Podstawy programowania.
Wykład 4
Konstrukcje sterujące

Kod źródłowy i wykonanie programu

- Kod źródłowy w języku programowania zawiera przepis wykonania programu
- W maszynie von Neumana procesor wykonuje kolejne rozkazy w kodzie binarnym
  - przy kompilacji instrukcja po instrukcji oznacza to kolejne instrukcje kodu źródłowego

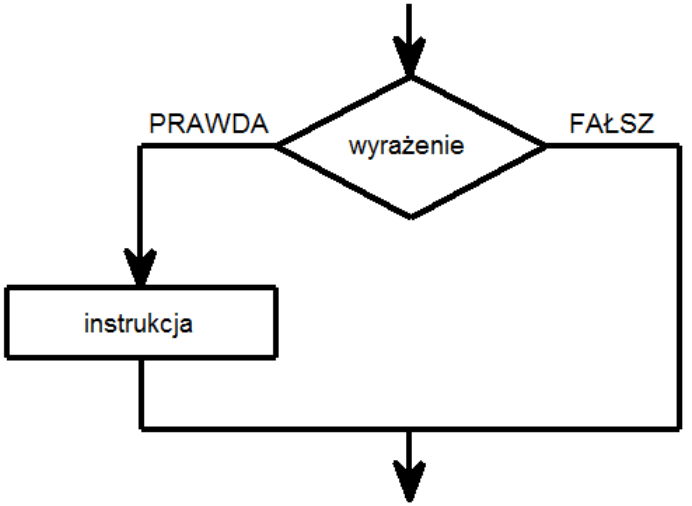
  chyba że ...
- w kodzie pojawiają się instrukcje sterujące
  - instrukcja sterująca umożliwia wybór kolejnego rozkazu do wykonania czyli
    - ewentualne ominięcie pewnych rozkazów
      - skok do przodu w kodzie binarnym
    - cofnięcie się do wcześniejszych rozkazów
      - skok do tyłu w kodzie binarnym

Kod źródłowy i wykonanie programu

- Dzięki instrukcjom sterującym
  - pewne fragmenty kodu mogą nie być wykonane
    - kompilator może pominąć kod nigdy nie wykonywany
  - pewne fragmenty kodu mogą być wykonane wiele razy
  - w konsekwencji liczba rozkazów wykonanych w trakcie działania programu nie ma związku z liczbą rozkazów w kodzie binarnym
    - zależy częściej od (rozmiaru) danych wejściowych
- W kodzie binarnym instrukcje sterujące mają postać skoków (do przodu lub do tyłu)
  - wykonanie skoku może zależeć od spełnienia pewnych warunków przez dane w programie
- W kodzie źródłowym występują instrukcje sterujące zdefiniowane dla danego języka programowania
  - najważniejsze instrukcje sterujące w C/C++ to
    - instrukcje wyboru (*if, switch*)
    - pętle (*for, while, do*)

