Dr inż. Robert Wójcik, p. 313, C-3, tel. 320-27-40

Zakład Podstaw Informatyki i Teleinformatyki
Instytut Cybernetyki Technicznej (I-6)
Instytut Informatyki, Automatyki i Robotyki
Politechniki Wrocławskiej

E-mail: **robert.wojcik@pwr.wroc.pl**
Strona internetowa: **sprocket.ict.pwr.wroc.pl/~wojcik**

Wykład 1

1. WSTĘP

1.1. Podstawowe definicje

Problem – zadanie do rozwiązania

Specyfikacja zadania – określenie danych wejściowych oraz wyników, które powinny być uzyskane, a także warunków jakie powinny one spełniać; może zawierać również związki pomiędzy danymi a wynikami; abstrakcyjny model.

Algorytm jest ciągiem czynności, które prowadzą do rozwiązania zadania.

Komputer – urządzenie elektroniczne służące do automatycznego przetwarzania danych według zadanego algorytmu.

Z punktu widzenia techniki komputerowej.

Algorytm – sposób przetwarzania informacji wejściowych (danych wejściowych) na informacje wyjściowe (wyniki), w *skończonej* liczbie kroków.

Algorytm definiuje:
- abstrakcyjne obiekty, na których wykonywane są działania – zmienne i stałe algorytmu;
- operacje realizujące cel algorytmu;
- kolejność wykonywania działań.

Program komputerowy – algorytm i struktury danych zapisane w odpowiednim języku programowania zrozumiałym przez komputer (np. w języku maszynowym procesora – ciąg liczb stanowiących rozkazy i dane dla procesora).

Język maszynowy jest trudno przyswajalny przez człowieka (ciągi bajtów reprezentujące instrukcje procesora). W praktyce algorytmy są zapisywane za pomocą instrukcji języków programowania wyższego poziomu *(języków algorytmicznych*), które udostępniają podstawowe elementy programowania strukturalnego (tj. struktury danych i konstrukcje języka umożliwiające ich przetwarzanie).

Kod źródłowy programu - zakodowana postać algorytmów stanowiących rozwiązanie problemu; tworzony pod dowolnym edytorem; jest najczęściej zapisywany za pomocą instrukcji takich języków programowania, jak: C, Pascal, Java, Fortran, Cobol.

Przed wykonaniem program źródłowy należy przetłumaczyć na postać zrozumiałą dla komputera czyli na *kod wynikowy*.

Kod wynikowy - kod pośredni w języku maszynowym, który jest zrozumiały dla komputera; ciąg rozkazów i danych procesora, zapisanych w pamięci komputera w kodzie binarnym.

Kod wynikowy jest przekształcany przez *program linkera* do postaci wykonywalnej.

Linker – program łączący kody wynikowe odpowiednich modułów programu w *kod wykonywalny*, który może być wielokrotnie uruchamiany w komputerze.

W praktyce linker łączy w jeden plik wykonywalny następujące elementy:
- ***pliki wynikowe*** (obiektowe), otrzymane w wyniku kompilacji modułów programu;
- ***standardowy kod startowy*** programu dla danego systemu operacyjnego;
- ***kody wynikowe funkcji***, wykorzystywanych w programie, zapisane w odpowiednich bibliotekach.

Kod wykonywalny - zawiera liczby, które są pobierane z pamięci komputera przez procesor i interpretowane jako rozkazy podlegające wykonaniu lub jako dane stanowiące argumenty rozkazów.

Translator – realizuje przekształcenie programu z postaci źródłowej na postać wynikową.

Rodzaje translatorów:
- kompilatory,
- interpretatory.

Kompilator – program przetwarzający kod źródłowy na kod wynikowy (kod pośredni w języku maszynowym, który jest zrozumiały dla komputera).

Interpretator – realizuje translację instrukcji naprzemiennie z ich wykonywaniem; przy zastosowaniu interpretatora każde wykonanie programu jest związane z jego ponowną translacją (np. Basic, SQL).

Plik – wydzielony, posiadający unikalną nazwę obszar pamięci (najczęściej dyskowej), w którym przechowywane są dane.

