RAČUNALNIŠKI PRAKTIKUM

doc. dr. ANDREJ TARANENKO

Kdo bo z vami?

Predavatelj:

- dr. Andrej Taranenko
- andrej.taranenko@uni-mb.si
- kabinet: 0/95
- govorilne ure: http://matematika-racunalnistvo.fnm.uni-mb.si/

Asistentka:

- Mojca Bračič
  - ostale podrobnosti od nje

Obveznosti in ocenjevanje

- Naloge (vaje)
  - 20 %
  - kaj je potrebno, bo povedala Mojca
- Pisni izpit - problemi
  - 40 %
  - problemi – reševanje na papir
- Pisni izpit - teorija
  - 40 %
  - teorija – snov s predavanj

Prisotnost na vajah je obvezna,
na predavanjih zaželena!

Osnovna literatura

- Viljem Žumer, Janez Brest: *Strukturirano in objektno usmerjeno programiranje v C++ (1. del)*, FERI UM.
- Viljem Žumer, Janez Brest: *Objektno programiranje v C++*, FERI UM.
- Edward Scheinerman: *C++ for mathematicians*, Chapman & Hall
- Derek M. Capper: *Introducing C++ for scientists, engineers and mathematicians*, Springer
- splet ...

http://matematika-racunalnistvo.fnm.uni-mb.si/

- ŠTUDIJ
  - Moji predmeti
    - Študijski program: Matematika / Izobraževalna matematika
    - Predmet: Računalniški praktikum
- ČLANI ODDELKA
  - Mojca Bračič
  - Andrej Taranenko

KRATKA PONOVITEV OSNOV C++

SPREMENLJIVKE
POGOJNI STAVKI
ZANKE
FUNKCIJE
POLJA
NIZI

Spremenljivke (1)

7

- spremenljivke uporabljamo za shranjevanje podatkov, imenovali jih bomo tudi **objekti**
- vsaka spremenljivka ima določen
  - podatkovni tip ... vrsta podatka, ki ga objekt predstavlja
  - ime ... s tem imenom se skličemo na podatek
- deklaracija spremenljivke
  - stavek, s katerim uvedemo nov pojem – spremenljivko
- definicija spremenljivke
  - kadar z deklaracijo spremenljivke ustvarimo tudi sam objekt

Spremenljivke (2)

8

- deklaracija spremenljivke

  tip ime;
- podatkovni tipi, ki smo jih spoznali:
  - int cela števila
  - double realna števila
  - char 1 znak
  - bool logična vrednost (true/false)
  - tip[] polje elementov tipa tip
  - string niz znakov

Prireditveni stavek

9

- stavek, s katerim povemo, kateri podatek spremenljivka predstavlja
- pravimo, da priredimo vrednost
- operator =
- sintaksa prireditvenega stavka:

  ime_spremenljivke = izraz;

spremenljivka, ki je že deklarirana

izraz mora biti enakega oz. kompatibilnega tipa kot spremenljivka na levi strani enačaja

Podatkovni toki

10

- podatkovni vir, s katerega dobivamo podatke oz. na katerega podatke pošiljamo
- vhodni podatkovni tok – podatke dobivamo
  - npr. tipkovnica – cin
- izhodni podatkovni tok – podatke pošiljamo
  - npr. konzola na ekranu – cout
- tekstovni podatkovni tok – podatke dobivamo / pošiljamo v obliki teksta

Operatorja << in >>

11

- z << pošljemo podatek na podatkovni tok
  - npr. cout << "To je stavek." << endl;
- z >> pridobimo podatek s podatkovnega toka
  - npr. cin >> stevilo;

Stavki v C++

12

- večina stavkov v C++ se zaključi s podpičjem
- poznamo več vrst stavkov
  - deklaracijski – napovemo nekaj novega
  - prireditveni – priredimo vrednost
  - pogojni – if stavek
  - izbirni – switch stavek
  - iterativni – zanke (while, do..while, for)
  - ...

