8. Stavki

Stavki v C++

- poznamo več vrst stavkov
  - prireditveni stavki
  - proceduralni stavki
  - pogojni stavki
  - izbirni stavki
  - iterativni stavki
  - deklarativni stavki

- deklaracija spremenljivke je stavek

8.1. Prireditveni stavki (1)

- različne oblike prireditvenih stavkov
- najbolj tipični – prirejanje vrednosti spremenljivki

spremenljivka = izraz;

PRIMER:

```
a = b + 1;
```

- izvajanje prireditvenega stavka
  - najprej se ovrednoti desna stran prirejanja
  - vrednost se pretvori v tip kompatibilen s tipom spremenljivke
  - ta vrednost se shrani v spremenljivko na levi strani

8.1. Prireditveni stavki (2)

- najbolj tipični – prirejanje vrednosti spremenljivki

  spremenljivka = izraz;

- leva stran prireditvenega stavka mora biti t.i. **lvrednost** (angl. lvalue)
  - predstavlja lokacijo v pomnilniku, kamor lahko shranimo in od koder lahko preberemo vrednost
- preproste spremenljivke so lvrednosti
- prirejanje se obnaša kot funkcija
  - vrača vrednost, ki se prireja spremenljivki
  - zato lahko imamo v C++ prirejanje znotraj prirejanja

Primer 01


```
#include <iostream>
using namespace std;

int main()
{
    int a = 2;
    int b = 3;
    int c, d;

    d = 5;
    c = a + d;

    cout << "c = " << c << endl;
    return 0;
}
```

Primer 02


```
#include <iostream>
using namespace std;

int main()
{
    int x = 10;
    x = x + 1;
    cout << "x = " << x << endl;
    return 0;
}
```

Operatorji prirejanja

- poznamo operatorje prirejanja
  - klasični operator prirejanja
    - `=`
  - ki so združeni z aritmetičnimi operacijami
    - `+=  -=  *=  /=  %=`

PRIMERI:

`a += b;`	enako kot	`a = a + b;`
`a *= a+b;`	enako kot	`a = a*(a+b);`

Operatorja ++ in --

- operator povečanja ++ in operator zmanjšanja --
  - oba se pojavita v predponski in priponski obliki
  - povečanje/zmanjšanje za 1
- v predponski obliki
  - vrednost se najprej poveča/pomanjša, nato vrednost vrne
- v priponski obliki
  - vrne vrednost spremenljivke, nato poveča/pomanjša vrednost

PRIMER:

```
a = ++b;        enako kot    b = b + 1; a = b;
a = b++;        enako kot    a = b; b = b + 1;
```

Primer 03


```
#include <iostream>
using namespace std;

int main()
{
    int x = 10;
    cout << ++x << endl;
    cout << x++ << endl;
    cout << x << endl;
    return 0;
}
```

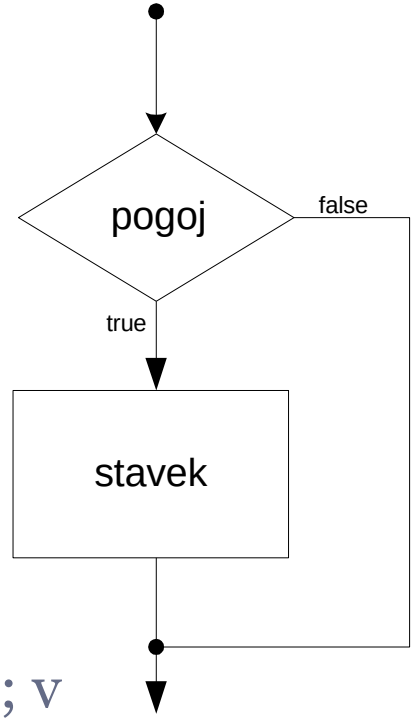
8.2. Prazen in sestavljeni stavek

- prazen stavek
  - ne naredi ničesar
  - samo podpičje
- sestavljeni stavek
  - skupina več stavkov zapisanih med { }
  - več stavkov združuje v sintaktično gledano en stavek

8.3. `if` in `if-else` stavek

- namen `if` stavka: *neko dejanje izvedemo le ob določenem pogoju*
- pogoj je poljuben izraz tipa bool
- sintaksa `if` stavka

```
if (pogoj)
  stavek;
```



- Kako deluje?
  - če se `pogoj` ovrednoti na `true`, se izvede `stavek`; v nasprotnem primeru se `stavek` preskoči

Logični izrazi

Vrsta	Pomen	Simbol
relacijski operatorji		
	manj kot	<
	več kot	>
	manj ali enako kot	<=
	več ali enako kot	>=
operatorji enakosti		
	enako kot	==
	različno kot	!=
logični operatorji		
	negacija	!
	logični in	&&
	logični ali	||

Relacijski, logični in operatorji enakosti (2.)