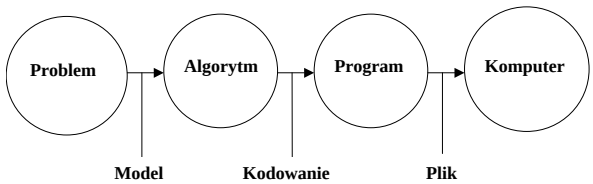
Z punktu widzenia języków programowania plik jest ciągiem danych o odpowiedniej strukturze (w najprostszym przypadku ciągiem bajtów). Pliki mogą być interpretowane jako znakowe lub binarne.

Każdy plik posiada rozmiar określony w bajtach.

1 Bajt [B] = 1 znak, 1 KB = 1024 B, 1MB = 1024 KB, 1 GB = 1024 MB.

Etapy rozwiązywania problemów z wykorzystaniem komputera:

- specyfikacja problemu,
- określenie danych wejściowych,
- określenie celu (wyniku końcowego),
- analiza problemu i wybór modelu,
- synteza algorytmu prowadzącego do rozwiązania,
- przedstawienie algorytmu:
  - w postaci opisu słownego,
  - w postaci listy kroków,
  - w postaci schematu blokowego (postać graficzna algorytmu),
  - za pomocą jednego z języków formalnych (np. UML).
- analiza poprawności rozwiązania,
- ocena efektywności algorytmu (złożoności obliczeniowej),
- kodowanie algorytmu w postaci instrukcji języka programowania (projekt programu komputerowego),
- zapis programu do pliku,
- kompilacja i usuwanie usterek,
- utworzenie wersji wykonywalnej,
- automatyczne wykonanie programu w komputerze,
- testowanie i analiza wyników.



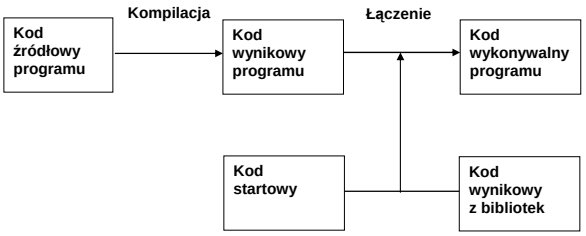
Etapy programowania

1. Utworzenie za pomocą edytora tekstu pliku źródłowego zawierającego algorytm zapisany w wybranym języku programowania, np. *program.pas* (program w języku Pascal), *program.cpp* (program w języku C++).

2. Kompilacja programu za pomocą kompilatora i utworzenie pliku wynikowego (obiektowego), np. *program.obj*.

3. Połączenie za pomocą linkera kodu wynikowego programu, kodów wynikowych funkcji bibliotecznych oraz kodu startowego w jeden plik wykonywalny, np. *program.exe*.

Kompilacja programów



Wykonywanie programów

DANE ⟶ **Programy (algorytmy)** ⟶ **WYNIKI**

System operacyjny komputera – zbiór programów sterujących pracą urządzeń wchodzących w skład systemu komputerowego i nadzorujących wykonywanie programów użytkowników.

Oprogramowanie użytkowe – programy uruchamiane pod kontrolą systemu operacyjnego.

Podczas projektowania algorytmów należy pamiętać, aby opracowywane algorytmy posiadały niską złożoność obliczeniową.

Czasowa złożoność obliczeniowa – określa liczbę elementarnych kroków obliczeniowych (tzw. operacji elementarnych, np. porównań, sumowań, itp.).

Pamięciowa złożoność obliczeniowa – określa rozmiar pamięci niezbędnej do wykonania programu.

W praktyce złożoność obliczeniową określa się za pomocą funkcji ograniczających z góry ponoszony nakład obliczeniowy, np. O(n), O(nlog(n)).

Algorytmy efektywne – posiadają wielomianową lub logarytmiczną złożoność obliczeniową.

Przetwarzanie sekwencyjne – wykonywanie instrukcji programów kolejno jedna za drugą.

Przetwarzanie współbieżne – wykonywanie instrukcji programów równocześnie na tym samym procesorze (z podziałem czasu procesora).

Przetwarzanie równoległe – wykonywanie instrukcji programów równocześnie na różnych procesorach.