Vejitveni stavki

13

- pogojni stavek (if)
  - dosežemo, da se stavek oz. pa skupina stavkov izvede le ob določenem pogoju
- izbirni stavek (switch)
  - preverjamo enakost neke spremenljivke številnega tipa na določene vrednosti

if stavek – 1. oblika

14

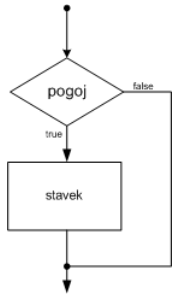
- poznamo več oblik
- 1. oblika ima naslednjo sintakso

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

- beremo tako:

  Če je res, kar pravi pogoj, potem izvedi stavek.

- stavek – je en stavek v C++ ali sestavljen stavek
- sestavljen stavek – več stavkov znotraj zavitih oklepajev



if stavek - 2. oblika

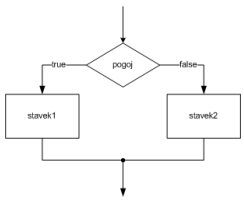
15

- 2. oblika ima naslednjo sintakso

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

- beremo tako:

  Če je res, kar pravi pogoj, potem izvedi stavek1, v nasprotnem primeru izvedi stavek2.



switch stavek

16

```
switch ( test ) {
  case vrednost1 :
      // test == vrednost1
      ...
      break;
  case vrednost2 :
      // test == vrednost2
      ...
      break;
  default:
      // vse druge vrednosti
      ...
}
```

izraz, ki se ovrednoti na številno vrednost

nato se ena za drugo preverjajo naštete vrednosti – torej ali je test enak kateri od teh vrednosti

če je enak, izvedi stavke pri tisti vrednosti

stavek break, skoči nemudoma ven iz switch stavka (ne preverja nadaljnjih vrednosti)

možnost default – če test ni enak nobeni od prej naštetih vrednosti, izvede te stavke. Po potrebi lahko možnost default izpustimo.

Zanke

17

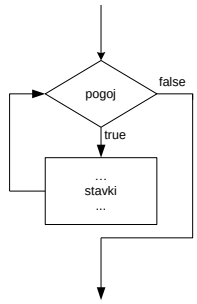
- z zankami dosežemo, da se stavek oz. skupina stavkov zaporedoma večkrat izvede
- kolikokrat se izvede je odvisno od
  - v naprej znanega števila ponovitev ali
    - najbolj primerna **for** zanka
  - nekega splošnega pogoja
    - najbolj primerni **while** ali **do..while** zanki

while zanka

18

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

1. Ovrednoti pogoj. Če je pogoj resničen (*true*), nadaljuj na koraku 2. V nasprotnem primeru nadaljuj za zanko.
2. Izvedi stavke.
3. Skoči na korak 1.

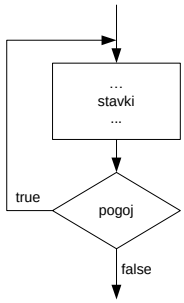


do...while zanka

19

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

1. Izvedi stavke.
2. Ovrednoti pogoj. Če je pogoj resničen (*true*), skoči na korak 1. V nasprotnem primeru nadaljuj za zanko.

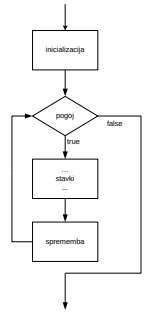


for zanka

20

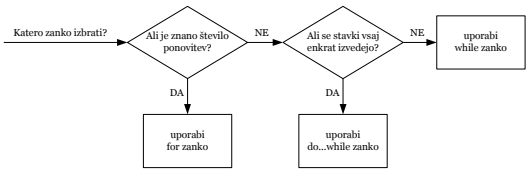
```
for (inicializacija; pogoj; sprememba) {
    stavki;
};
```

1. Izvedi inicializacijo.
2. Ovrednoti pogoj. Če je pogoj resničen (*true*), skoči na korak 3. V nasprotnem primeru nadaljuj za zanko.
3. Izvedi stavke.
4. Izvedi spremembo.
5. Skoči na korak 1.



zanke – katero naj uporabim?

