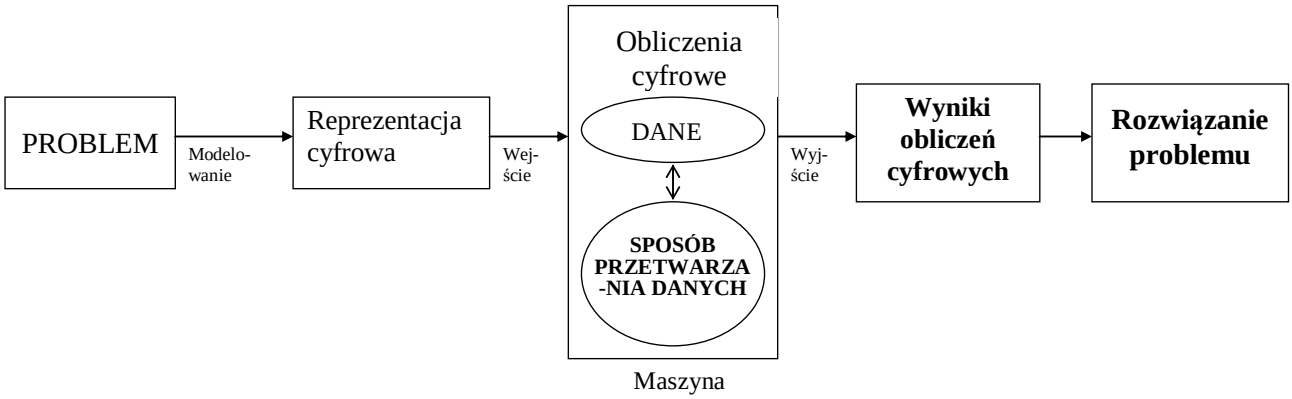
Język C – zajęcia nr 1

Cel stosowania metod i środków informatyki

Głównym celem stosowania metod i środków informatyki jest wspomaganie człowieka w procesie **rozwiązywania problemów**.

Przykładowe rodzaje problemów:

- obliczenia numeryczne - realizacja złożonych obliczeń,
- przetwarzanie danych - realizacja prostych obliczeń na dużej liczbie danych (np. tworzenie listy płac, rozliczanie rozmów telefonicznych),
- obsługa systemów baz danych - systemy magazynowe, systemy rezerwacji biletów kolejowych lub lotniczych,
- symulacja komputerowa - modelowanie fragmentów rzeczywistości (systemów biologicznych, ekonomicznych, fizycznych),
- zapewnienie komunikacji - poczta elektroniczna, transmisja plików, strony WWW w sieci Internet,
- sterowanie procesami - np. sterowanie linią produkcyjną,
- wspomaganie nauczania - słowniki, testy, programy do nauki języków obcych, programy encyklopedyczne,
- gry i zabawy komputerowe.



Dane stanowią reprezentację pewnych faktów, pojęć lub wielkości występujących w danym problemie. **Sposób tej reprezentacji musi umożliwiać dalsze przetwarzanie danych**.

Algorytm - określony **sekwencyjny sposób postępowania (przetwarzania danych)** prowadzący do rozwiązania postawionego problemu. Twórcą algorytmu jest zawsze człowiek. Algorytm może być realizowany prze człowieka lub przez komputer.

Wybrane definicje algorytmu:

Władysław Turski, Propedeutyka informatyki, PWN:

Przez algorytm będziemy rozumieć opis obiektów łącznie z opisem czynności, które należy wykonać z tymi obiektami, aby osiągnąć określony cel.

Maciej Sysło, Algorytmy, WSiP,
Algorytm jest przepisem opisującym krok po kroku rozwiązanie problemu lub osiągnięcie jakiegoś celu.

