Wstęp do programowania

Wykład 2. Instrukcje warunkowe.

Plan wykładu

1. Funkcje i procedury
2. Instrukcje warunkowe:
- if … else if … else …,
- switch.
3. Operatory
4. Tablice, wskaźniki

if ... else if ... else

```
double podziel(double a, double b){
  return a/b;
}
```

Ale co w przypadku, gdy `b=0`?

```
double podziel(double a, double b){
  if (b==0){ //sprawdzamy, czy b=0, jeśli tak wykonujemy poniższy kod
    printf("Nie można dzielić przez 0\n");
    return 0;
  }else  // w przeciwnym razie wykonujemy poniższy kod
    return a/b;
}
```

To co zapiszemy po znakach `//` (do końca linii) jest komentarzem – program nie zwraca na to uwagi

if ... else if ... else

Sprawdzamy, ile rozwiązań ma równanie: $a\,x^2+b\,x + c = 0$

```
int ileRozwiazan(double a, double b, double c){
  double delta = b*b - 4*a*c;
  if (delta>0)
    return 2;
  else if (delta==0)
    return 1;
  else
    return 0;
}
```

Podstawowe operatory logiczne

`==` operator porównania, np. `a==b`, w przypadku gdy wartości są równe zwraca liczbę dodatnią, w przeciwnym razie 0.

```
if (1) printf("To się wykona zawsze\n");
else printf("To się nie wykona nigdy\n");
```

`!=` operator nierówności, np. `a!=b`, w przypadku gdy wartości są różne zwraca liczbę dodatnią, w przeciwnym razie 0.

`!` operator zaprzeczenia, np. `!(a==b)` jest równoważne `(a!=b)`. `(!a)` jest dodatnie tylko gdy `a` będzie równe `0`.

`&&` operator logiczny „i”, np. `(a>1) && (a<=10)`, zwróci wartość dodatnią tylko gdy `a` będzie z przedziału `(1, 10]`.

`||` operator logiczny „lub”, np. `(a==7) || (a==8)`, zwróci wartość dodatnią tylko gdy `a` będzie równe `7` lub `8`.

Podstawowe operatory arytmetyczne

`=` operator podstawienia (przypisania), np. `a=b`, powoduje, że pod zmienną `a` zostanie skopiowana wartość ze zmiennej `b`. Wynikiem tej operacji jest kopiowana wartość

```
if (a=7) printf("To się wykona zawsze\n");
else printf("To się nie wykona nigdy\n");
```

`+`, `-`, `*`, `/` dodawanie, odejmowanie, mnożenie, dzielenie. W przypadku liczb całkowitych dzielenie zwraca liczbę całkowitą, zaokrągloną w dół: `7/3` daje w wyniku `2`, `-7/3`. daje w wyniku `-3`.

`%` dzielenie modulo, reszta z dzielenia liczb całkowitych: `7%3` daje w wyniku `1`.

`+=`, `-=`, `*=`, `/=` skrócona forma dodawania, odejmowania, mnożenia i dzielenia. Np. `a*=2` jest równoważne `a=a*2`. `a++` jest równoważne `a=a+1`. `a--` jest równoważne `a=a-1`.

Operator ?:

?:

przykład:

```
(a%2==0) ? printf("liczba parzysta") : printf("liczba nieparzysta");
```

Jest to równoważne konstrukcji

```
if (a%2==0)
  printf("liczba parzysta");
else
  printf("liczba nieparzysta");
```

switch...case

```
switch(a){
  case 1: printf("a = 1\n");
          break;
  case 2: printf("a = 2\n");
          break;
  default: printf("a > 2\n");
          break;
}
```

To jest równoważne:

```
if(a==1)
  printf("a = 1\n");
else if (a==2)
  printf("a = 2\n");
else
  printf("a > 2\n");
```

Przekazywanie zmiennych do funkcji

```
double mojaFunkcja(double a){
  a*=7; // a = 7*a;
  return a;
}

void main(){
  double a=3, b=5;
  printf("%f %f\n", mojaFunkcja(a), mojaFunkcja(b));
  printf("%f %f\n", a, b);
}
```

Argumenty funkcji są przekazywane przez wartość. Tworzone są nowe zmienne (lokalne), pod które podstawia się **wartości** przekazywanych argumentów. Po zakończeniu funkcji te zmienne są usuwane. Zmienna a w `mojaFunkcja` i zmienna `a` w `main` to różne zmienne, zapisane w innych miejscach w pamięci komputera.

Czy funkcja może zmodyfikować przekazaną do niej wartość? Tak, ale...

Tablice (ciągi)

```
void pomieszaj(float tablica[]){
  float t0 = tablica[0], t1 = tablica[1], t2 = tablica[2];
  tablica[0] = t2; tablica[1] = t0; tablica[2] = t1;
}

void main(){
  float a[3];  // tablica składająca się z trzech liczb typu float
  a[0] = 1.1; a[1] = 7.2; a[2] = 3.5;
  printf("%f %f %f\n", a[0], a[1], a[2]);
  pomieszaj(a);
  printf("%f %f %f\n", a[0], a[1], a[2]);
}
```

Tablice są przekazywane poprzez adres (wskaźnik) do pierwszego elementu tablicy. Wyrażenie `a[2]` jest więc interpretowane jako

Tablice (ciągi)

Tablice są obszarami pamięci w których kolejno zapisane są wartości określonego typu (np.. `int`, `float`, ...). Gdy tablica jest przekazywana do funkcji nie jest tworzona jej kopia, tylko przekazywany jest adres początku tablicy (jej pierwszego elementu). Wyrażenie `a[2]` jest więc interpretowane jako wartość, umieszczona pod adresem `(a + 2*` rozmiar zmiennej określonego typu`)`.