21



stavka `break` in `continue`

22

- stavek `break` lahko uporabljamo le
  - znotraj `switch` stavka
  - znotraj poljubne zanke
- stavek `break` nemudoma prekine izvajanje zanke oz. switch, v katerih se pojavi
- stavek `continue` lahko uporabljamo le
  - znotraj poljubne zanke
- stavek `continue` nemudoma skoči na preverjanje pogoja v zanki, v kateri se pojavi

Struktura programa v C++

23

- s stališča strukturiranega programiranja
  - program je razdeljen na knjižnice in funkcije
  - v knjižnicah so definirani
    - razni objekti
    - podatkovni tipi
    - funkcije
    - razredi
  - uporabimo jih lahko, če v program vključimo knjižnico
- vsak program ima funkcijo `main()`
  - v tej funkciji se prične izvajanje programa
  - vrača celo število, ki operacijskemu sistemu pove, ali je prišlo do napake

funkcije v C++ (1)

24

- lastno funkcijo moramo
  - deklarirati preden jo uporabimo in jo definiramo kjerkoli v programu
  - definirati preden jo uporabimo
- funkcjo definiramo na naslednji način:

```
tip imeFunkcije(formalni parametri) {
  ...
  return vrednost; // če funkcija vrača vrednost
}
```

funkcije v C++ (2)

25

```
tip imeFunkcije(formalni parametri) {
  ...
  return vrednost; // če funkcija vrača vrednost
}
```

`tip`	vrsta rezultata, ki ga funkcija vrača
`imeFunkcije`	s tem imenom kličemo funkcijo, izvedemo stavke v funkciji
`formalni parametri`	so opis podatkov, ki jih posredujemo funkciji ob klicu

parametri

26

- so podatki, brez katerih v funkciji ne moremo rešiti naloge
- opišemo jih s t. i. formalnimi parametri
- formalni parametri so
  - deklaracije (lokalnih) spremenljivk
  - povedo nam
    - število podatkov, ki jih potrebujemo
    - kakšnega tipa morajo biti podatki ob klicu funkcije
    - vrstni red podatkov ob klicu funkcije

Primer 1 – klic funkcije

27

- v knjižnici `cmath` je definirana funkcija `sqrt`

```
double sqrt(double x);
```

nam pove, da moramo ob klicu funkcije podati en podatek, ki je realno število

- uporaba funkcije:

```
double y;
```

je izraz tipa `double`

```
y = sqrt(17.54);
```

Primer 2

28

- napiši funkcijo, ki izračuna največji skupni delitelj dveh celih števil
- Def: Največji skupni delitelj dveh celih števil a in b je tako največje celo število, ki deli oba a in b.
  - nsd(a,b)..... oznaka za največji skupni delitelj.
  - nad(a,b) = nsd (-a,b) = nsd(a,-b) = nsd (-a,-b)
  - nsd(a,0) = |a|
  - ni definirano: nsd (0,0)

Primer 2

29

```
int nsd(int a, int b) {
  a = abs(a);
  b = abs(b);

  // drugo st. bo manjse
  if (a < b) {
    int t = a;
    a = b;
    b = t;
  }

  if (a==0 && b==0) {
    cout << "ni def.";
    return 0;
  }

  if (b==0)
      return a;

  // preverimo vse možne
  // kandidate
  for(int d=b; d>=1; d--)
    if (a%d==0 && b%d==0)
      return d;
}
```

Primer 2 – hitreje (Evklidov algoritem)

30

TRDITEV:

Naj bosta a in b celi števili, $a>0$, $b>0$, in $c = a \bmod b$. Potem je nsd(a, b) = nsd(b, c).

Primer

```
   nsd(80, 25) =              80 mod 25 = 5
= nsd(25, 5)  =               25 mod 5  = 0
= nsd(5, 0) =
= 5
```

Primer 2 – hitreje (Evklidov algoritem)

31

```
int nsd(int a, int b) {
  a = abs(a);
  b = abs(b);

  // drugo st. bo manjse
  if (a < b) {
    int t = a;
    a = b;
    b = t;
  }

  if (a==0 && b==0) {
    cout << "ni def.";
    return 0;
  }

  if (b==0)
      return a;

  // Evklidov algoritem
  do {
    int novi_a = b;
    int novi_b = a%b;
    a = novi_a;
    b = novi_b;
  } while (b!=0);
  return a;
}
```