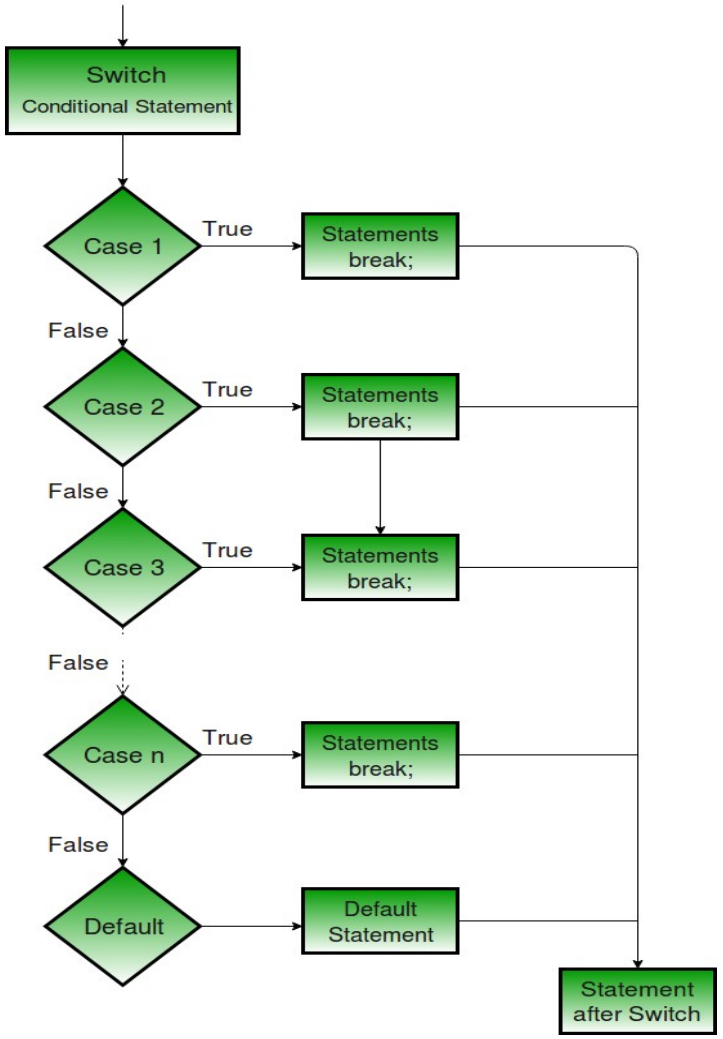
Instrukcje sterujące

- if(wyrażenie) { ...(instrukcja lub blok instrukcji)... }
  - instrukcja jest wykonywana jeśli wyrażenie jest prawdziwe (różne od zera)
- if(wyrażenie) { .... } else { ..... }
- if(wyrażenie_1) { .... }
  else if (wyrażenie_2) { .... }
  else if (wyrażenie_3) { .... }
  else { .... }
  - jeśli któreś z wyrażeń jest prawdziwe, wartości kolejnych nie są obliczane



Instrukcje sterujące

- Switch
  - przykład prosty:



  - przykład bardziej skomplikowany:

```
printf("\nWprowadź cyfrę od 1 do 5: \n");
scanf(" %c",&c);
switch (c) {
   case '0':
     printf("Wprowadzono: 0\n"); break;
   case '1':
     printf("Wprowadzono: 1\n");
   case '2':
     printf("Wprowadzono: 1 lub 2\n"); break;
   case '3':
     printf("Wprowadzono: 3\n"); break;
   case '4':
   case '5':
     printf("Wprowadzono: 4 lub 5\n"); break;
   default:
     printf("Wprowadzono: znak spoza zakresu 0-5\n"); break;
}
```

Rozwiązanie równania kwadratowego

- Definicja problemu:
  - dane wejściowe: liczby a, b, c
  - dane wyjściowe:
    - liczby x_1 i x_2 będące pierwiastkami równania kwadratowego $ax^2 + bx + c = 0$ ($a*(x-x_1)*(x-x_2)=0$)
  - kontrakt procedury obejmuje także:
    - warunki wstępne (*precondition*)
      - typy i dopuszczalne wartości a, b, c (kiedy procedura ma sprawdzać dane wejściowe?)
    - warunki końcowe (*postcondition*)
      - dodatkowe informacje o x_1, x_2 (np. czy algorytm ma uwzględniać tylko pierwiastki rzeczywiste)
      - sytuacje zgłaszania komunikatów o błędzie

Rozwiązanie równania kwadratowego

- Algorytm w pseudokodzie:

- wczytaj a, b, c
- oblicz delta = b*b – 4*a*c
- jeżeli(delta<0) wypisz(„złe dane”) i zakończ program
- jeżeli(delta>0) (tzn. w każdym innym przypadku)
- oblicz x1 = (-b - sqrt(delta))/(2*a)
- oblicz x2 = (-b + sqrt(delta))/(2*a)
- wypisz wartości x1,x2
- zakończ program

(co jeśli a=0 ?)