1.2. Charakterystyka języka C/C++

Geneza języka

Język kompilowalny ogólnego stosowania. Został opracowany w 1972 roku przez Dennisa Ritchie'go z firmy Bell Labs w ramach prac prowadzonych nad opracowaniem systemu operacyjnego UNIX (większość oprogramowania tego systemu została napisana w języku C). Język C pochodzi od języka B opracowanego w 1970 roku przez Kena Thompsona dla pierwszego systemu Unix działającego na komputerze PDP-7. Z kolei język B pochodzi od języka BCPL stworzonego przez Martina Richardsa.

Język C został ukierunkowany na użyteczność w tworzeniu oprogramowania (np. w odróżnieniu od języka Pascal, który ma służyć nauce podstawowych zasad programowania). Jest on powszechnym narzędziem pracy programistów.

Język C++ jest rozwinięciem języka C uwzględniającym elementy programowania obiektowego.

Podstawowe elementy języka

Język C jest wyposażony w podstawowe konstrukcje sterujące wykorzystywane w programowaniu strukturalnym:

- grupowanie instrukcji ({ });
- podejmowanie decyzji (if);
- powtarzanie ze sprawdzaniem warunku zatrzymania na początku (while, for) lub na końcu pętli (do);
- wybór jednego z kilku możliwych przypadków (switch);
- funkcje (podprogramy).

Ponadto, w języku C występują wskaźniki, które służą do przechowywania adresów oraz wykonywania na nich różnych operacji. W oparciu o wskaźniki można zrealizować przekazywanie argumentów do funkcji. Możliwe są dwa sposoby przekazywania argumentów do funkcji:

- *przekazywanie przez wartości* – argumenty są kopiowane do funkcji; nie jest możliwa zmiana wartości argumentów w miejscu wywołania funkcji;

- *przekazywanie przez wskaźniki* – do funkcji są przesyłane adresy argumentów; funkcja ma możliwość zmiany wartości argumentów.

W języku C++ wprowadzono przekazywanie argumentów do funkcji za pomocą referencji. Również i w tym przypadku do funkcji są przekazywane adresy argumentów jednak dostęp do nich jest ułatwiony, gdyż wewnątrz funkcji można korzystać z nazw zmiennych (np. x), a nie ze wskaźników do zmiennych (np. *x).

Funkcje można wywoływać rekurencyjnie. Zmienne lokalne funkcji są zwykle „automatyczne", tzn. tworzone na nowo przy każdym jej wywołaniu. Definicje funkcji nie mogą być zagnieżdżone, natomiast zmienne można deklarować lokalnie wewnątrz funkcji zgodnie z regułami struktury blokowej (zmienne widziane lokalnie w bloku { }). Ponadto, funkcje programu mogą być zawarte w modułach zapisanych w różnych plikach i kompilowanych osobno.

Zmienne w języku C można podzielić na:

- zmienne wewnętrzne funkcji;
- zmienne zewnętrzne, ale znane w obrębie jednego pliku źródłowego;
- zmienne zewnętrzne o charakterze globalnym (widoczne w obrębie całego programu).

Zmienne wewnętrzne funkcji mogą być automatyczne lub statyczne (istnieją i zachowują dane pomiędzy kolejnymi wywołaniami funkcji).

W języku C możliwe są konwersje typów danych (tzw. rzutowanie zmiennych i wskaźników). Natomiast język ten nie dostarcza narzędzi wejścia i wyjścia umożliwiających automatyczny dostęp do plików, np. za pomocą funkcji READ (czytaj) lub WRITE (pisz). Są to w przypadku języka C mechanizmy wyższego poziomu dostępne poprzez odpowiednie funkcje standardowe, znajdujące się w bibliotekach dołączanych do programów.

Zalety języka C

- *Wysoka użyteczność* – język udostępnia większość funkcji sterujących, które są pożądane w teorii i praktyce programowania strukturalnego.

- *Wydajność i uniwersalność* – język charakteryzuje wysoki stopień precyzji, porównywany z asemblerem; prowadzi to do zwięzłości i małej objętości kodu wynikowego programów oraz ich dużej szybkości działania (język C jest wykorzystywany do tworzenia zaawansowanego oprogramowania, np. projektowania systemów operacyjnych (UNIX) lub kompilatorów).