Primer 2 – klic funkcije

32

- ne glede na to, kako smo največji skupni delitelj poiskali, funkcijo kličemo na enak način

```
int main() {
  int st1, st2;
  cout << "Vnesi dve celi stevili: ";
  cin >> st1 >> st2;
  cout << "Najvecji skupni delitelj stevil " << s1;
  cout << " in " << st2 << " je " << nsd(st1, st2) << endl;
  return 0;
}
```

Naključna števila (1)

33

- generator naključnih števil
  - je funkcija, ki vrne na videz naključno število, v bistvu pa se rezultat določi deterministično.
- s semenom generatorja
  - določimo na katerem mestu v zaporedju vračanih števil funkcija prične vračati vrednosti.
- seme generatorja naključnih števil določimo s funkcijo `void srand(int seme)`
- seme običajno določimo s sistemsko uro
- funkcija je definirana v knjižnici **`cstdlib`**

Naključna števila (2)

34

- za delo z naključnimi števili moramo vključiti knjižnico `cstdlib`
- generator naključnih števil
  - je funkcija, ki vrne na videz naključno število, v bistvu pa se rezultat določi deterministično.
- s semenom generatorja
  - določimo na katerem mestu v zaporedju vračanih števil funkcija prične vračati vrednosti.
- seme generatorja naključnih števil določimo s funkcijo `void srand(int seme)`
- seme običajno določimo s sistemsko uro
  - `srand(time(0));`
  - za funkcijo `time()` moramo vključiti knjižnico `ctime`

Naključna števila (3)

35

- naključno število dobimo s klicem funkcije

  `int rand();`
- funkcija `rand()`
  - vrne naključno celo število med `0` in `RAND_MAX`
  - `RAND_MAX` konstanta definirana v `cstdlib` (2^{31}-1)
- naključno celo število med `0` in `a`, $a \in \mathbf{N}$
  - `int y = rand ()%(a+1);`
    - (a+1) – koliko števil je na izbiro

Naključna števila (4)

36

- naključno celo število med `a` in `b`, $a \leq b$, $a,b \in \mathbf{Z}$
  - `int y = rand ()%(b-a+1) + a;`
    - (b-a+1) – koliko števil je na izbiro, če štejemo a in b
    - a je najmanjše število, ki ga želimo
- naključno realno število med `0` in `1`
  - `double y = rand ()/double(RAND_MAX);`
- naključno realno število med `a` in `b`, $a \leq b$, $a,b \in \mathbf{R}$
  - `double y = rand ()/double(RAND_MAX)*(b-a)+a;`

Polja – zbirke elementov istega tipa (1)

37

- polja ločimo na
  - statična
    - velikost polja je znana v naprej (konstanta)
  - dinamična
    - velikost je lahko spremenljivka
    - sami rezerviramo in sprostimo pomnilnik
    - uporabimo kazalec

statična polja	dinamična polja

tip ime[velikost];	tip *ime; // kazalec na polje, prostor za
	// elemente še ni rezerviran
	ime = new tip[velikost]; // rezerviraj pomn.

int polje[5];	int *polje = new int[5];

Polja – zbirke elementov istega tipa (2)

38

- delamo s posameznimi elementi v polju
- elementi so indeksirani od 0 naprej
- dostop do posameznega elementa
  - operator [ ]
  - indeks elementa
  - npr. `polje[0] = 10;`
- pri delu z dinamičnim poljem moramo sami sprostiti rezerviran pomnilnik
  - `delete[] ime;`

Primer 3 - polja

39

```
/* program prebere polje n celih
   števil, kjer n preberemo preko
   tipkovnice, nato preveri, ali so vsa
   števila v polju deljiva s 3. */

#include <iostream>
using namespace std;

void preberi (int*aP, int aN){
   for (int i = 0; i < aN; i++){
         cout << "Vnesi element: ";
         cin >> aP[i];
   }
}

bool preverilastnost(int*aP, int aN){
   // iskali bomo takega,
   // ki ne ustreza
   for (int i = 0; i < aN, i++){
         if (a[i] %3 != 0)
         return false;
   }
   return true;
}
```

```
int main(){
   int n;
   cout << "Vnesi stevilo elementov: ";
   cin >> n;
   int *polje = new int [n];
   preberi (polje, n);
   if (preverilastnost(polje,n) == true)
     cout << "Vsa so deljiva s 3.";
   else
     cout << "Niso vsa deljiva s 3." ;
   delete [] polje;
   return 0;
}
```

