Anna Jasielec

Podstawy programowania, grupa nr 4

Sprawozdanie z laboratorium nr 5

Cel laboratorium: Opanowanie podstaw stosowania konstrukcji sterujących if i switch w C.

Przebieg zajęć: Utworzenie katalogu lab_5 oraz podkatalogu switch. Kompilacja w terminalu i sprawdzenie działania pliku simple_switch.c.

- Zamiana konstrukcji *switch* na postać standardową, modyfikacja komunikatow, uzycie instrukcji *break*. Dodanie opcji, która kończy program po wciśnięciu klawisza Q.

```
printf("\nWprowadź cyfrę od 1 do 5: ");
scanf(" %c",&c);

switch (c) {

case '1':
  printf("Wprowadzono: 1\n");
  break;

case '2':
  printf("Wprowadzono: 2\n");
  break;

case '3':
  printf("Wprowadzono: 3\n");
  break;

case '4':
  printf("Wprowadzono: 4\n");
  break;

case '5':
  printf("Wprowadzono: 5\n");
  break;

case 'Q': return(-1);
case 'q': return(-1);


default:
  printf("Wprowadzono: znak spoza zakresu 1-5\n");
  break;
}
```

- Uzycie konstrukcji if … else if … else do napisania programu o takim samym działaniu jak powyższy.

```
for(;;) {

   printf("\nWprowadź cyfrę od 1 do 5: ");
   scanf("%c",&c);


  if (c=='1') {
     printf("Wprowadzono: 1\n");
     }

  else if (c=='2') {
     printf("Wprowadzono: 2\n");
  }

  else if (c=='3') {
     printf("Wprowadzono: 3\n");
  }

  else if (c=='4') {
     printf("Wprowadzono: 4\n");
  }

  else if (c=='5') {
     printf("Wprowadzono: 5\n");
   }

   else if (c=='Q' || c=='q') {
       return(-1);
   }

   else {
     printf("Wprowadzono: znak spoza zakresu 1-5\n");
   }
  }
```

- Utworzenie podkatalogu *rownanie_kwadratowe,* analiza pliku *kompiluj.sh*, modyfikowanie pliku *rownanie_kwadratowe.c.*
- Sprawdzanie, czy obliczone pierwiastki są rzeczywiście pierwiastkami, używając funkcji *fabs*.

```
//sprawdzenie czy pierwiastki sq pierwiastkami(czy sq rowne 0)
      if(fabs((a*x1*x1)+(b*x1)+c) <= TOLERANCJA) {
        printf("x1 jest pierwiastkiem rownania\n");
    }
    else {printf("x1 nie jest pierwiastkiem");}

 if(fabs((a*x2*x2)+(b*x2)+c) <= TOLERANCJA) {
        printf("x2 jest pierwiastkiem rownania");
    }
    else {printf("x2 nie jest pierwiastkiem\n");}
```

```
Podaj parametr a: 2
Podaj parametr b: 12
Podaj parametr c: 7
Dwa pierwiastki rzeczywiste: x1 =   -5.345207879911715, x2 =   -0.654792120088285
x1 jest pierwiastkiem rownania
x2 jest pierwiastkiem rownania
```

```
if(fabs(x1-x2)<TOLERANCJA) {
  printf("\nRoznica wartosci jest rowna 0 (zbyt bliska 0)");}
```

- Rozważenie przypadków kiedy a jest bardzo małą liczbą, a b i c są standardowe, dla typów zmiennych float oraz double. **$a = 10^{-k}$,** b = 2, c = 1:

k =	**5**	**6**	**7**
wynik dla **float**	x1 = -199999.500000000000000 x2 = -0.500679016113281	x1 = -1999999.500000000000000 x2 = -0.536441802978516	x1 = -20000000.000000000000000 x2 = -0.596046447753906
wynik dla **double**	x1 = -199999.499998749990482 x2 = -0.500001250003379	x1 = -1999999.499999875202775 x2 = -0.500000124969979	x1 = -19999999.499999988824129 x2 = -0.500000012504387
k =	**8**	**9**	**10**
wynik dla **float**	x1 = -200000000.000000000000000 x2 = 0.000000000000000	x1 = -2000000000.000000000000000 x2 = 0.000000000000000	x1 = -2000000000.000000000000000 x2 = 0.000000000000000
wynik dla **double**	x1 = -199999999.500000000000000 x2 = -0.500000008063495	x1 = -1999999999.499999761581421 x2 = -0.500000041370185	x1 = -19999999999.500000000000000 x2 = -0.500000041370185
k =	**11**	**12**	**13**
wynik dla **double**	x1 = -199999999999.500000000000000 x2 = -0.500000041370185	x1 = -1999999999999.500000000000000 x2 = -0.500044450291171	x1 = -19999999999999.496093750000000 x2 = -0.500710584105946

Dla typu float wyniki zaczynają się zaburzać przy k = 5 (x2 powinien być coraz bliższy 0.5), a dla typu double przy k = 9.

