ujemnych = 0;
  int n;
  printf("Wpisuj liczby całkowite.\n");
```

[3] Bit przeniesienia związany jest z algorytmem dodawania dwóch liczb zapisanych w systemie binarnym. Kiedy dodaje się dwa bity, to wynik może przekroczyć wartość 1 i wówczas bit przeniesienia przyjmuje wartość 1 i dodawany jest do sumy bitów na kolejnej pozycji. Jeśli suma nie przekracza wartości 1, to bit przeniesienia ma wartość 0. Przed rozpoczęciem dodawania zeruje się bit przeniesienia.



Rysunek 1.3: Schemat blokowy: suma dodatnich i suma ujemnych

```
  printf("Aby zakończyć wpisz liczbę zero.\n");
  scanf("%d", &n);
  while(n != 0)
  {
    if(n > 0)
      suma_dodatnich = suma_dodatnich + n;
    else
      suma_ujemnych = suma_ujemnych + n;
    scanf("%d", &n);
  }
  printf("Suma dodatnich wynosi: %d\n", suma_dodatnich);
  printf(" Suma ujemnych wynosi: %d\n", suma_ujemnych);
  return 0;
}
```


1.5.1 Kompilacja

Program sumujący liczby znajduje się w pliku `sumy.c`. Aby go skompilować wykonamy polecenie:

```
$ clang sumy.c -o sumy
```

W wyniku jego wykonania powstaje nowy plik o nazwie `sumy`. Jest to plik wykonywalny (można go uruchomić).

1.5.2 Uruchomienie

Aby uruchomić przekład programu w pliku `sumy` wykonujemy polecenie:

```
$ ./sumy
Wpisuj liczby całkowite.
Aby zakończyć wpisz liczbę zero.
```

```
1
-2
3
-4
0
Suma dodatnich wynosi: 4
 Suma ujemnych wynosi: -6
```