Nizi – delo s tekstom

40

C-nizi
- izvirajo iz jezika C
- besedilo predstavimo s poljem znakov
- zadnji znak mora biti `'\0'`
- tip
  - char[]
  - char*

razred `string`
- razred definiran v knjižnici `string`
- imenski prostor `std`
- tip
  - string

razred `string` (1)

41

- deklaracija in inicializacija spremenljivke
  - `string niz1;` // prazen niz – 0 znakov
  - `string niz2("Test");` // kopija niza ali c-niza v oklepajih
  - `string niz3(10, '*');` // 10x ponovi znak '*'
- operator +
  - lepljenje nizov
  - `string niz4 = niz2 + niz3;` // niz4 = "Test**********"
- operator +=
  - dodajanje na konec
  - `niz2 += " in test";` // niz2 = "Test in test"

razred `string` (2)

42

- operator <<
  - definiran za uporabo na izhodnem podatkovnem toku
  - `cout << niz2;` // izpiše na ekran
- operator >>
  - definiran za uporabo na vhodnem podatkovnem toku
  - pridobi vse znake do prvega nevidnega znaka ali nove vrstice
  - `cin >> niz1;`

razred string (3)

43

- branje z podatkovnega toka
  - funkcija `getline()`
    - definirana v knjižnici `string`
    - je globalna funkcija, ne razredna
    - bere tudi nevidne znake
- dve različici uporabe funkcije getline()
  - `getline(vhodniPodatkovniTok, niz);`
  - `getline(vhodniPodatkovniTok, niz, ločilniZnak);`

razred string (4)

44

- razredne funkcije
  - kličemo jih drugače, kot "klasične" funkcije
  - najprej povemo objekt, nad katerim kličemo funkcijo
  - preko operatorja . (pika) dostopamo do razredne funkcije
- funkcija size()
  - vrne dolžino niza – število znakov v nizu
  - `int x = niz3.size();`
- funkcija clear()
  - odstrani vse znake v nizu
  - `niz2.clear();`

razred string (5)

45

- funkcija find(iskaniNiz)
  - išče podniz v danem nizu
  - `int x = niz3.find("ABC");`
  - vrne indeks, kjer se iskani niz prvič pojavi v nizu ali `string::npos`, če iskanega niza ne najde
- funkcija insert(i, podniz)
  - v niz na indeks *i* vrine podani *podniz*
  - spreminja obstoječi niz
  - paziti moramo, da je indeks *i* ustrezen
  - `niz2.insert(0, "ABC");`

razred string (6)

46

- funkcija erase(i, n)
  - iz danega niza izbriše znake
  - indeks *i* je indeks prvega znaka, ki ga izbriše
  - število *n* je skupno število znakov, ki jih brišemo
  - paziti moramo, da je indeks *i* ustrezen
  - `niz3.erase(0, 10);`
- funkcija substr(i, n)
  - vrne kopijo dela niza
  - indeks *i* je indeks prvega znaka, ki ga kopira
  - število *n* je skupno število znakov, ki jih kopiramo
  - paziti moramo, da je indeks *i* ustrezen
  - `niz2.insert(0, "ABC");`

Primer 4 - nizi

47

```
/* Program prebere niz znakov in en
   sam znak. Obstoječi niz spremeni
   tako, da izbriše vse pojavitve
   prebranega znaka. */

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

void brisiZnake(string &aN, char aZ){
   string::size_type i;
   do {
     i = aN.find(aZ);
     if (i!=string::npos)
       aN.erase(i,1);
   } while(i!=string::npos);
}
```

```
int main (){
   string niz;
   cout << "Vnesi niz: ";
   getline(cin, niz);
   char znak;
   cout << "Vnesi znak: ";
   cin >> znak;
   brisiZnake(niz, znak);
   cout << niz;
   return 0;
}
```