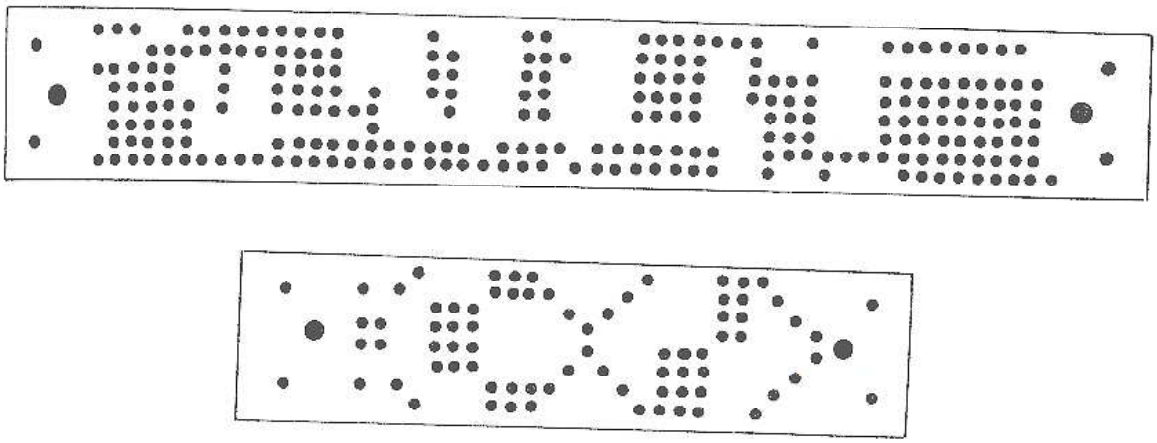
Algorytmika - gałąź wiedzy zajmująca się algorytmami.

Ciekawostki:

Słowo **algorytm** pochodzi od nazwiska perskiego matematyka Muhammada ibn Musa al-Chorezmi (przełom VIII i IX w. n.e.), którego nazwisko w języku łacińskim przyjęło postać: Algorismus.

Uważa się, że pierwszym algorytmem stworzonym przez człowieka był podany przez **Euklidesa** *algorytm wyznaczania największego wspólnego podzielnika* dwóch dodatnich liczb całkowitych (IV wiek p.n.e.).

Pierwszym urządzeniem realizującym algorytm było **krosno tkackie Jacquarda** (1801, Francja). Sposób pracy maszyny (tkane wzory) zapisywane były na kartach perforowanych.



Wzory kart perforowanych Jacquarda.

Podstawowe cechy algorytmu

Aby pewien sposób postępowania można było uznać za algorytm, musi on spełniać następujące warunki:

- **dyskretność** - algorytm składa się z szeregu realizowanych w odpowiedniej kolejności kroków. Każdy krok jest pewną operacją elementarną (np. dodaj dwie liczby) lub innym algorytmem (np. rozwiąż równanie kwadratowe),
- **jednoznaczność** - muszą istnieć ściśle określone zasady dotyczące kolejności wykonywania poszczególnych kroków algorytmu. Dla określonych danych wejściowych algorytm musi być zawsze realizowany w ten sam sposób,
- **skończoność (finistyczność)** - algorytm musi się zakończyć po zrealizowaniu skończonej liczby kroków. Wynikiem działania algorytmu może być rozwiązanie problemu lub informacja, że zadanie nie ma rozwiązania.

Dobry algorytm powinien spełniać również dwa dodatkowe postulaty:

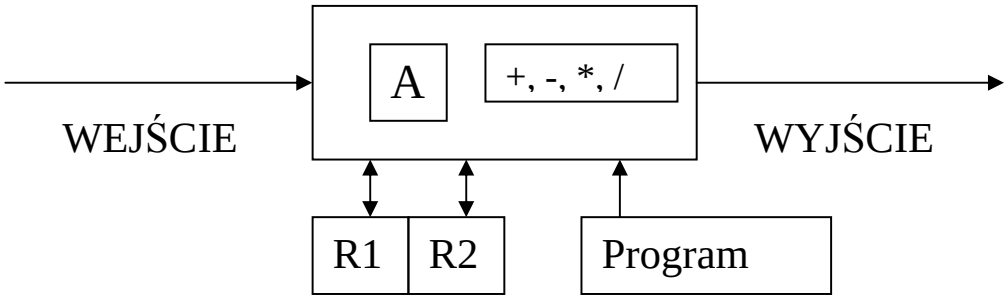
- **Efektywność** - algorytm zapewnia uzyskanie rozwiązania w rozsądnym czasie, adekwatnym do rozmiaru rozwiązywanego problemu.
- **Uniwersalność** - ten sam algorytm rozwiązuje możliwie szeroką klasę problemów umożliwiając ich parametryzację.

