Proudy v C++

Jmenné prostory

Prostory jmen

- při vkládání několika hlavičkových souborů může vzniknout kolize

zeleznice.h

```
const int x=10;
typedef struct
{
   ...
} Hradlo;
```

logika.h

```
const int x=5;
typedef struct
{
   ...
} Hradlo;
```

```
#include "zeleznice.h"
#include "logika.h"
void main(void)
{

  Hradlo h1,h2;
  int a = x+3;
}
```

Které Hradlo?

ze zeleznice.h
nebo z hradlo.h

- řešením je v C++ vnoření deklarace do prostoru jmen

zeleznice.h

```
namespace Zeleznice {
  const int x=10;
  typedef struct
  {
    ...
  } Hradlo;
}
```

logika.h

```
namespace Logika {
  const int x=5;
 typedef struct
  {
    ...
  } Hradlo;
}
```

- mimo deklarovaný prostor není jméno přístupné, musíme se odkázat na příslušný prostor jmen

```
Logika::Hradlo
```

```
#include "zeleznice.h"
#include "logika.h"

void main(void)
{
  Zeleznice::Hradlo h1;
  Logika::Hradlo h2;
  int a = Logika::x+3;
}
```

- abychom nemuseli psát stále odkaz na prostor jmen, lze dosáhnout přímé viditelnosti pomocí direktivy **`using`**

```
#include "zeleznice.h"
#include "logika.h"

using namespace Zeleznice;
void main(void)
{
  Hradlo h1;
  Logika::Hradlo h2;
  using namespace Logika;
  int a = x+3;
}
```

- deklarace jmenných prostorů lze vnořovat
- deklarace jsou otevřené, tj. deklarace lze přidávat
  - v prvním souboru: `namespace A { … }`
  - ve druhém souboru: `namespace A { … }`
- pro běžné objekty, např. `cout`, je definován jmenný prostor `std`
- jména lze zkracovat pomocí aliasů:
  - `namespace Zel=Zeleznice;`

Proudy (streamy)

Konzolový vstup a výstup pomocí streamů

- V jazyce C++ jsou definovány tři streamy (proudy) v knihovně `iostream`:
  - výstupní (na stdout) `cout`
  - vstupní (ze stdin) `cin`
  - chybový(na stderr) `cerr`
- k proudům se váží dva operátory <<, >>, jejichž význam je pro tento účel předefinován (tzv. **přetížení operátorů**)

- proudy `cout, cin, cerr` jsou objekty (tři proměnné objektového typu `ostream` a `istream`, již deklarované v knihovnách)
  - co jsou objekty poznáme později
  - nejde tedy o příkazy
  - funkcionalita je skryta v přetížení operátorů (tj. k přiřazení jiné funkce k operátorům << a >> vzhledem k `ostream` a `istream`
- nic nebrání nadále používat funkce `printf, scanf`
- často se plete << a >>

- příklad:

v C++ jsou hlavičkové soubory bez přípony

```
#include <iostream>
using namespace std;
int main(void)
{
  int pl,pz;
  cout << "Zadejte pocet lidi a zvirat: ";
  cin >> pl >> pz;
  cout << "Pocet lidi je " << pl <<
  ", pocet zvirat je " << pz << '.';
  return 0;
}
```

Mnemotechnická pomůcka

při výstupu data směřují **do** streamu

```
cout << "Ahoj";
```

při vstupu data směřují **ze** streamu do proměnné

```
cin >> pz;
```

Manipulátory

- slouží pro řízení vstupní a výstupní konverze
- jsou definovány v `iomanip`

```
// nový řádek
cout << "Ahoj" << endl;
// nastaví šestnáctkový výpis
cout << hex << x;
// vstup čísla v šestn. soustavě
cin >> hex >> a;
```

`dec`	dekadická konverze
`hex`	šestnáctková konv.
`oct`	osmičková konverze
`endl`	konec řádku + “flush”
`setw(int n)`	šířka položky *n* znaků
`setfill(int c)`	plnící znak *c*
`setprecision(int n)`	*n* desetinných míst
`showpos`	vypíše znaménko
`boolalpha`	vypíše true/false

- více v nápovědě

- nastavit např. výpis znaménka můžeme také použitím manipulátoru

```
setiosflags(ios_base::fmtflags
                    flag)
```

- kde příznakem je bitový součet konstant z prostoru jmen ios_base
- `setiosflags(ios_base::showpos)`