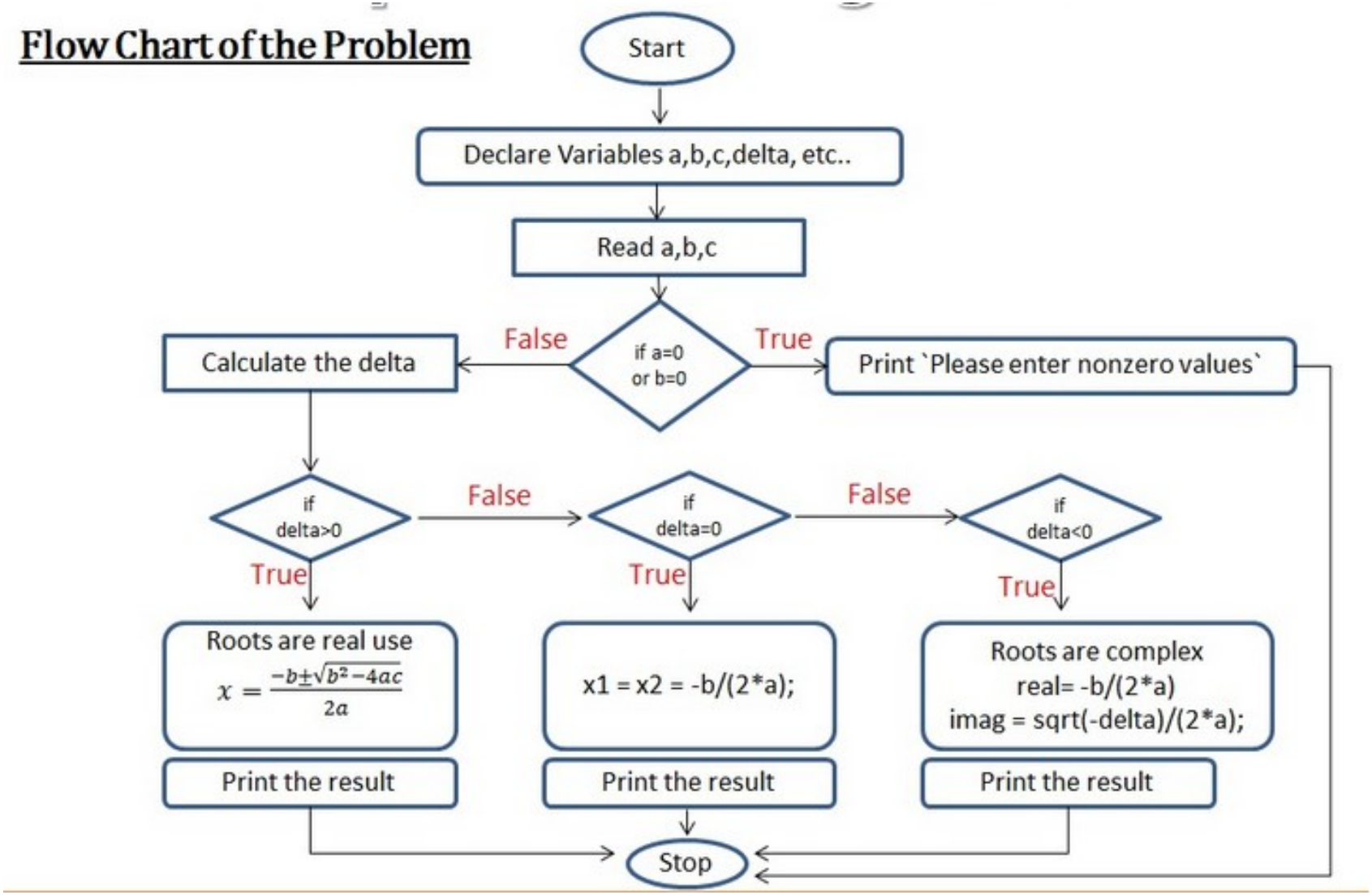
Rozwiązanie równania kwadratowego

- Możliwa realizacja algorytmu (zbliżona do assemblera):

wczytaj(a,b,c)
delta = b*b – 4*a*c
PORÓWNAJ(delta,0)
JEŚLI MNIEJSZE SKOCZ DO L1
x1 = (-b - sqrt(delta))/(2*a)
x2 = (-b + sqrt(delta))/(2*a)
wypisz(x1,x2)
SKOCZ DO L2
L1: *wypisz*(„złe dane”)
L2: STOP

W przypadku uwzględnienia sprawdzania wartości a i b zapis jest jeszcze mniej czytelny