Problem szczegółowości algorytmów

Przy zapisie algorytmu może być przyjęty różny **poziom szczegółowości**. Na różnych poziomach szczegółowości przyjmuje się inny zestaw **operacji elementarnych** (a więc takich, które mogą być zrealizowane przez maszynę lub człowieka bez dodatkowych objaśnień).

Algorytm obliczania wartości $y = \frac{a+b}{c+d}$

Rozważmy niżej opisaną maszynę. Maszyna umie wykonywać następujące czynności:
- wczytywać liczbę podaną na wejściu do rejestru A,
- przesyłać zawartość rejestru A na wyjście,
- przesyłać zawartość A do rejestrów R1 i R2 i na odwrót,
- sprawdzać, czy A = 0
- wykonywać dwuargumentowe operacje +, -, *, / na zawartościach A i R1 lub A i R2, umieszczając wynik w A.



Algorytm zrozumiały dla maszyny:

Krok 1. Wczytaj wartość *c* do rejestru A

Krok 2. Prześlij wartość A do rejestru R1

Krok 3. Wczytaj wartość *d* do rejestru A

Krok 4. Wykonaj dodawanie A + R1, umieszczając wynik w A

Krok 5. Jeżeli A = 0 to zgłoś komunikat o błędzie dzielenia przez 0 i zakończ obliczenia, w przeciwnym wypadku przejdź do kroku 6.

Krok 6. Prześlij wartość A do rejestru R1

Krok 7. Wczytaj wartość *a* do rejestru A

Krok 8. Prześlij wartość A do rejestru R2

Krok 9. Wczytaj wartość *b* do rejestru A

Krok 10. Wykonaj dodawanie A + R2, umieszczając wynik w A

Krok 11. Wykonaj dzielenie A / R1, umieszczając wynik w A

Krok 12. Drukuj zawartość A i stop.

Bardziej „przyjazny” dla człowieka ale niezrozumiały bezpośrednio dla maszyny sposób zapisu algorytmu:

Krok 1. Wczytaj *a*, *b*, *c*, *d*

Krok 2. Oblicz $y = c + d$

Krok 3. **Jeżeli** $y = 0$ **to**
- drukuj komunikat o błędzie

w przeciwnym przypadku
- oblicz $y = (a + b) / y$
- drukuj y

Krok 4. Stop.

Problem efektywności (złożoności obliczeniowej) algorytmów

Często istnieje wiele różnych algorytmów służących rozwiązaniu tego samego zadania. W takiej sytuacji należy wybrać ten, który charakteryzuje się *najmniejszą złożonością obliczeniową*.

Złożoność obliczeniowa algorytmu – ilość zasobów komputerowych potrzebnych do realizacji algorytmu. Do podstawowych zasobów zalicza się czas działania (**złożoność czasowa**) i ilość zajmowanej pamięci (**złożoność pamięciowa**).

Złożoność obliczeniowa jest uzależniona od wielkości zadania (jest *funkcją* wielkości zadania).

Czas potrzebny na rozwiązanie pewnego problemu przez komputer można zredukować dwoma sposobami:
- można zastosować szybszy komputer,
- można zmodyfikować sposób rozwiązywania problemu (algorytm).

Sposoby zapisu algorytmów

- **zapis w języku naturalnym** - z uwagi na niejednoznaczność i małą precyzję opis taki jest rzadko stosowany,

  Jeżeli ciąg wejściowy nie jest pusty, to pobierz kolejny znak z wejścia, a w przeciwnym przypadku zdejmuj kolejne znaki umieszczone na stosie i przekazuj je na wyjście, a po opróżnieniu stosu zatrzymaj się. Jeżeli znak pobrany z wejścia jest operandem, to przekaż go bezpośrednio na wyjście ...

- **zapis przy pomocy notacji matematycznej** - bardzo precyzyjny, ale nie posiada pełnych możliwości w zakresie opisu struktury algorytmu,