- *Przenośność* – programy korzystające z bibliotek standardowych mogą być łatwo przenoszone pomiędzy różnymi platformami (np. Windows, Unix); zazwyczaj wymagana jest zmiana zawartości plików nagłówkowych, dołączanych do pliku głównego; problemy mogą pojawić się z programami wykorzystującymi specyficzne funkcje systemu operacyjnego lub realizujące bezpośredni dostęp do urządzeń sprzętowych.

- *Ukierunkowanie na programistę* – język zawiera wiele standardowych funkcji umożliwiających rozwiązywanie typowych zadań (np. sortowanie qsort).

Wady języka C

- Brak informacji o kolejności opracowywania wyrażeń będących argumentami funkcji (np. f(x = a-b, x+y)).

- Istnieje możliwość tworzenia skomplikowanych wyrażeń.

- Istnieje możliwość popełnienia wielu pomyłek podczas korzystania ze wskaźników.

1.3. Elementy projektowania i programowania algorytmów

W praktyce przedstawianie algorytmów w postaci słownej jest niewygodne, gdyż wymaga użycia dużej liczby słów i symboli, a przy tym może być niezbyt precyzyjne. Zapisywanie algorytmów od razu w określonym języku programowania, zwłaszcza przy skomplikowanych problemach, jest uciążliwe. Dlatego stosuje się etap pośredni w postaci opisu algorytmów w postaci graficznej za pomocą schematów blokowych i/lub za pomocą języków formalnych (np. UML). W dalszej części przedstawiono przykłady tworzenia programów umożliwiających rozwiązanie określonych zadań.

Problem 1. Znaleźć minimum spośród dwóch liczb całkowitych a i b. Wyprowadzić wartość minimum. Jeśli liczby są równe, to wyprowadzić odpowiedni komunikat.

Opis słowny algorytmu

Po wczytaniu danych wejściowych a i b porównać wprowadzone liczby. Jeśli a < b, to min = a. Wyprowadzić wynik. Jeśli a >= b, to sprawdzić czy b < a. Jeśli tak, to min = b. Wyprowadzić wynik. W przeciwnym przypadku min = a = b. Wyprowadzić wynik.

Opis algorytmu za pomocą listy kroków

Krok 1. Wprowadź dwie liczby całkowite a i b. Przejdź do kroku 2.

Krok 2. Jeśli a < b, to podstaw min = a, wyprowadź wynik min = a.
Przejdź do kroku 5. W przeciwnym przypadku przejdź do kroku 3.

Krok 3. Sprawdź, czy b < a? Jeśli tak, to podstaw min = b, wyprowadź wynik min = b. Przejdź do kroku 5. W przeciwnym przypadku przejdź do kroku 4.

Krok 4. Podstaw min = a, wyprowadź wynik min = a = b. Przejdź do kroku 5.

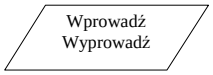
Krok 5. Zakończ program.

Postać graficzna algorytmu (sieć działań)

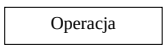
W sieciach działań (schematach blokowych) definiujących algorytmy są wykorzystywane następujące bloki.

- Blok początkowy (start programu)

- Blok wejścia / wyjścia (wprowadzanie lub wyprowadzanie danych)

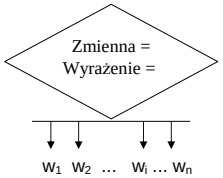


- Blok operacyjny (wykonywanie działań)

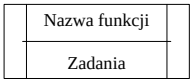


- Blok decyzyjny (warunkowy)

- Blok wyboru (wybór jednego z możliwych wariantów w zależności od wartości zmiennej lub wyrażenia)

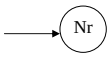


- Blok podprogramu (definiuje nazwę funkcji i realizowane przez nią zadania)

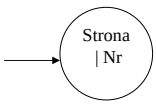


- Blok kolekcyjny (łączy dwie różne drogi algorytmu)

- Blok wejściowy lub wyjściowy łącznika na stronie (przejście pomiędzy fragmentami algorytmu w ramach tej samej strony)



- Blok wejściowy lub wyjściowy łącznika stronicowego (przejście pomiędzy fragmentami algorytmu znajdującymi się na różnych stronach)