- vsi omenjeni operatorji z izjemo negacije so binarni
- izraz se ovrednoti na vrednost tipa **bool**
  - ima vrednost bodisi **true**, bodisi **false**

- kjer prevajalnik pričakuje vrednost tipa **bool**, se aritmetični izrazi samodejno pretvorijo
  - vrednost `0` se pretvori v **false**
  - neničelne vrednosti pretvori v **true**

Relacijski, logični in operatorji enakosti (3.)

POZOR:

- operator primerjanja enakosti (==) se pogosto zamenja z operatorjem prirejanja (=)

  PRIMER:

  namesto

```
if (i == 1)
  // naredi nekaj
```

  napišemo

```
if (i = 1)
  // naredi nekaj
```

- tovrstnim napakam se lahko včasih izognemo
  - konstanto pišemo na levo stran operatorja: `(1 == i)`

Logični operatorji (1)

- logični operatorji `!`, `&&` in `||` dajo rezultat tipa `bool`

- NEGACIJA
  - uporabimo nad poljubnim izrazom

izraz	!(izraz)
true	false
false	true

Logični operatorji (2)

- logični in (&&)
  - veljati morajo vsi pogoji, ki jih združuje

izraz1	izraz2	izraz1 && izraz2
false	false	false
false	true	false
true	false	false
true	true	true

Logični operatorji (3)

- logični ali (||)
  - veljati mora vsaj en od pogojev, ki jih združuje

izraz1	izraz2	izraz1 || izraz2
false	false	false
false	true	true
true	false	true
true	true	true

Logični operatorji (4)

- vrednotenje **s kratkim stikom**
  - vrednotenje logičnega izraza se ustavi takoj, ko je rezultat znan

PRIMER:

Imejmo izraza `i1` in `i2`. Naj ima izraz `i1` vrednost `false`.

V izrazu `(i1 && i2)` se izraz `i2` ne vrednoti.

Podobno, če je vrednost `i2` enaka `true`, se v izrazu `(i2 || i1)` izraz `i1` ne vrednoti.

if stavek - primer

V spremenljivki x imamo shranjeno celo število. Želimo imeti absolutno vrednost tega števila.

```
if (x < 0) {
        x = -x;
}
```

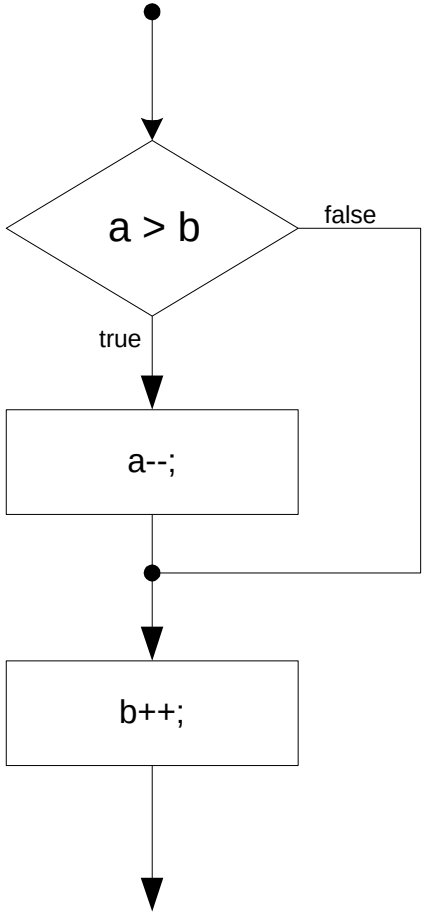
Zavite oklepaje smo pisali, kljub temu, da imamo samo en stavek. To ni narobe, je pa narobe, če oklepaje izpustimo, kadar želimo pod pogojem izvesti več stavkov. Glej naslednji primer.

if stavek - primer

Imamo dve celi števili *a* in *b*. Če je *a* večji od *b*, želimo zmanjšati *a* in povečati *b*.

```
if (a > b)
        a--;
        b++;
```

Ker ni zavitih oklepajev, samo prvi stavek spada k if stavku, drugi se izvede vedno.