  Algorytm znajdowania pierwiastków równania kwadratowego $ax^2 + bx + c = 0$

  $$\Delta = b^2 - 4ac$$

  $$x_1 = \frac{-b - \sqrt{\Delta}}{2a}$$

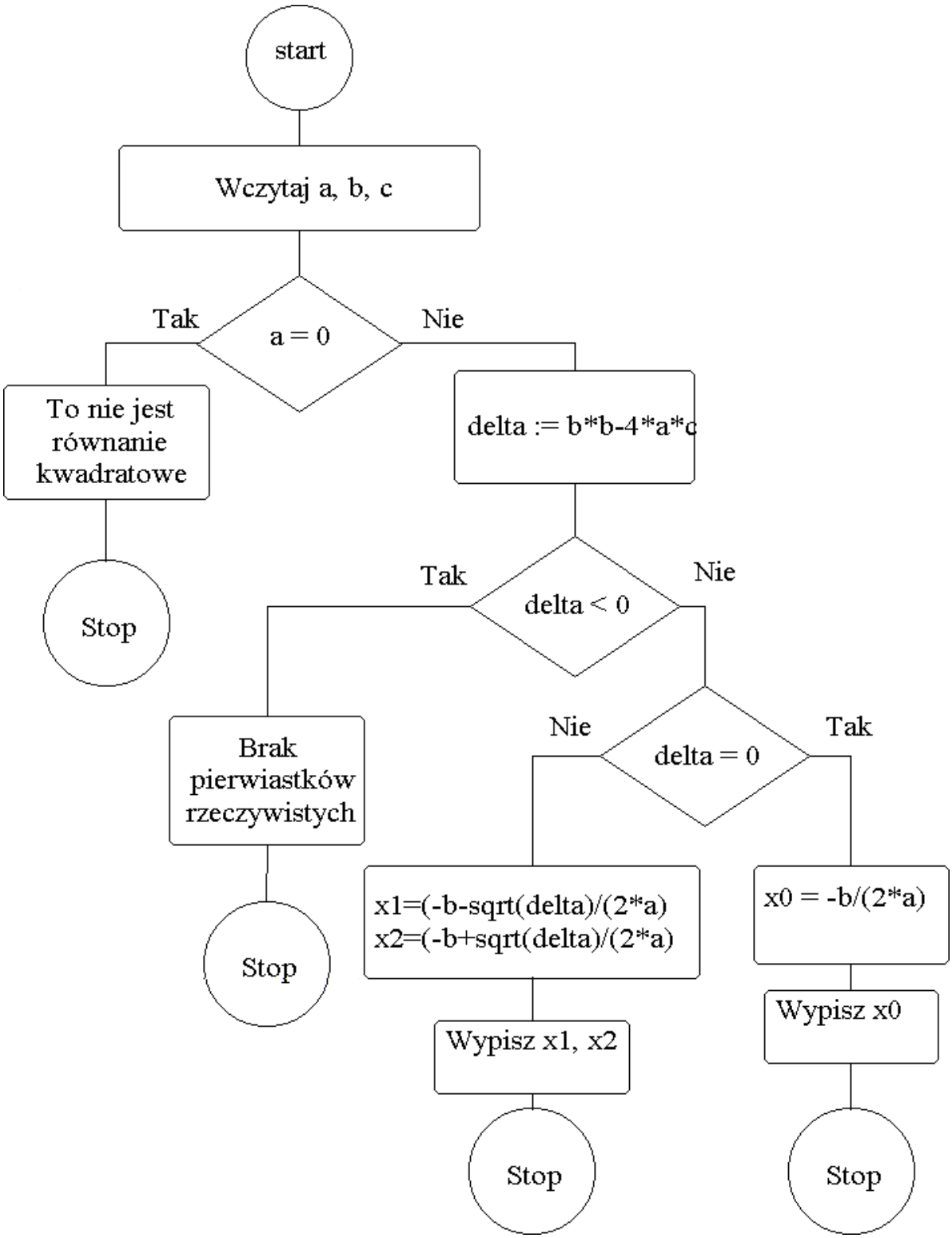
  $$x_2 = \frac{-b + \sqrt{\Delta}}{2a}$$

  W zależności od wartości, a raczej znaku, wyróżnika Δ rozwiązywane równanie ma:

  - dwa różne pierwiastki rzeczywiste ($\Delta>0$),
  - jeden podwójny pierwiastek rzeczywisty ($\Delta=0$),
  - dwa różne pierwiastki zespolone ($\Delta<0$).