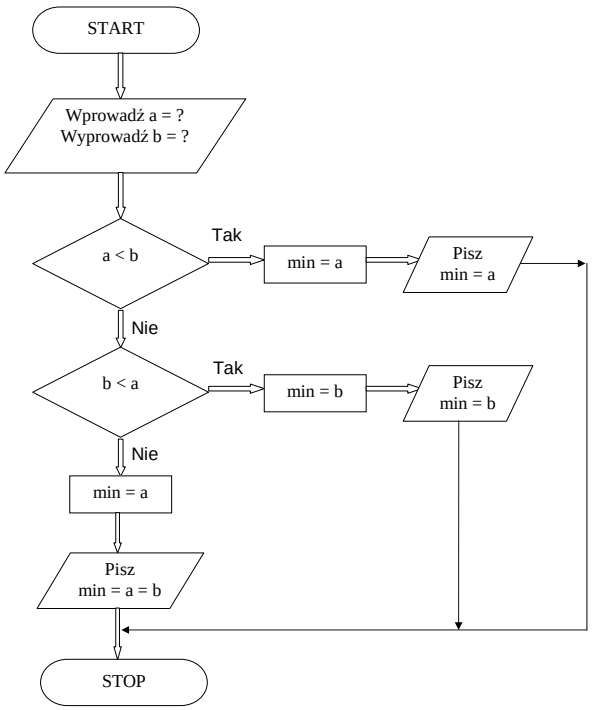
- Blok końcowy (koniec programu)

Opisując algorytmy za pomocą sieci działań należy pamiętać, aby:

- do każdego bloku dochodziła jedna strzałka,

- linie łączące bloki nie rozgałęziały się.

Schemat blokowy algorytmu wyznaczania min(a,b)



Implementacja algorytmu w postaci programu w języku Pascal

```
{ Obliczanie min(a,b) }
uses crt;   { dla clrscr }

{ zmienne globalne do przechowywania danych }
Var a, b, min: integer;

begin { początek programu }

 clrscr;  { wyczysc ekran }
 writeln('Wprowadz dane');
 readln(a);   readln(b);

 if (a<b) then
   begin  min:=a; writeln('Min = a = ', min); end
   else
    if (b<a) then
     begin min:=b;  writeln('Min = b = ', min); end
     else
     begin min:=a;  writeln('Min = a = b = ', min); end;
end.
```

Implementacja algorytmu w postaci programu w języku C

```
#include <stdio.h>  // dla printf i scanf
#include <conio.h>  // dla clrscr i getch
// ------------------      kompilator BC++ 3.1
int a, b, min;   //zmienne globalne
                 //do przechowywania danych
// Obliczanie min(a,b)

void main(void)  // główna (startowa) funkcja programu
{
 clrscr();       // wyczyść ekran
 printf("Wprowadz dane \n");
 scanf("%d", &a);   scanf("%d", &b);

 if (a<b) { min=a; printf("\nMin = a = %d \n", min); }
  else
   if (b<a) { min=b;  printf("\nMin = b = %d \n", min); }
   else { min=a;  printf("\nMin = a = b = %d \n", min); }
 getch();    // czekaj na enter
}
```

Problem 2. Znaleźć minimum spośród n wczytanych liczb a_0, a_1, ... , a_{n-1}. Wyprowadzić wartość minimum.

Opis słowny algorytmu

Po wczytaniu danych wejściowych a_i, dla i=0, ... , n-1, przyjąć min = a_0. Jeśli są jeszcze elementy do sprawdzenia (0<n-1), to sprawdzić czy a_i < min, dla i=1? Jeśli tak, to podstawić min = a_i. Powtórzyć sprawdzenie dla i=2, ... , n-1. Wyprowadzić wynik.

Opis algorytmu za pomocą listy kroków

Krok 1. Wczytaj dane a_0, ..., a_{n-1}.

Krok 2. Podstaw min = a_0 oraz i = 1.

Krok 3. Jeśli i > n-1 (nie ma więcej elementów), to przejdź do kroku 6.

