Odločitveni stavki

Zakaj potrebujemo odločitvene stavke?

- Problemov, ki se jih da rešiti s preprostimi programi, kjer se stavki izvajajo lepo po vrsti, eden za drugim, je bolj malo.
- Skoraj v vsakem programu se moramo odločiti med več možnostmi, oziroma izvršiti določene stavke samo, če je izpolnjen nek pogoj.
- Odločitveni stavki zajemajo:
  - Pogojne izraze
  - Pogojne stavke
  - Stavek „switch“

Pogojni stavek `if-else`

- Stavek `if` preverja izraz pogoj.
- Če je pogoj izpolnjen (je torej ocenjen kot `true`), se bo izvršil blok stavkov `stavki1`.
- Stavek `if` je lahko dopolnjen s stavkom `else`, ki definira blok stavkov `stavki2`.
  Ti se bodo izvršili,če pogoj ni bil izpolnjen (je torej ocenjen kot `false`).

```
if (pogoj) {
     stavki1;
}
Else {
     stavki2;
}
```

Veriženi stavki `if-else`

- Zaporedno lahko preverjamo tudi več pogojev, izvede pa se le skupina stavkov, ki ustreza prvemu izpolnjenemu pogoju:

```
if (prvi logični pogoj) {
     prvi stavki
}
else if (drugi logični pogoj) {
     drugi stavki
}
else if (tretji logični pogoj) {
     tretji stavki
}
else if (zadnji logični pogoj) {
     zadnji stavki
}
```

Gnezdeni stavki `if`

- Med stavki v zavitih oklepajih imamo lahko tudi stavke tipa if. Tako dobimo na primer naslednjo obliko:

```
if (prvi logični pogoj) {
     stavki
     if (drugi logični pogoj) {
            stavki
     }
     stavkov
}
```

Pogojni stavek `switch`

- Izbira med več možnostmi, ki jo določa stanje neke spremenljivke
- Izvedejo se vse opcije od izbrane naprej

- Če želimo samo izbrano, moramo izvajanje prekiniti (z *`break`*)

- Stavek `switch` zelo poenostavi programiranje, če bi morali uporabiti večkratno zaporedje stavkov `if`

Pogojni stavek `switch`

```
switch (izraz)
{
case vrednost1:
stavki1
break
case vrednost2:
stavki2
break
...
default: //ta del lahko izpustimo
stavki
}
```

Pogojni stavek `switch` – `if-else`

```
switch (izraz)
{
case vrednost1:
stavki1
break
case vrednost2:
stavki2
break
...
default:
stavki
}
```

```
if (izraz == vrednost1)
{
        stavki1
}
else if (izraz == vrednost2)
{
        stavki2
}
...
else
{
        stavki
}
```

Zanke

Zakaj potrebujemo zanke?

- Kadar želimo določen stavek ponoviti večkrat, ga zapišemo v posebnem stavku, imenovanem zanka.
- Program bo izvrševal dani stavek, dokler je izpolnjen nek pogoj.
- Ko pogoj ne bo več izpolnjen, se bo program nadaljeval z naslednjim stavkom, ki sledi zanki.
- Poznamo več vrst zank:
  - `while`
  - `for`
  - `do`

Zanka `while`

- Na začetku zanke se testira pogoj; zanka se ponavlja, dokler je izpolnjen.
- Če na začetku ni izpolnjen, se zanka nikoli ne izvede.

```
while (pogoj)
{
     stavek
}
```

Zanka `for`

- zanka je predvidena za iteracije, v katerih se najprej inicializira neka spremenljivka, ki se nato povečuje in testira njeno stanje
- ko je izpolnjen pogoj, se zanka konča

```
for (start; pogoj; korak)
{
    stavek
}
```

Zanka `for`

- V `for` zanki lahko vsi ali nekateri izrazi manjkajo:
  - Če manjka `pogoj`, velja, da je pogoj vedno izpolnjen.
  - Manjka lahko tudi telo
    ```
    for(i=1;i<=5;i++) {}
    ```
  - Odsotnost izrazov in uporaba posebnih operatorjev:int for
    ```
    (;i<=10;)
    ```
  - Odsotnost vseh izrazov, kompleksen pogoj:
    ```
    for(;;)
    if(pogoj)
    ```

Zanka `for` - gnezdenje

- Zanke `for` lahko gnezdimo

```
for (start; pogoj; korak)
{
     for (start; pogoj; korak)
     {
     for (start; pogoj; korak)
            stavek
     }
}
```

Primerjava `for` - `while`

```
var i=1
while (i<10)
{
        ++i
        …
}
```

```
for (var i=1; i<10; ++i)
{
…
}
```

Zanka do-(while)

- ```
  do
  {
       stavki
  }
  while (pogoj)
  ```
  ```

Stavek `break`

- Stavek `break` uporabljamo v telesih zank `while`, `for` in `do`.
- **Prekine izvedbo stavkov v zanki ali bloku stavkov. Sledi izvedba stavka ZA tem blokom.**

Stavek `continue`

- Stavek `continue` uporabljamo v telesih zank `while`, `for` in `do`.
- **Prekine tekočo iteracijo v zanki, same zanke pa NE prekine.**