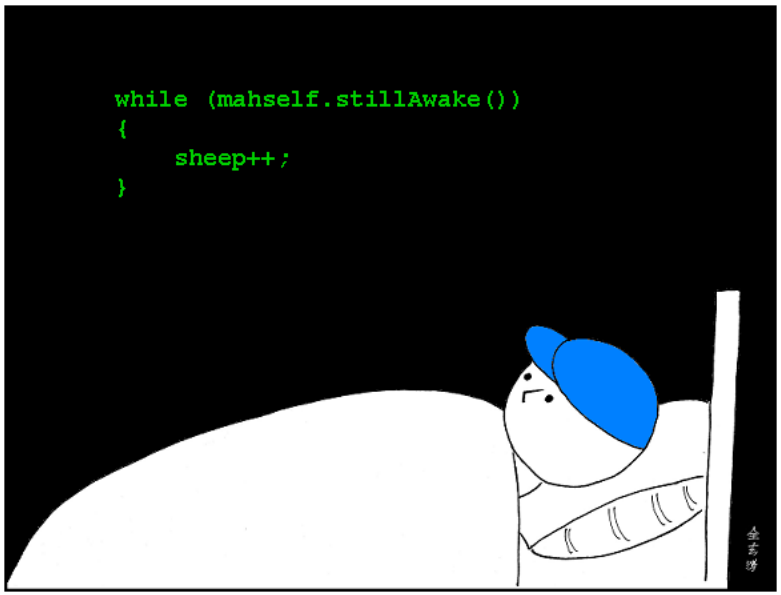
Programovanie, 1. prednáška

Peter Vanya
Jarná škola FX

21. apríla 2014

Veda & Programovanie

- Programovanie je po teórii a experimentoch „tretí spôsob“, ako spoznávať svet
- Vedu si dnes nemožno predstaviť bez počítačov, a väčšinu vedcov bez programátorských schopností



Čo frčí vo vede... a čo vás neminie

Vo vede sa možno stretnúť s nasledovnými jazykmi:

- Fortran/FORTRAN – najstarší (stále sa používa verzia z roku 77), najrýchlejší a najpoužívanejší (je v ňom napísane obrovské množstvo knižníc a programov)
- C/C++ – starý, rýchly, (C++) objektovo orientovaný a obsahujúci množstvo predprogramovaných algoritmov
- Python – moderný, príjemný a vhodný na úplne všetko od vedy až po webstránky. Na Linuxe ľahko dostupný: `$ python`
- MATLAB/Octave – od slova MAtrix LABoratory, ideálny na vedecké výpočty. GNU Octave je „MATLAB zadarmo“.

  ```
  $ sudo apt-get install octave
  ```

- Mathematica – symbolický, user-friendly a... drahý.

Poďme na to!

V tomto kurze sa naučíte C++, viac-menej jeho C-čkovú podmnožinu. Budem dúfať, že

- tak budete schopní ľahko sa naučiť ostatné jazyky,
- bude to dobrý základ pre potenciálne učenie sa objektov.

Užitočná stránka: `www.cplusplus.com/reference`

Ako sa naučiť dobre programovať? Recept je jednoduchý:

- *Practice makes perfect.* (IB English teacher)
- *The only difference between good and perfect is practice.* (Vladimir Horowitz tuším)
- O koncepte 10000 hodín ste už určite počuli. (Ak nie, prečítajte *Outliers* od Malcolma Gladwella.)

Hello world

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

int main(){
  cout << "Hello World!" << endl;
  return 0;
}
```

Vysvetlenie:

- `#include<iostream>` je knižnica, ktorá pre vstup a výstup, napr. `cout` a `cin`.
- `namespace std` znamená, že možno písať `cout` miesto `std::cout`, čo je otravné (neskôr ale rozumné).
- `int main()` a `return 0` ignorujte, proste to tam žiaľbohu je.

Premenné

- **int** – celé čísla
- **const int** – konštantné celé čísla
- **unsigned int** – celé pozitívne čísla, občas sa zíde
- **double** – 8 bajtová premenná, od 10^{-308} do 10^{308}
- pole: `int a[5]={1,2,3,4,5}` alebo `double a[N]`, kde N musí byť známe už pred kompiláciou, i.e. konštantné. **Pozor!** Číslovanie je od 0 do $N-1$. Ale kompilátor vám nepovie, ak sa náhodou v kóde rozhodnete zmeniť/vypísať `a[N]`. Zistíte to, až keď sa vám zasekne program (seg fault).
- **bool** – hodnota je `true` alebo `false`
- **char** – jedno písmeno: `char a="a"`
- **string** – viacero písmen: `string a[10]`, treba na to `#include<string>`

Pointre zatiaľ necháme tak.

Misc potrebné veci

Celý hello world program možno napchať na jeden riadok, v C++ je to technicky jedno:

```
\int main(){cout << "Hello World!" << endl; return 0;}
```

... ale pre čitateľa vášho kódu to jedno nie je, takže pekne odsádzajte a komentujte.

- `a = 5;` – do premennej *a* je zapísaná hodnota 5.
- `a == 5;` – kontrolujeme, či *a* je rovné 5.
- `a != 5;` – kontrolujeme, či *a* nie je rovné 5.
- Prírastok: `i=i+1` možno zapísať ako `i++`.
- `i=i-1` je `i--`.

Misc potrebné veci

Logické operátory:

- `(a!=b)` – $a \neq b$.
- `(a==0 || b==0)` – logické *alebo*.
- `(a==0 && a==0)` – logické *zároveň*.

`a%4` – zvyšok po delení a štyrmi.

Podmienky

Jednoducho:

```
int a;
cout << "Zadajte cislo: ";
cin >> a;
if(a%3 == 0){
  cout << "Cislo je delitelne tromi" << endl;
}
else if(a%2 == 0){
  cout << "Cislo je parne" << endl;
}
```

Pozor! Častá chyba se tady dělá. Program

```
if(a%2 = 0){
  cout << "Cislo je parne" << endl;
}
```

triviálne vypíše *Cislo je parne* za každej okolnosti. (Overte si.)

Cykly

`For` cyklus sa skladá z troch častí: počiatočná a konečná hodnota a prírastok.