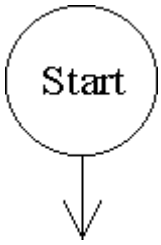
- **zapis przy pomocy schematów blokowych** - graficzny sposób opisu kolejnych kroków postępowania,

Algorytm rozwiązywania równania kwadratowego - schemat blokowy

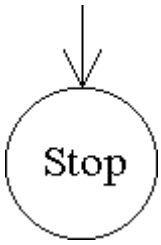


Bloki

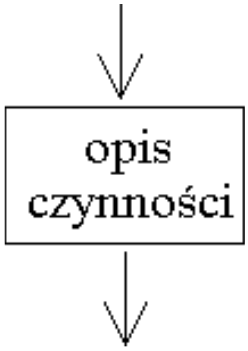
a) początek algorytmu



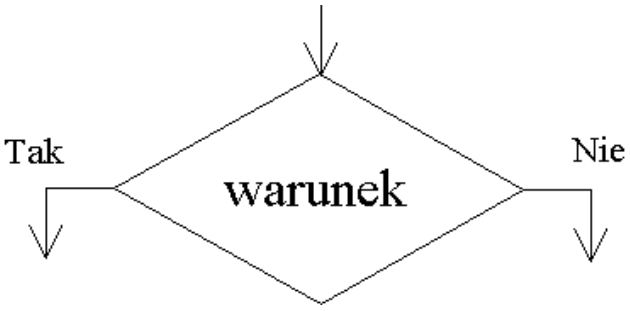
b) koniec algorytmu



c) opis czynności



d) blok warunkowy



- **zapis przy pomocy języków programowania** - umożliwia zapis algorytmu w sposób zrozumiały przez komputer (i w większości przypadków przez człowieka).

Program zawiera:
- opis **obiektów** na których wykonywane będą operacje,
- opis **czynności** wykonywanych w trakcie realizacji algorytmu.

Uwagi:
- Program ma postać *tekstu*, na który składa się opis **danych** oraz **instrukcje** opisujące algorytm,
- Każdy język programowania posiada inny zestaw instrukcji (różny od zestawu instrukcji procesora),
- Program zapisany przy pomocy języka programowania jest określany *programem źródłowym*.

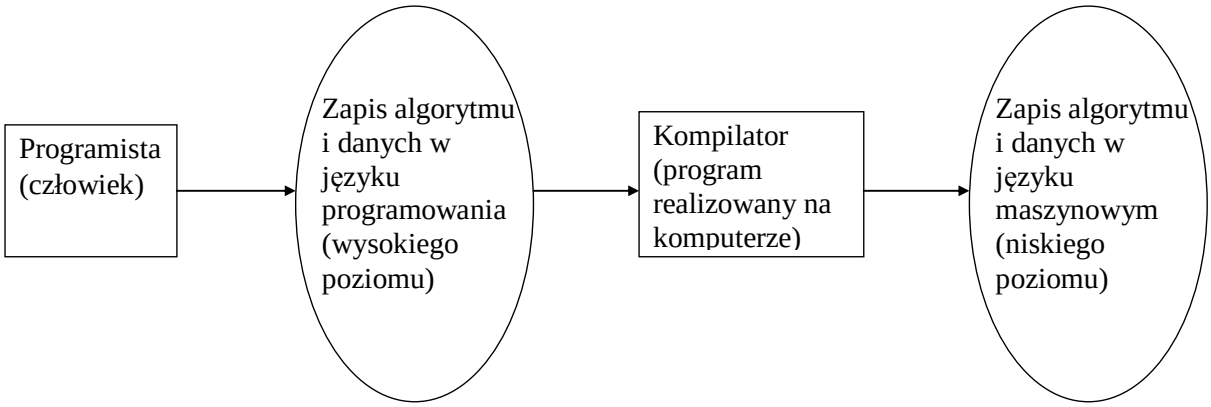
Przykład (język Pascal):

```
program dwieliczby;
var
   x,y:integer;
begin
   writeln('Podaj dwie liczby calkowite:');
   read(x,y);
   write('x=',x,'  y=',y)
end.
```

Program źródłowy musi zostać przetłumaczony do postaci binarnej (czyli na ciąg instrukcji zrozumiałych przez procesor).

Maszynowa reprezentacja programów

...
00110101
11001101
10011011
11000011
...



TRANSLACJA - proces **tłumaczenia** programu *z postaci źródłowej na postać wynikową* (binarną, maszynową).

Wykonywany jest program w postaci binarnej (wynikowej, maszynowej).

Translacja wykonywana jest przez **program** zwany **translatorem**. Dla każdego języka programowania istnieje odpowiedni program tłumaczący (translator danego języka).