Dlatego, zmieniając elementy tablicy wewnątrz funkcji, elementy te pozostaną zmienione po zakończeniu tej funkcji.

Jeśli zmienna `a` jest adresem wartości określonego typu, to wartość wpisaną pod tym adresem dostajemy poprzez konstrukcję `*a`. Dlatego `a[2]` jest równoważne `*(a+2)`. Jeśli mamy zmienną `b` to jej adres dany jest przez `&b`.

Funkcja scanf

```
#include<stdio.h>

void main(){
  int a, b;
  printf("Podaj dwie liczby\n");
  scanf("%d", &a);
  scanf("%d", &b);
  printf("%d * %d = %d\n", a, b, a*b);
}
```

scanf, pozwala pobrać dane od użytkownika. Jako argumenty tej funkcji podaje się adresy (wskaźniki) zmiennych, gdzie mają zostać zapisane wprowadzane wartości.

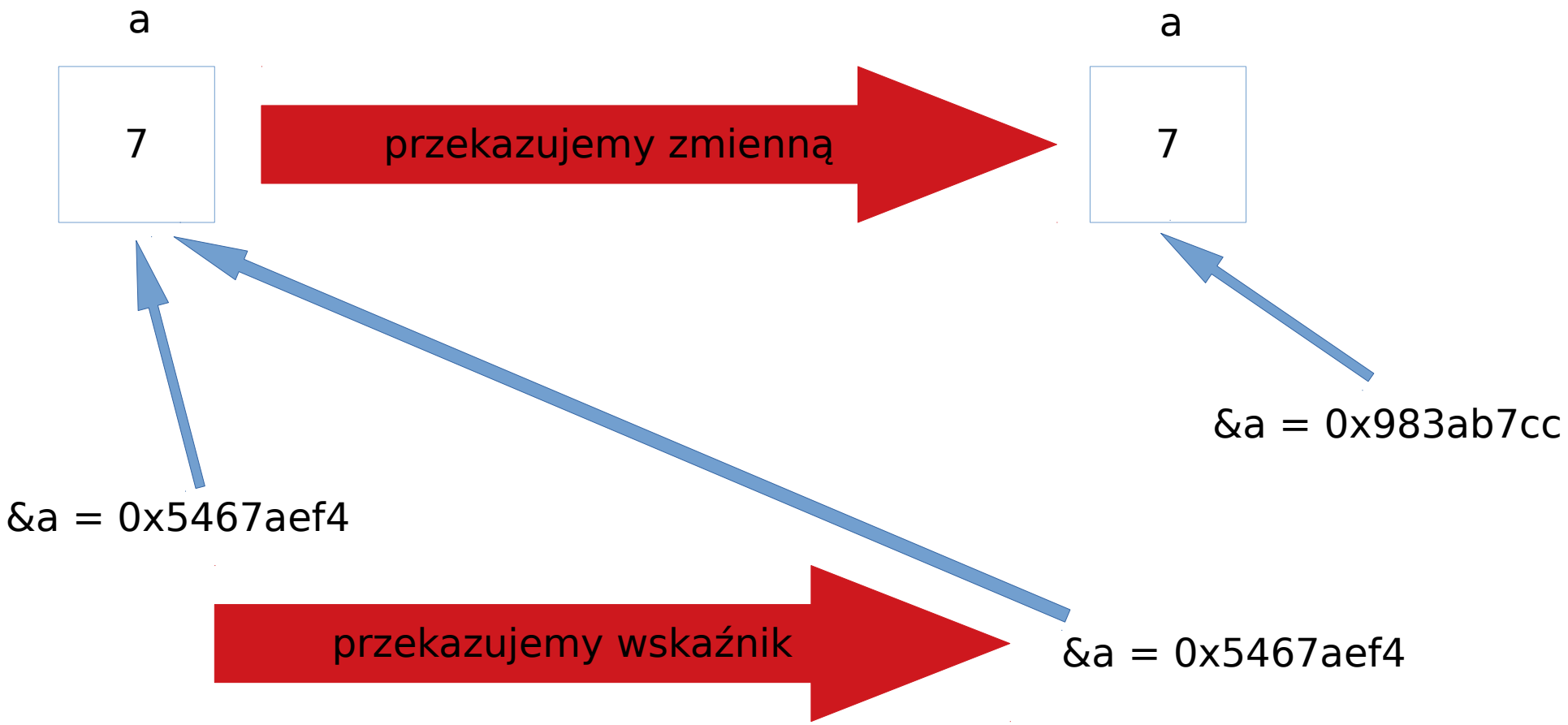
Przekazywanie zmiennych do funkcji

```
int ileRozwiazan(double a, double b, double c, double *x1, double *x2){
  double delta = b*b - 4*a*c;
  if (delta>0){
    *x1 = (-b+sqrt(delta))/(2*a);
    *x2 = (-b-sqrt(delta))/(2*a);
    return 2;
  }else if (delta==0){
    *x1 = -b/(2*a);
    return 1;
  }else
    return 0;
}
```

Wartości zapisane pod **adresami** `x1` i `x2` zostaną wypełnione przez funkcję `ileRozwiazan`.

Funkcja `sqrt()` znajduje się w bibliotece `math.h`. Aby korzystać z tej biblioteki w niektórych systemach trzeba użyć dodatkowej opcji kompilatora **-lm**.

Przekazywanie zmiennych do funkcji



Plan wykładu

Dziękuję za uwagę