Builder.cz - http://www.builder.cz
Autor: Radim Dostál
Rubrika: C/C++
Datum vydání: 02.05. 2001
Url: http://www.builder.cz/art/cpp/cpp_neformat_io.html

Neformátovaný vstup a výstup v C++

Neformátovaný vstup a výstup v C++

Ještě než se dostaneme k samotnému tématu článku, rád bych se vrátil ke článku minulému. Na konci jsem slíbil několik málo krátkých příkladů na formátovaný vstup a výstup. Jako první uvedu bez velkého komentáře program, který vše, co přečte ze vstupu, zapíše na výstup a do souboru:

```
#include <iostream>
#include <fstream>
int main(int argc, char *argv[])
{
   ofstream log("log.txt");
   int znak;
   znak = cin.get();
   while (!cin.eof() && cout && log)
   {
      log << (char)znak;
      cout << (char)znak;
      znak = cin.get();
   }
}
```

Zde používám metodu `get` . Metoda `get` bez parametrů přečte jeden znak.

Druhý program přečte obsah parametrem zadaného souboru a zapíše jeho obsah v šestnáctkové soustavě.

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

int main(int argc, char *argv[])
{
   if (argc != 3)
   {
```

```
    cerr << "Špatné parametry" << endl;
    return -1;
  }
  ifstream in(argv[1]);
  ofstream out(argv[2]);
  out.setf(ios::hex | ios::uppercase);
  char znak;
  int pocet = 0;
  in >> znak;
  while (in && out)
  {
    if (pocet++ % 16 == 0)
    {
      out << endl << setfill('0') << setw(8) << (pocet-1) << ": " ;
    }
    out << (int)znak << " ";
    in >> znak;
  }
  out << endl;
  return 0;
}
```

Tolik tedy k formátovanému vstupu a výstupu. Vše lze výhodně použít při práci s textovými soubory, nebo při tvorbě filtrů v Unixových OS. Stejně tak lze tyto možnosti použít při tvorbě CGI programů. K formátovanému vstupu a výstupu se ještě vrátíme u paměťových proudů. Nyní se ale konečně podívejme na neformátovaný vstup a výstup.

Neformátovaný vstup a výstup se používá hlavně při práci s binárními soubory. Vstup se provádí metodou `read` . Její první parametr je `char*` a značí ukazatel na blok paměti, do kterého bude zapsán přečtený obsah souboru. Druhý parametr je `int` . Udává maximální počet přečtených bytů. Dojde-li k chybě (I v tomto případě je chybou také konec souboru.), můžeme skutečný počet přečtených bytů zjistit metodou `int gcount()`. K zapsání bloku dat slouží metoda `write` . Tato metoda má stejné parametry jako metoda `read` . Samozřejmě rozdíl je v tom, že metoda `write` blok paměti zapíše, ne přečte. Vše ukážu na následujícím příkladě. Vytvoříme něco jako příkaz copy, který nakonec vypíše, kolik bytů zkopíroval.

```
#include <iostream>
#include <fstream>
int main(int argc, char *argv[])
{
  if (argc != 3)
  {
    cerr << "Syntaxe: " << argv[0] << " zdroj cil" << endl;
    return -1;
  }
  ifstream in(argv[1],ios::binary);
  ofstream out(argv[2],ios::binary);
```

```
    char buffer[1000];
    int pocet = 0;
    while (in && out)
    {
      in.read(buffer,1000);
      out.write(buffer,in.gcount());
      pocet++;
    }
    --pocet *= 1000;
    pocet += in.gcount();
    cout << "Počet zkopírovaných bytů: " << pocet << endl;
    return 0;
}
```

Chtěl bych upozornit na konstruktory tříd `ifstream` a `ofstream` . Jako jejich druhý parametr je tak zvaný režim souboru. Mód `ios::binary` znamená, že soubor je otevřen v binárním módu. Existuje mnoho souborových módů. Například: `ios::app` - pokud soubor existoval, bude se připisovat na jeho konec, `ios::ate` - nastaví "ukazatel" souboru (Spíš bych měl říci pozici, nebo zapisovací hlavu.) na konec atd... Také bych chtěl upozornit na druhý parametr metody `write` . Zapíšu jen tolik, kolik jsem přečetl. Jinak soubory čtu i zapisuji po 1000 bytových blocích.

Tolik tedy k neformátovanému vstupu a výstupu. Jeho využití je hlavně při práci s binárními soubory. Příště se podíváme na paměťové proudy, které umožní formátovat řetězec v operační paměti.