Vyzkoušejte…

```
#include <iostream>
#include <iomanip>

using namespace std;
int main(int argc, char **argv)
{
  int x;
  cout << "Ahoj, svete!" << endl;
  cout << hex << 65 << endl;
  cin >> oct >> x;
  cout << dec << x << endl;
  cout << showpos << x << endl;
  cout << setw(5) << setfill('0') << x << endl;
  cout << boolalpha << true;
  return 0;
}
```

Úloha 1

Napište program, který vypočítá nejmenší společný násobek dvou čísel. Vstup a výstup naprogramujte pomocí proudů `cin, cout.`

Otestujte správnost zadání vstupu pomocí funkce (metody) `fail()`:

```
cin >> a >> b;
if (cin.fail()==true)
{ cerr << ...;
  return -1;
}
```

Další proudy

- existují i jiné proudy:
  - např. pro soubor:
    - `fstream`
    - `ofstream`
    - `ifstream`
  - pro výstup do řetězce
    - `istringstream`
    - `ostringstream`
    - `stringstream`

Proudy a soubory

- otevření souboru
  - při deklaraci
    - jméno souboru je parametrem tzv. konstruktoru)
  - metodou `open`
    - parametrem je ještě mód otevření souboru

Příklad:

```
ofstream soubor("vystup.txt");
```

nebo

```
ofstream soubor;
soubor.open("vystup.txt", ios::out);
```

- zavření souboru - metodou `close()`

Módy otevření

`ios::in`	pro vstup
`ios::out`	pro výstup (implicitní pro výstupní soubory)
`ios::binary`	binární mód (nejsou interpretovány řídicí znaky)
`ios::ate`	pozice nastavena na konec
`ios::app`	pro přidávání na konec
`ios::trunc`	soubor bude přepsán (vyprázdněn)

- kombinace módů = ***bitový součet***

Testování stavu souboru

- všechny metody vracejí typ bool:

`is_open()` soubor úspěšně otevřen

`bad()` předchozí operace čtení nebo zápisu neúspěšná

`fail()` totéž co bad(), navíc pokud došlo k formátovací chybě

`eof()` pozice ve vstupním souboru je na konci souboru

`good()` vše je O.K.

`clear()` maže všechny příznaky

Čtení a zápis dat

- přetíženými operátory <<, >>
- metodami
  - `get(char &c),put(char c)`
  - `read(char *str, streamsize count )`
  - `write(const char *str, streamsize count )`

- další pomocné funkce, např:
  - `getline(char *s,int n)` pro čtení celého řádku
  - `gcount()` pro zjištění, kolik bytů bylo přečteno

Příklad

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
  ofstream soubor("vystup.txt");
  if (!soubor.is_open()) return -1;
  soubor << "Ahoj, světe" << endl;
  soubor.close();
  return 0;
}
```

Zkopírujeme textový soubor po řádcích

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
  char radek[81];
  ifstream vstup("vstup.txt“);
  ofstream vystup("vystup.txt");
  while (vstup.eof()==false)
  {
    vstup.getline(radek,81);
    vystup << radek << endl;
  }
  vstup.close();  vystup.close();
  return 0;
}
```

kopieradky.cpp

- funkce getline() přečte řádek do konce, ale znak konce řádku do řetězce nevloží
  - proto tisknu `vystup << radek << endl;`
- bude-li řádek v souboru delší než 80 znaků, funkce getline() jej nenačte celý
  - nastaví příznak, který můžeme testovat voláním funkce rdstate():
  - `if ((vstup.rdstate() & failbit)!=0) { ... }`
- lepší řešení:
  - využijeme typ `string`, který implementuje řetězec „neomezené“ délky
  - pak musíme použít globální funkci getline()

```cpp
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main()
{
  string radek;
  ifstream vstsoubor("vstup.txt“);
  ofstream vystsoubor("vystup.txt");
  while (vstsoubor.eof()==false)
  {
    getline(vstsoubor,radek);
    vystsoubor << radek << endl;
  }
  vstsoubor.close();  vystsoubor.close();
  return 0;
}
```

kopieradky2.cpp

Řešení má jeden háček...

1. Poslední řádek v souboru není ukončen znakem konce řádku
   - funkce getline() přečte poslední řádek
   - protože při čtení nenarazí na znak konce řádku, ale na konec souboru, nastaví se příznak konce souboru
   - vytiskne se řádek do souboru + endl
   - při následném testu podmínky ve while cyklus končí
   - ve výstupním souboru mám znak konce řádku navíc

Řešení má jeden háček...