Rozwiązanie równania kwadratowego



Rozwiązanie równania kwadratowego

```
#include <math.h> // USES (podobnie stdlib.h i stdio.h)
int main(void) // rozwiązanie równania kwadratowego ax^2 + bx + c = 0
{
   int a, b, c;    // rozważania o kontrakcie..., input - uodpornianie na błędy wczytywania danych
   printf("Podaj parametr a: "); scanf("%d", &a); // adres! - podobnie wczytanie b i c
   if(a==0 && b==0){ // alternatywa: if( a==0 || b==0 ) - zależnie od kontraktu
      printf("Błędne dane: a i b równe 0. Przerwanie programu.\n");  exit(-1);
   }
   else{
      if(a==0) { // równanie liniowe, co mówi kontrakt?
      }else{
         double delta = b*b – 4*a*c; // zakres widoczności nazwy - powiązanie z czasem życia
         if(delta<0){ printf("Dwa pierwiastki zespolone - nie umiem obliczyć\n");  }
         else if (delta == 0){ printf("Pierwiastek rzeczywisty: x = %lf\n", -b/(2.0*a) ); }
         else {
            double temp = sqrt(delta);
            printf("Dwa pierwiastki rzeczywiste: x1 = %lf, x2 = %lf\n",  (-b-temp)/(2.0*a), (-b+temp)/(2.0*a) );
         }
      } // znaczenie wcięć i nawiasów klamrowych dla zwiększenia czytelności kodu
   }
   return(0);
}
```

Problem skończonej precyzji

- *Problemy dla reprezentacji z ograniczoną dokładnością*
  - ile bitów potrzeba żeby dokładnie zapisać liczbę 0.1?
    - tylko niektóre spośród liczb rzeczywistych dają się zapisać w reprezentacji binarnej o skończonej dokładności
    - równość matematyczna nie jest równoznaczna z równością przy reprezentacji ze skończoną dokładnością
  - problemy z arytmetyką
    - $x * (1/x) \neq 1$ – dla 135 spośród pierwszego tysiąca liczb naturalnych przy zapisie z pojedynczą precyzją
    - $a + b = a$ (dla b>0 ale znacznie mniejszego od a)
      - co z 1/((a+b)-a) ?
    - (a+b)-a != (a-a)+b - brak łączności i przemienności operacji
    - przydatność poprawy dokładności w trakcie dokonywania obliczeń
    - rozwiązanie niektórych problemów za pomocą zmiany algorytmu

Problem skończonej precyzji

- Algorytm rozwiązania równania kwadratowego:
  - x1 [x2] = (- b + [-] sqrt(delta)) / (2*a)
    - przykład (4 cyfry znaczące): a = 1, b = -320, c = 16
      - b^2 = 102400, 4*a*c = 64, delta = 102400 – 64 = 102336
      - delta zaokrąglana do 102300, pierwiastek do 319.8
      - x1 = (320+319.8)/2 = 319.95 ≈ 319.9
        - dokładnie x1 = 319,949992185 – błąd względny ≈ 0,01%
      - x2 = (320-319.8)/2 = 0.1 – utrata cyfr znaczących
        - dokładnie x2 = 0,050007815 – błąd względny ≈ 100%
- Zmiana algorytmu
  - x1 = - b +/- sqrt(delta)) / (2*a) ; x2 = c / (a*x1)
    - przykład (4 cyfry znaczące):
      - x1 = (320+319.8)/2 = 319.95 ≈ 319.9 – błąd względny ≈ 0,01%
      - x2 = 16/319.9 = 0.05 – błąd względny ≈ 0,015%

Problem skończonej precyzji

- *Przykładowe wskazówki postępowania z liczbami zmiennoprzecinkowymi, dla uniknięcia błędów wynikających ze zbyt małej precyzji:*
  - sumować najpierw małe, potem duże liczby
  - unikać cząstkowych wyników bardzo dużych lub bardzo małych
  - unikać odejmowania liczb bardzo bliskich sobie
  - nie porównywać liczb tylko ich różnicę z małą wartością

```
double wartosc, wzorzec; // tak samo dla float
// obliczanie wartości
if( wartosc != wzorzec ) {....}
#define TOLERANCJA 1e-9 // tolerancja może być inna dla double i float
// znajomość przewidywanych wartości pozwala wybrać dokładność:
if( fabs( wartosc – wzorzec ) <= TOLERANCJA ) {....} // bezwzględną
if( fabs( (wartosc–wzorzec )/wzorzec ) <= TOLERANCJA ) {....} // względną
```