Istnieją dwie podstawowe techniki translacji:

- **kompilacja** - wykonywana przez kompilator,
- **interpretacja** - wykonywana przez interpreter.

Kompilator - tłumaczy cały program źródłowy na język maszynowy. Program w języku maszynowym zapisywany jest na dysk lub umieszczany w pamięci operacyjnej. Po zakończeniu kompilacji możliwe jest uruchomienie przetłumaczonego programu.

Interpreter - tłumaczy kolejne instrukcje składające się na program. Po przetłumaczeniu instrukcji następuje jej realizacja. Proces tłumaczenia prowadzony jest na przemian z procesem realizacji.

Obecnie *większość* translatorów działa na zasadzie *kompilatora*.

Ewolucja metod programowania

Programowanie w językach wewnętrznych

Programista posługuje się binarnymi kodami instrukcji procesora.

...
00110101
11001101
10011011
11000011
...

Programowanie w językach symbolicznych (językach asemblerowych)

Binarne kody rozkazów zastąpione zostały instrukcjami w postaci słów (lub ich skrótów) kojarzących się z wykonywaną czynnością.

```
dispbyte PROC      FAR
;--------------------------------
; Procedura wyswietla szesnastkowo
; zawartosc rejestru DL
;--------------------------------
push      ax
push      dx
mov       ah,02h
mov       dh,dl
rol       dl,4
and       dl,0Fh
cmp       dl,9
jbe       lab1
add       dl,7
lab1:    add       dl,30h
int       21h
mov       dl,dh
and       dl,0Fh
cmp       dl,9
```

.....

Programowanie w językach wysokiego poziomu (I generacji)

Języki wysokiego poziomu pozwalały na ukrycie szczegółów budowy i zasad działania komputerów i dostarczały instrukcji odpowiadających ogólnym pojęciom pozwalających na opisanie algorytmów.

Przykładowe języki: Fortran (1954 r., John Backus)
FORTRAN = FORmula Translator

Opis algorytmu:

- Istniała możliwość stosowania instrukcji: *podstawienia* (przypisania), *warunkowej*, *pętli* (iteracji), *skoku*. Przy opisie algorytmy następuje rezygnacja ze stosowania rozkazów z listy rozkazów procesora i pojawia się możliwość stosowania pojęć o znacznie większym stopniu ogólności.
- Liczba możliwych do zastosowania instrukcji była stosunkowa niewielka, co wymuszało częste stosowanie instrukcji skoku.

Struktury danych:

- W programach pojawiły się instrukcje umożliwiając deklarowania zmiennych (deklaracja zmiennej – określenie nazwy i typu zmiennej), np. w języku FORTRAN:

```
INTEGER I, J, K
```

- Ułatwione zostało operowanie na tekstach
- Pojawiła się możliwość korzystania ze złożonych typów danych (tablice, pliki).

Języki proceduralne

Języki proceduralne pozwalają programiście na definiowanie podprogramów (procedur oraz funkcji). Przykładowy język programowania: Fortran II, Cobol, BASIC.

Opis algorytmu:

- zdefiniowanie nowego podprogramu pozwala na **rozszerzenie zbioru instrukcji** dostępnych w stosowanym języku programowania.
- mechanizm parametrów zwiększył uniwersalność definiowanych instrukcji.

Struktury danych:

- pojawiła się możliwość definiowania *danych lokalnych* (dostępnych tylko w podprogramie) oraz *danych globalnych* (dostępnych w całym programie),
- dostępność struktur danych uzależniona była od przeznaczenia języka programowania (w języku FORTRAN nacisk położono na struktury przydatne przy rozwiązywaniu problemów numerycznych /liczby zespolone, tablice/; język Cobol posiadał szereg narzędzi specjalizowanych w zakresie przetwarzania plików).

Programowanie strukturalne

Programowanie zostało mocno powiązane z procesem projektowania. **Sposób realizacji poszczególnych funkcji programu opisany jest przez składające się na program podprogramy.**

Przykładowe języki programowania: Pascal, C.

Opis algorytmu:

- dopasowanie struktury programu do struktury algorytmu,
- zwiększenie liczby instrukcji sterujących (np.: różne postaci pętli, instrukcji warunkowych, wyboru, podstawienia) – dzięki czemu można było wyeliminować instrukcję skoku bezwarunkowego

Struktury danych:

- zwiększenie liczby dostępnych typów danych (tablice, rekordy, zbiory, pliki),
- możliwość stosowania dynamicznych struktur danych (stos, kolejka, listy, drzewa),
- możliwość definiowania własnych struktur danych.