2. Poslední řádek v souboru je ukončen znakem konce řádku

- funkce getline() přečte poslední řádek
- protože při čtení narazí na znak konce řádku, nenastaví se příznak konce souboru
- vytiskne se řádek do souboru + endl
- při následném testu podmínky ve while cyklus nekončí, protože eof vrací false
  - cyklus proběhne ještě jednou, getline() do řetězce nic nenačte (je prázdný); vytiskne se prázdný řetězec + endl
- ve výstupním souboru mám opět znak konce řádku navíc

Řešení ...

- tisknu řetězec a znak konce řádku zvlášť
- znak konce řádku vytisknu, není-li konec souboru

```
while(!vstup.eof())
{
  getline(vstup,radek);
  vystup << radek;
  if (!vstup.eof()) vystup << endl;
}
```

kopieradky3.cpp

Úloha 2

Napište program, který zkopíruje textový soubor (po znacích). Využijte streamy. Jména souborů zadávejte jako parametry hlavního programu, ošetřete správnost otevření souboru. Využijte např. metod get a put

Proudy a řetězce

- umožňují zápis do/čtení ze řetězce způsobem práce se streamy
  - srovnejte se `sprintf, sscanf`
  - řetězec je reprezentován typem `string` z knihovny `string`
    - `#include <string>`

Výstup do řetězce

- **`ostringstream`**
- konstruktory

```
ostringstream(openmode which = ios_base::out);
ostringstream( const string & str, openmode
  which = ios_base::out );
```

- metoda `str()`
  - vrací/nastavuje řetězec, se kterým proud pracuje

```
#include <string>
#include <sstream>
int main()
{ string retez;
  ostringstream oss(retez);
  oss << "Pisi do retezce";
  cout << retez;
  return 0;
}
```

Příklad 2:

```
#include <string>
#include <sstream>
int main()
{ string retez;
  ostringstream oss;
  oss << "Pisi do retezce";
  retez = oss.str();
  cout << retez;
  return 0;
}
```

Vstup ze řetězce

- **`istringstream`**
- konstruktory

```
istringstream(openmode which =
  ios_base::in);
ostringstream( const string & str, openmode
  which = ios_base::in );
```

- hodí ze pro parsing (analýzu) řetězce

Příklad

- máme textový soubor obsahující na každém řádku město (jednoslovný název) a číslo (počet chodníků)
- úkolem je napsat program, který přečte soubor po řádcích, vynechá prázdné řádky a do řetězce město uloží název, do proměnné x typu int počet chodníků a vytiskne text na obrazovku, počet chodníků zdvojnásobí

Příklad

```
while(!vstup.eof())
{
  getline(vstup,radek);
  if(radek.length()> 0)
  {
    istringstream analyz(radek);
    analyz >> mesto; // cte do bileho znaku
    analyz >> pocet;
    pocet *=2;
    cout << mesto << '\t' << pocet << endl;
  }
}
```

soubor.cpp

Poznámka

- funkce `getline` může mít další parametr
  - ukončující znak (delimiter); pokud na něj při čtení funkce narazí, vyzvedne ho, neuloží do řetězce a zastaví čtení; další volání funkce čte data za tímto znakem

Příklad

- máme 3 údaje na řádku v textovém souboru (jméno, příjmení, rodné číslo) oddělené tabulátorem (čárkou, středníkem, ...)
  - načteme řádek do stringstreamu
  - jednotlivé údaje získáváme pomocí
  - `getline(stream,data,'\t')`

Příklad

```
string radek, jmeno, prijmeni, RC;
istringstream analyz;

while(!vstup.eof())
{
  getline(vstup,radek);
  analyz.str(radek);
  analyz.clear();
  getline(analyz,jmeno,'\t');
  getline(analyz,prijmeni,'\t');
  getline(analyz,RC);
  cout << jmeno << ';' << prijmeni << ';' << endl;
}
```

soubor2.cpp

Příklad jinak

```
string radek, jmeno, prijmeni, RC;
istringstream analyz;
string *data[3] = {&jmeno, &prijmeni, &RC}
while(!vstup.eof())
{
  getline(vstup,radek);
  analyz.str(radek);
  analyz.clear();
  for(i=0;i<3;i++)  getline(analyz,*data[i],'\t');
  cout << jmeno << ';' << prijmeni << ';' << endl;
}
```

soubor3.cpp

Poznámka

- volání metody `clear`

```
analyz.clear();
```

zajistí vymazání příznaků uvnitř proudu

- pokud bychom metodu nezavolali, proud by si "pamatoval", že došel na konec "souboru" (řetězce) a ve druhém běhu cyklu by nic nenačetl
  - přiřazení nového řetězce příznaky automaticky nenuluje