- Dobranie parametru TOLERANCJA i wprowadzenie warunku if(fabs(a)<TOLERANCJA).

```
#define TOLERANCJA 1e-9 //tolerancja dla double
#define TOLERANCJA 1e-5 //tolerancja dla float
```

```
if( fabs(a)<TOLERANCJA) {
  printf("Przekroczono wartosc tolerancji\n");
  exit(-1);
}
```

- Modyfikacja programu tak, aby rozwiązywał przypadki równania liniowego i o pierwiastkach zespolonych.

```
 if(fabs(a)<TOLERANCJA) { // równanie liniowe
   printf("Równanie liniowe:\n");
SCALAR x;
x = -c/b;
printf("Jeden pierwiastek x = %20.15lf\n",x);
 }
```

```
Podaj parametr a: 0
Podaj parametr b: 7
Podaj parametr c: 6
Równanie liniowe:
Jeden pierwiastek x =   -0.857142857142857
```

```
  if(delta<0){

printf("Dwa pierwiastki zespolone:\n");

SCALAR complex x1,x2;

double complex negSqrt = csqrt(delta);
double pReal = creal(negSqrt);
double pImag = cimag(negSqrt);

SCALAR complex zespolona = pReal + pImag*I;


x1 = (-b-negSqrt)/2*a;
x2 = (-b+negSqrt)/2*a;

printf("x1 = %.2f + %.2fi\tx2 = %.2f + %.2fi\n"
, creal(x1), cimag(x1), creal(x2), cimag(x2));
  }
```

```
Podaj parametr a: 42
Podaj parametr b: 4
Podaj parametr c: 123
Dwa pierwiastki zespolone:
x1 = -84.00 + -3017.58i
x2 = -84.00 + 3017.58i
```

- uwzględnienie przypadku: a i c standardowe, b duże: **b = 10** ^{k}, a = 1, c = 1

k =	**2**	**3**	**4**	**5**
wynik dla **float**	x1 = -99.989997863769531 x2 = -0.010002136230469	x1 = -999.999023437500000 x2 = -0.001007080078125	x1 = -10000.000000000000000 x2 = 0.000000000000000	x1 = -100000.000000000000000 x2 = 0.000000000000000
wynik dla **double**	x1 = -99.989998999799951 x2 = -0.010001000200049	x1 = -999.998999999000034 x2 = -0.001000001000023	x1 = -9999.999899999998888 x2 = -0.000100000001112	x1 = -99999.999989999999661 x2 = -0.000010000003385
k =	**6**	**7**	**8**	**9**
wynik dla **double**	x1 = -999999.999998999992386 x2 = -0.000001000007614	x1 = -9999999.999999899417162 x2 = -0.000000099651515	x1 = -100000000.000000000000 x2 = -0.000000007450581	x1 = -1000000000.000000000000 x2 = 0.000000000000000

Jeden z pierwiastków zbliża się do -1/b, a drugi do -b. W typie float wyniki zaburzają się przy k = 4, w double k = 9. Dzieje się tak, ponieważ liczby przy kompilacji są zaokrąglane, tracimy znaczące cyfry, mamy duży błąd względny.

- Modyfikacja algorytmu, aby zmniejszyć błąd względny:

```
x1 = - b + sqrt( delta ) / (2*a);
x2 = c / (a*x1);
```

- Modyfikacja programu tak, aby wszystkie przypadki były uwzględnione w jednej konstrukcji if ... elsif ... elsif ... else. Taki kod jest nieczytelny.

```
if( fabs(a)<SMALL_NUMBER && fabs(b)<SMALL_NUMBER ){ // poprawnie
printf("Bledne dane: a i b rowne 0 (zbyt bliskie 0). Przerwanie programu.\n");
exit(-1);}

else if(fabs(a)<TOLERANCJA) {
printf("Rownanie liniowe:\n");
SCALAR x;
x = -c/b;
printf("Jeden pierwiastek x = %20.15lf\n",x);}

else if(delta<0){
printf("Dwa pierwiastki zespolone:\n");
SCALAR complex x1,x2;
double complex negSqrt = csqrt(delta);
double pReal = creal(negSqrt);
double pImag = cimag(negSqrt);
SCALAR complex zespolona = pReal + pImag*I;
x1 = (-b-negSqrt)/2*a;
x2 = (-b+negSqrt)/2*a;
printf("x1 = %.2f + %.2fi\tx2 = %.2f + %.2fi\n", creal(x1), cimag(x1), creal(x2), cimag(x2));}

else if (delta == 0){
printf("Jeden pierwiastek rzeczywisty: x = %20.15lf\n", -b/(2*a) );}

else if (delta > 0) {
SCALAR temp = sqrt(delta);
SCALAR x1 = (-b-temp)/(2*a);
SCALAR x2 = (-b+temp)/(2*a);
printf("Dwa pierwiastki rzeczywiste: x1 = %20.15lf, x2 = %20.15lf\n", x1, x2);
```

Wnioski:

- for (;;) {} to pętla nieskończona. Chcąc, aby poprawnie działała w niej komenda scanf trzeba dać spację przed %c, inaczej wczyta nam enter jako znak.
- CTRL + C zakończy program.
- Używając konstrukcji switch należy pamiętać o stosowaniu break po każdym z przypadków.
- return(-1) zakończy pętlę.
- Funkcja *fabs* zwraca wartość bezwzględną liczby przekazanej jako argument.
- Porównując taką liczbę z dobranym parametrem TOLERANCJA, który ma bardzo małą wartość, możemy sprawdzić czy liczba nie jest zbyt bliska 0.
- Jeżeli liczba jest zbyt bliska 0 kompilator zaokrągla ją i pojawiają się niedokładne wyniki.
- Jak parametr b w równaniu kwadratowym ma bardzo dużą wartość to tracimy znaczące cyfry, mamy duży błąd względny.
- Uwzględnianie wszystkich przypadków w jednej konstrukcji sprawia, że kod staje się mniej czytelny.