W programie wyraźnie uwidocznił się podział na dwie zasadnicze części:

- opis danych (służących do przechowywania informacji o stanie przetwarzanych obiektów),
- opis algorytmów (sposób przetwarzania obiektów).

Idea programowania strukturalnego wprowadziła pewien porządek i systematykę w proces powstawania programów komputerowych.

Główny postulat programowania strukturalnego można sformułować w postaci następującego zalecenia dla programisty:

Tworzony program musi być czytelny i zrozumiały nie tylko dla jego autora. Struktura programu powinna odpowiadać strukturze rozwiązywanego problemu.

Istotna metoda programowania strukturalnego:

Jedną z metod programowania strukturalnego jest programowanie **metodą kolejnych ulepszeń**. Zgodnie z tą metodą programowanie i konstruowanie algorytmu obejmuje szereg etapów.

W pierwszym etapie skupiamy się na sprawach ogólnych. Pierwsza wersja rozwiązania nie zawiera szczegółów. W miarę postępu prac projektowych główny problem rozpada się na **podproblemy**. Rozważanych jest coraz więcej szczegółów.

Podczas programowania cały proces obliczeń dzielimy na akcje, co odpowiada dzieleniu programu na instrukcje. Podział taki musi gwarantować, że z rozwiązania wszystkich cząstkowych podproblemów wynika rozwiązanie całości problemu.

W programowaniu strukturalnym występują dwie różne odmiany metody kolejnych ulepszeń, które charakteryzuje przeciwstawny kierunek podążania od sformułowania problemu do utworzenia programu, który służy rozwiązywaniu postawionego zadania:

metoda analityczna nazywana również **zstępującą** lub metodą Top-Down; programista tworzy kolejne wersje programu w każdej z nich uwzględniające coraz więcej szczegółów,

metoda syntetyczna nazywana także **wstępującą** lub Bottom-Up; programista buduje proste podprogramy złożone z niewielu instrukcji procesora, a dalej łączy je w większe moduły, aż do utworzenia pełnego programu.

Obydwie metody prowadzą do powstania programów o wyraźnej strukturze odzwierciedlającej algorytm rozwiązania.

Programowanie obiektowe

Zasadniczą cechą programowania obiektowego jest łączne rozpatrywanie zagadnień dotyczących algorytmów i struktur danych. Wyrazem tego jest zmiana w sposobie reprezentacji rzeczywistych obiektów przejawiająca się w możliwości definiowania klas. Definicja klasy obejmuje opis stanu obiektu (dane charakteryzujące obiekt) oraz sposób zachowania (lub przetwarzania) obiektu.

Języki programowania pozwalające na programowanie obiektowe:

- Simula (1967, pierwszy język obiektowy),
- Ada,
- Object Pascal,
- Modula-3,
- C++,
- Smalltalk,
- Java.

Ogólna charakterystyka języków C i C++

C

- powstał w 1972 roku,
- jest językiem wysokiego poziomu, ale posiada również mechanizmy bezpośredniego operowania na elementach sprzętowych komputera (np. rejestrach, pamięci operacyjnej),
- jest niezależny sprzętowo,
- program składa się z szeregu podprogramów,
- tekst źródłowy jest krótki (zwartość),
- jest bardzo popularny wśród programistów tworzących programy w systemie UNIX (kompilator języka C jest częścią składową systemu operacyjnego UNIX).

C++

- unowocześniona wersja języka C,
- powstał w roku 1985,
- umożliwia stosowanie **techniki programowania obiektowego.**

Język C

Plik źródłowy. Tekst programu w języku C może być zapisany w jednym lub większej liczbie plików. Kompilacja każdego pliku źródłowego (modułu źródłowego) przebiega oddzielnie. Wszystkie uzyskane podczas kompilacji fragmenty wynikowe (moduły półskompilowane) są łączone ze sobą oraz innymi podobnymi fragmentami pochodzącymi z bibliotek, co prowadzi do utworzenia modułu wykonywalnego.