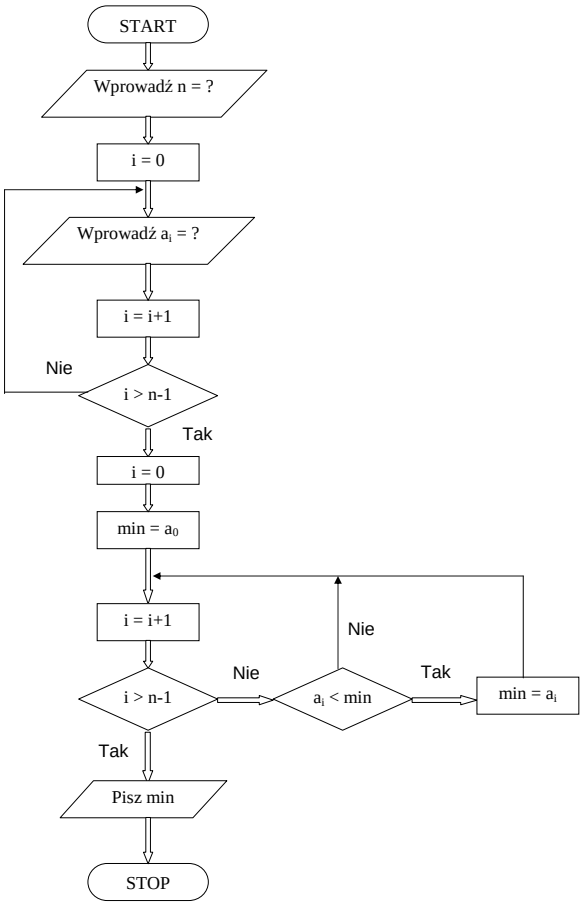
Krok 4. Jeśli a_i < min, to podstaw min = a_i.

Krok 5. Podstaw i = i + 1. Przejdź do kroku 3.

Krok 6. Wyprowadź wartość min.

Krok 7. Zakończ program.

Schemat blokowy algorytmu znajdowania min(a_0, ..., a_{n-1})



Implementacja algorytmu w postaci programu w języku C

```
#include <stdio.h>    // dla printf i scanf
#include <conio.h>    // dla clrscr i getch
#include <stdlib.h>   // dla random
// Obliczanie min(a[0],a[1], ... ,a[n-1])
// --------------------      kompilator BC++ 3.1
// Zmienne globalne
const  ROZ = 10;   // stała określająca
                   // maksymalny rozmiar tablicy
int a[ROZ]; //tablica - zawsze od 0, tj. a[0],...,a[ROZ-1]
int min; //wartosc minimum

void main(void)  //glowna (startowa) funkcja programu;
                 // void - bezparametrowa
{ // Zmienne lokalne funkcji main()
  int n;      //liczba n < ROZ wprowadzanych
              //(losowanych) elementów
  int i;   //zmienna pomocnicza indeksująca kroki pętli

 clrscr();    // wyczyść ekran
 randomize();   // inicjuj generator liczb losowych

 printf("Wprowadź liczbę elementów 0 < n <= %d \n", ROZ);
 scanf("%d", &n);

 /* do {
 printf("Wprowadź liczbę elementów 0 < n <= %d \n", ROZ);
 scanf("%d", &n);
    } while ( !(0<n && n<=ROZ) ); // czy poprawne n?
 */
 if (0<n && n<=ROZ) {  // czy n nie przekracza ROZ ?
   for (i=0; i<n; i++) {
     a[i] = random(100);  // losowanie danych
     printf("a[%d] = %d \n", i, a[i]); }

   i=0;          // wart. pocz. i
   min = a[0];  //  wart. pocz. minimum
   i=i+1;
   while ( !(i>n-1) ) {
     if (a[i] < min) min = a[i];
     i++;       // i = i+1
   }
   printf ("\n Wartosc minimum = %d \n", min);
   }  else
    { printf("\nWartosc n wykracza poza zakres !\n");
      printf("Uruchom ponownie program\n");
    }
getch();  //czekaj na enter
}
```

Przykładowe wyniki:

```
Wprowadz liczbe elementow 0 < n <= 10
5
a[0] = 40
a[1] = 6
a[2] = 93
a[3] = 12
a[4] = 34

Wartosc minimum = 6
```