```
int main(){
  for(int i=0; i<10; i++){
    cout << i*i << endl;  // vypise druhe mocniny 0 az 9
  }
  return 0;
}
```

Občas sa viac zíde iný typ for cyklu – `While`.

```
int main(){
  double a = 1.0
  while(a>1e-10){
    a=a/2;
    cout << a << endl;
  }
  return 0;
}
```

Funkcie

Počítame objem gule.

```
const double pi = acos(-1.0); // vsimnite si tuto definiciu!
double r = 1.0, V;
V = 4.0*pi*r*r*r/3.0;
cout << "Objem gule o polomere " << r << " je " << V << endl;
```

Toto je trochu nepraktické. Napíšme si funkciu `volume`:

```
double volume(double r){    // double znaci typ premennej
                            // ktoru vracia funkcia
  double V;
  V = 4.0*pi*r*r*r/3.0;
  return V;
}
```

Funkcie

... prípadne

```
double volume(double r){
  return 4.0*pi*r*r*r/3.0;
}
```

Potom ju v kóde zavoláme ako

```
cout << "Objem gule o polomere " << r << " je "
     << volume(r) << endl; // riadky mozno odsadzat lubovolne
```

Pozor! Funkcia vždy vracia **jednu** hodnotu, pole teda nebude fungovať. (Neskôr si vysvetlíme, ako to obísť.)
Funkcia môže aj vracať *žiadnu* hodnotu, napríklad:

```
void print(int a){
  cout << a << endl;
}
```

Prototyp

Keď píšeme funkciu pod `main` funkciu, potrebujeme navrch pridať *prototyp*, aby bola viditeľná pre `main`.

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

double volume(double);  // prototyp

int main(){
  // kod
}

double volume(double r){
  double pi = acos(-1.0);
  return 4*pi*r*r*r/3;
}
```

Passing by value & reference

Keď napíšeme funkciu

```
double volume(double r){
  double pi = acos(-1.0);
  return 4.0*pi*r*r*r/3.0;
  r = 2.0;   // argument sa meni vnutri
}
```

tak v kóde sa nám hodnota *r* nezmení. To je *predávanie argumentu hodnotou* (passing by value).

```
double r = 1.0;
cout << volume(r) << "\t" << r << endl; // vypise 12.56 a 1.0
```

Pri zavolaní funkcie sa vytvorí v pamäti nové miesto, kde sa všetky premenné skopírujú, a po skončení funkcie sa toto miesto vymaže.

Passing by value & reference

Keď ale do funkcie pridáme ampersand &, hodnota *r* sa nám zmení:

```
double volume(double &r){
  double pi = acos(-1.0);
  return 4.0*pi*r*r*r/3.0;
  r = 2.0;
}

double r = 1.0;
cout << volume(r) << "\t" << r << endl; // vypise 12.56 a 2.0
```

To je *predávanie argumentu referenciou* (passing by reference). Funkcia priamo ukazuje na premennú v pamäti a mení ju.

Pozor! Pole je možné zmeniť vo funkcii, aj keď je predané hodnotou. Toto často spôsobuje nevyžiadané chyby.

Rekurzívne funkcie

```
int factorial(int n){
  if(n==1)
    return 1;
  else
    return n*factorial(n-1);
}
```

Matematické funkcie

```
#include<cmath>
```

Táto knižnica obsahuje funkcie `sin`, `cos`, `exp`, `log` (ln), `sqrt`, `abs` a mnoho iných základných vecí.

Vektory

V C++ existuje štruktúra `std::vector`, ktorá je sofistikovanejším poľom:

```
#include<vector>
using namespace std;
...
int N = 10;          // velkost nemusi byt konstantna
vector<int> a(N);
for(int i=0; i<a.size(); i++) // pozna vlastnu velkost
  a[i] = N-i;
cout << a.size() << endl;      // vypise velkost vektora
a.push_back(1);   // prida element s hodnotou 1 na koniec
```

Po pripísaní `#include<algorithm>` je možné vektor triediť:

```
sort(a.begin(), a.end());
```

Output/Input

„Tlačenie“ dát do súboru:

```
#include<fstream>
...
int a[5] = {1,2,3,4,5};
ofstream tlac("data.txt");
for(int i=0; i<5; i++){
  tlac << a[i] << endl;
}
```

Alternatívne môžeme dáta vypísať na obrazovku a použiť príkaz

```
$ ./mojprogram > data.txt
```

Načítanie z klávesnice:

```
int n;
cout << "Zadajte cislo: ";
cin >> n;     // pozor na sipky ukazujuce opacnym smerom
```

Input

Načítanie dát zo súboru:

```
#include<fstream>
...
int a[10];
ifstream tlac("data.txt");
for(int i=0; i<10; i++){
  tlac >> a[i];
}
```

Alternatívne alternatívne môžeme načítavať, dokým nenarazíme na koniec súboru:

```
ifstream tlac("data.txt");
while(!tlac.eof()){
  tlac >> a[i];
  i++;
}
```