Niezależna kompilacja modułów źródłowych:
kotek1.c → kotek1.obj
kotek2.c → kotek2.obj
kotek3.c → kotek3.obj
Konsolidacja modułów półskompilowanych i bibliotek:
kotek1.obj + kotek2.obj + kotek3.obj + bib_1.lib + ... + bib_n.lib → kotek.exe

Pierwsze programy:

Zadanie 1. Program wypisujący określony tekst.

a) Uruchom Dev C++.
Rozpocznij pracę z nowym projektem.
Wybierz typ projektu: *Console Application*, opcje: *Projekt C*, nazwę: C01-1.
Zapisz projekt w dostępnej lokalizacji (zalecane: *Twój pendrive*).

Zaproponowany zostanie początkowy, „stały” fragment programu w języku C, który należy potem uzupełnić:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{

  system("PAUSE");
  return 0;
}
```

Uwagi:

`#include <stdio.h> -` dyrektywa preprocesora (informacja dla kompilatora), realizująca dołączenie standardowej biblioteki wejścia wyjścia umieszczonej w pliku `stdio.h`, która zawiera m.in. definicję funkcji `printf`

`#include <stdlib.h> -` j.w. - dotyczy pliku `stdlib.h` zawierającego zestaw standardowych przydatnych funkcji

`main` – nazwa funkcji, funkcja o tej nazwie musi wystąpić w programie; program zaczyna działanie od początku funkcji main; funkcja ta powinna zwracać wartość całkowitą (int) (nie jest to wymóg języka C, a tylko zalecenie Dev C++) i może posiadać argumenty (lista argumentów w nawiasach po nazwie funkcji)

`return 0; –` instrukcja określająca wartość zwracaną przez funkcję `main` (w naszym przypadku wartość ta jest nieistotna)

`system` – funkcja wydająca komendę dla systemu operacyjnego, tekst komendy (przekazywanej tak jak w wierszu poleceń systemu operacyjnego) stanowi argument funkcji SYSTEM (w nawiasie), w przypadku komendy PAUSE

następuje wstrzymanie wykonania programu do chwili naciśnięcia dowolnego klawisza

Skompiluj i uruchom program, zapisując pliki w tej samej lokalizacji co projekt. Program ten oczywiście nic nie robi.

b) Zmodyfikuj program do postaci jak niżej, skompiluj i uruchom; jaki jest efekt wykonania programu?

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
  printf("Nasz pierwszy program\n");
  system("PAUSE");
  return 0;
}
```

Uwagi:

`printf` – funkcja wypisująca określone wartości na urządzeniu wyjścia

`"tekst"` – ciąg znaków (stała tekstowa, tekst) – ujęty w cudzysłowy, tu stanowi argument funkcji PRINTF

`\n` – sekwencja stanowiąca znak sterujący powodujący przejście do nowej linii (w kodzie ASCII dwa znaki: CR i LF)

c) zmodyfikuj program tak, aby wypisywał tekst w postaci:

```
Nasz
pierwszy
program
```

Zadanie 2.

a) Program C01-2, wypisujący wartość całkowitą. Wprowadź, skompiluj i uruchom następujący program:

```
#include <stdio.h>
int i;
int main()
{
   i=1;
   printf("Zmienna i ma wartosc: %d \n",i);
   system("PAUSE");
   return 0;
}
```

Uwagi:

`int i –` deklaracja zmiennej `i` typu całkowitego,

`i=1 –`przypisanie, zmiennej `i` przypisana zostaje wartość 1

`%d –` specyfikator elementu listy wyjściowej – dziesiętna liczba całkowita

b) Wprowadź, skompiluj i uruchom program C01-3 przypisujący dwóm zmiennym całkowitym i, j wartości 1234 i 5678 i wypisujący wynik:

```
Zmienne i, j mają odpowiednio wartości:
1234
5678
```

c) Napisz program C01-4 przypisujący dwóm zmiennym całkowitym `i`, `j` określone (dowolne) wartości i wypisujący ich sumę i różnicę w układzie:

```
i = 1234, j = 5678
suma = ……
roznica = ……
```

Uwaga: wykorzystaj przypisanie postaci:

```
suma = i+j;
```

w wyniku którego najpierw obliczana jest wartość wyrażenia po prawej stronie znaku `=`, a następnie wartość ta zostaje przypisana zmiennej stojącej po lewej stronie `=`, pamiętaj aby zadeklarować odpowiednie zmienne!