Zanke

torek, 29. oktober 2024 16:54

Omogočajo, da del kode večkrat ponovimo, da ne pišemo ponavljajoče se kode.

for - ko vnaprej vemo, kolikokrat želimo nekaj ponoviti
while - ne vemo točno, kolikokrat se bo koda ponovila - le, da pogoj drži (tudi, če pogoj ne drži*)
do... while - podobno kot while, ampak bo zagotovo delovala vsaj 1x

FOR zanka:

```
for (inicializacija; pogoj; korak) {
}
```

začetna točka, spr. i | pove, kdaj naj se zanka konča | nadaljuje krog

Na primer:

```
int main() {
    for (int i=1; i<=5; i++) {
        cout << i <<;
    }
}
```

WHILE zanka:

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

↳ če drži, se bo izvajala koda, sicer ne

Na primer:

```
int main () {
    int i=1;
    While (i <=5) {
        cout << i <<;   ← najprej izpis
        i++;            ← nato inkrementacija
    }
}
```

DO... WHILE zanka:

```
do {
} while (pogoj);
```

zagotavlja, da se bo koda izvedla vsaj 1X

Na primer:

```
int main () {
    int i = 1;
    do {
        cout << i <<;
        i++;
    } while (i<=5);
    return 0;
}
```

break ... takoj zaključi (najbolj notranjo) zanko (npr. da imamo znotraj if stavek)
continue ... nazaj na vrh zanke

Gnezdene zanke:

Funkcije

petek, 18. oktober 2024 19:35

- del kode, ki opravlja določeno nalogo in jo lahko uporabimo (kličemo) večkrat.

Deli funkcije:

- ime
- parametri - znotraj ()
- return type
- telo - znotraj {}

definirane morajo biti PREDEN so uporabljene!

Primer:

```
int sestej(int a, int b) {
    int sum = a+b;
    return sum;
}

int main() {
    int vsota = sestej(3, 4);
    cout << "Vsota je " << vsota << endl;
    return 0;
}
```

* Obstaja pa drug način:

```
void funkcija();          ← deklaracija

int main() {
    funkcija();           ← klic
    return 0;
}

void funkcija() {         ← definicija
    cout << "Hello World" << endl;
}
```

PARAMETRI: Spremenljivke znotraj funkcije

a. Nepravi: zgolj mesta, kamor funkcija pričakuje podatke ("actual")
(v zg. primeru sta to a in b)

b. Pravi: dejanske vrednosti, ki jih podamo funkciji ob klicu ("formal")
(v zg. primeru 3 in 4)

„Default value"

```
void funkcija(string drzava = "Slovenija") {
    cout << drzava << endl;

int main() {
    funkcija("Francija");   → Francija
    funkcija();             → Slovenija
}
```

Več parametrov: lahko jih je kolikorkoli

Vračanje vrednosti:

- void ne vrne vrednosti
- return

npr.

```
int funkcija(int a, int b) {
    return a+b;
}
```

PRENOS

a. po vrednosti: kopira se vrednost dejanskega parametra, izvirne vrednosti se ne spremenijo.
↳ spremembe znotraj funkcije ne vplivajo na vrednosti zunaj nje
↳ primer:

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

void increment(int num) {
    num++;
    cout << num << endl;
}

int main() {
    int number = 5;          ← se ne spremeni, le kopira v funkc.
    increment(number);       ← ker number damo v funkc., se inkrementira
    cout << number << endl;  ← originalna nespremenjena
    return 0;
}
```

izpis: 6
5

b. prenos po referenci: referenca je dejanski parameter, ki se prenese v funkcijo
(v spominu kažeta na isto vrednost)
↳ spremembe na parametrih znotraj funkcije se poznajo zunaj nje
↳ primer od prej:

```
void (int& num) {...
```

izpis: 6
6

Polja

petek, 08. november 2024 20:30

Polje („array") je zaporedje vrednosti istega tipa, ki so shranjene ena poleg druge v spominu računalnika.
↳ do elementov dostopamo z **indeksi** (n elementov ⇒ 0-(n-1))

Velikost polja = koliko elementov vsebuje polje (da se v pomnilniku rezervira dovolj mest)

velikost polja je stalna!
C++ ne preverja, ali je indeks znotraj meja polja, zato je treba paziti

```
int main () {
  int stevila [5] = {3, 5, 7, 9, 11};   // definiramo polje velikosti 5 in njegove elemente
  int tretjiElement = stevila[2];       // dostop do elementa; tretjiElement je 7
```
(indeksi: 0 1 2 3 4)

→ Uporaba FOR zanke:

Za dostop do elementov:
```
for (int i = 0; i < 5; i++) {
  cout << "Element na indeksu " << i << " je " << stevila[i] << endl;
}
```

Za polnjenje polja:
```
const int velikostPolja = 4;      // konst. spr. za velikost
int test [velikostPolja];         // dodelitev prostora

int vsota = 0;

for (int i = 0; i < velikostPolja; i++) {     // polnjenje polja, i predstavlja indeks
  test [i] = i * i;

  cout << "test [" << i << "] = " << test [i] << endl;     // izpisalo bo:
  vsota += test [i];
}

cout << "Vsota je " << vsota << endl;
return 0;
}
```
test [0] = 0
test [1] = 1
test [2] = 4
test [3] = 9
Vsota je 14.

Prenos polja v funkcijo:

Primer: funkcija za seštevanje

```
const int velikostPolja = 4;

int sestej (int polje [velikostPolja]) {
    int vsota = 0;
    for (int i = 0; i < velikostPolja; i++) {
        vsota += polje [i];
        return vsota;
    }
}

int main () {
  int prvo [velikostPolja];
  int drugo [velikostPolja],

  for int (int i = 0; i < velikostPolja; i++) {
    prvo [i] = i + 1;
    drugo [i] = (i+1) * (i+1);
  }

  cout << "Vsota prvega polja je " << sestej (prvo) << endl;
  cout << "Vsota drugega polja je " << sestej (drugo) << endl;

  return 0;
}
```

Iskanje elementov v polju:

- linearno iskanje („linear search")
  ↳ zaporedoma primerjamo vsak el. polja z iskano vred. (ključ)

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

int isciLinearno (int polje [], int velikost, int kljuc) {
    for (int i=0; i < velikost; i++) {
        if (polje [i] == kljuc) {
            return i;
        }
        return -1;
    }
}

int main () {
    const int vel_polja = 100;
    int soda_stevila [vel_polja];

    for (int i = 0; i < vel_polja; i++) {
        soda_stevila [i] = i * 2;
    }

    int kljuc, rezultat;
```

SORTIRANJE POLJA:

→ Bubble sort algoritem

i=0	5	6	1	3
i=1	5	6 ↔	1	3
i=2	5	1	6 ↔	3
i=3	5 ↔	1	3	6
i=4	1	5 ←	3	6
i=5	1	3	5	6