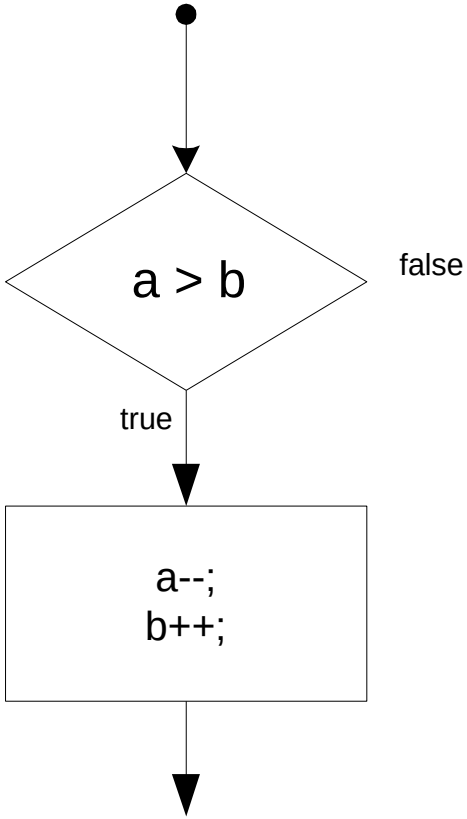


if stavek - primer

Imamo dve celi števili *a* in *b*. Če je *a* večji od *b*, želimo zmanjšati *a* in povečati *b*.

```
if (a > b) {
        a--;
        b++;
}
```

Z zavitimi oklepaji povemo, da oba stavka spadata k if stavku in se izvedeta, ko je določen pogoj.



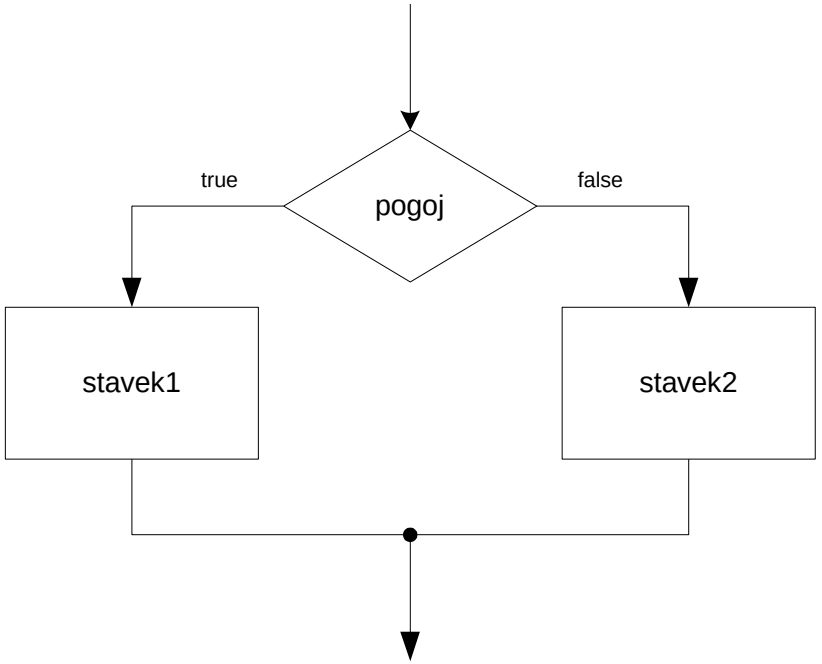
if stavek - primeri


```
if (ura >= 12 && ura < 19)
  cout << "Popoldan je." << endl;

if (temperatura > 0)
  cout << "Nad ničlo je." << endl;
cout << "Temperatura je " << temperatura << " stopinj Celzija." << endl;

if (rezultat > 70 && rezultat < 80)
  cout << "Opravil(a) si izpit.";
  ocena = 8;

if (rezultat > 70 && rezultat < 80) {
  cout << "Opravil(a) si izpit.";
  ocena = 8;
}
```

`if-else` stavek

- namen `if-else` stavka: *neko dejanje izvedemo, če je pogoj resničen, v nasprotnem primeru pa izvedemo neko drugo dejanje*
- sintaksa if-else stavka:

```
if (pogoj)
  stavek1;
else
  stavek2;
```



- Kako deluje?
  - če se `pogoj` ovrednoti na `true`, se izvede $\texttt{stavek}_1$; v nasprotnem primeru (pogoj je `false`) se izvede $\texttt{stavek}_2$

if-else stavek – primer


```
if (x < y)
   min = x;
else
   min = y;
cout << "min = " << min;
```

Če se `x<y` ovrednoti na `true`, spremenljivki `min` priredimo vrednost `x`. Če se `x<y` ovrednoti na `false`, spremenljivki `min` priredimo vrednost `y`.

Ta stavek se izvede v vsakem primeru, saj ne spada k if-else stavku.

Primer 04


```
/*
   Program izpiše oceno glede na rezultat.
*/
#include <iostream>
using namespace std;

int main()
{
    int rezultat,  // med 0 in 100
        ocena = 0; // 0 ter med 5 in 10
                   // vrednost 0 za neveljavne

    cout << "Vnesi dosezen rezultat: ";
    cin >> rezultat;

    if (rezultat > 100) {
        cout << "Neveljavna vrednost za rezultat.";
    }
    else if (rezultat == 100) {
        cout << "Dosezene so vse tocke.";
        ocena = 10;
    }
    else if (rezultat >= 90) {
        cout << "Odlicno.";
        ocena = 10;
    }
    else if (rezultat >= 80) {
        cout << "Prav dobro.";
        ocena = 9;
    }
    else if (rezultat >= 70) {
        cout << "Prav dobro.";
        ocena = 8;
    }
    else if (rezultat >= 60) {
        cout << "Dobro.";
        ocena = 7;
    }
    else if (rezultat >= 50) {
        cout << "Zadostno.";
        ocena = 6;
    }
    else if (rezultat >= 0) {
        cout << "Nisi opravil(a).";
        ocena = 5;
    } else
        cout << "Neveljavna vrednost za rezultat.";

    cout << endl << "Tvoja ocena je " << ocena;
    return 0;
}
```

8.4. `switch` stavek

- `switch` stavek je izbirni stavek
  - lahko ga primerjamo z več zaporednimi `if-else` stavki
- sintaksa `switch` stavka je

```
switch (izraz)
  stavek;
```

- pri tem je `stavek` sestavljeni stavek, ki vsebuje več `case` oznak in opcijsko vrednost `default`
- sintaksa `case` oznake

```
case celostevilska_konstanta:
```

- sintaksa `default` oznake

```
default:
```

`switch` stavek (2)

- vse konstantne vrednosti v `case` oznakah morajo biti enolične
- izvajanje `switch` stavka je sledeče:
  1. Ovrednoti izraz.
  2. Skoči na tisto `case` oznako, ki vsebuje enako konstantno vrednost kot je vrednost izraza. Če taka `case` oznaka ne obstaja, se izvajanje premakne na oznako `default`. Če oznaka `default` ne obstaja, se izvajanje nadaljuje za `switch` stavkom.
  3. Če se `switch` stavek ne preneha izvajati na koraku `2`, se izvedejo vsi stavki od tistega mesta naprej, bodisi do prvega `break` stavka, bodisi do konca `switch` stavka.

Primer 05


```
/*
   Program izpiše oceno glede na rezultat.
*/
#include <iostream>
using namespace std;

int main()
{
    int rezultat,  // med 0 in 100
        ocena = 0; // 0 ter med 5 in 10
                   // vrednost 0 za neveljavne

    cout << "Vnesi dosezen rezultat: ";
    cin >> rezultat;

    if (rezultat > 100 || rezultat < 0) {
        cout << "Neveljavna vrednost za rezultat.";
    } else
    switch (rezultat/10) {
        case 10:
            cout << "Dosezene so vse tocke.";
            ocena = 10;
            break;
        case 9:
          cout << "Odlicno.";
          ocena = 10;
          break;
        case 8:
          cout << "Prav dobro.";
          ocena = 9;
          break;
        case 7:
          cout << "Prav dobro.";
          ocena = 8;
          break;
        case 6:
          cout << "Dobro.";
          ocena = 7;
          break;
        case 5:
          cout << "Zadostno.";
          ocena = 6;
          break;
        case 4: case 3: case 2:
        case 1: case 0:
          cout << "Nisi opravil(a).";
          ocena = 5;
          break;
    }

    cout << endl << "Tvoja ocena je " << ocena <<
    endl;
    return 0;
}
```