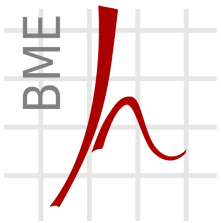
C++ I/O

Bevezetés



Híradástechnikai Tanszék

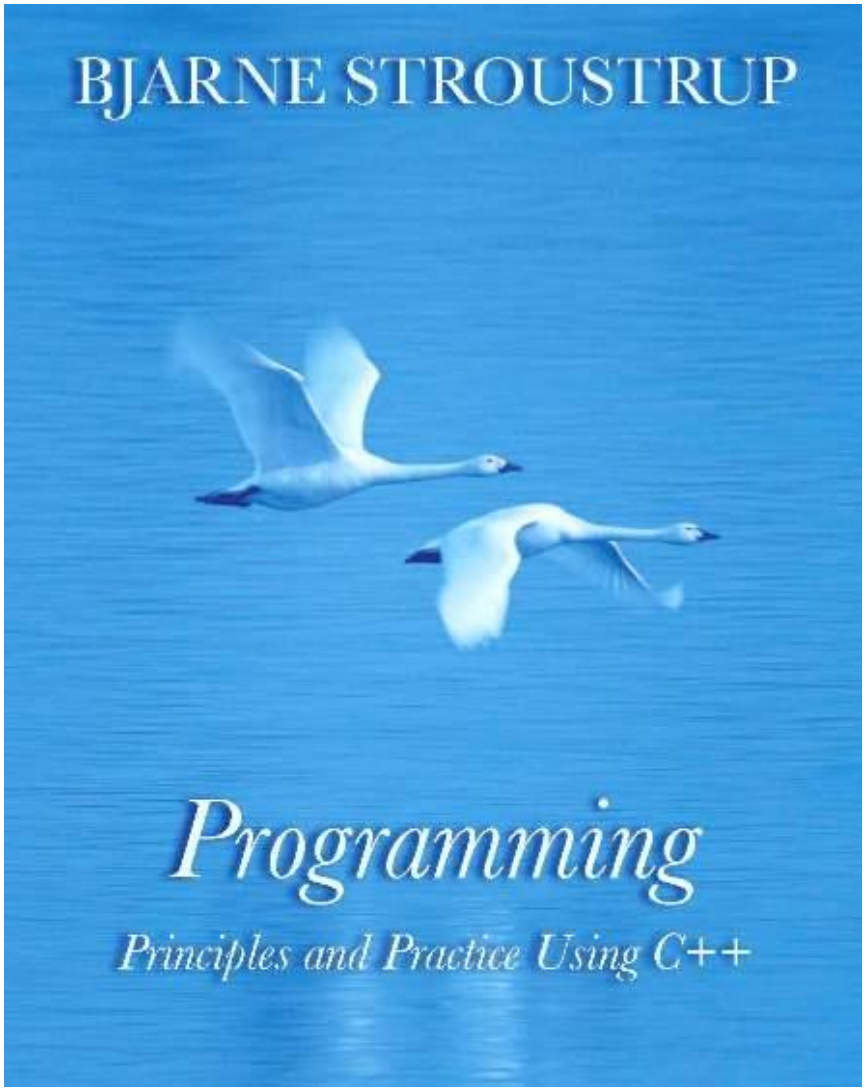
Izsó Tamás

2014. február 20.

Section 1

Bevezetés

Irodalom


BJARNE STROUSTRUP
Programming
Principles and Practice Using C++

Paraméter illesztés függvénynév túlterhelés esetén

1. Pontos típus illeszkedés. Nincs szükség konverzióra.
2. Egész kiterjesztés : bool –> int, char –> int, short –> int, unsigned –> int, float –> double
3. Szabványos konverziók: double <–> int, T* –> void*, Derived* –> Base*, int –> unsigned int
4. Felhasználó által írt konverziók.
5. Változó függvény argumentum : ...

Mit ír ki C-ben és mit C++-ban?

```
char ch;
printf("%d␣%d␣%d",sizeof(char),sizeof(ch),sizeof('x'));
```

Motiváció

```
#include<stdio.h>
void foo()
 {
   int i=19,
   double pi=3.14;
   printf("%d\n", i);   // OK
   printf("%d␣%d\n", i, 19, pi); // hibás formátum mező,
                                 // vagy paraméter
 }
```

Függvénynév túlterhelés

```
FILE* write(FILE* f, const int i)
{
    fprintf(f,"%d", i);
    return f;
}

FILE* write(FILE* f, const double r)
{
    fprintf(f,"%f", r);
    return f;
 }

FILE* write(FILE* f, const char *s)
{
    fprintf(f,"%s", s);
    return f;
}
```

Függvénynév túlterhelés

```
void foo()
{
    int i=4;
    double pi=3.14;

    write(stdout,"4*Pi=");
    write(stdout, 4*pi );
    write(stdout,"\n" );
 }
```

Függvények láncolása

```
void foo()
{
    int i=4;
    double pi=3.14;

    write(write( write(stdout,"4*Pi="), 4*pi) "\n" );
}
```

Függvények láncolása infix operátor átdefiniálással

```
FILE* operator<<(FILE* f, const int i)
{
    fprintf(f,"%d", i);
    return f;
}

FILE* operator<<(FILE* f,const double r)
{
    fprintf(f,"%f", r);
    return f;
 }

FILE* operator<<(FILE* f, const char *s)
{
    fprintf(f,"%s", s);
    return f;
}
```

Operátorok láncolása

```
void foo()
{
    int i=4;
    double pi=3.14;

    operator<<(operator<<(operator<<(stdout,"4*Pi="), 4*pi),
                          "\n" );
 }
```

Operátoros jelöléssel:

```
void foo()
{
    int i=4;
    double pi=3.14;

    stdout << "4*Pi=" <<  4*pi << "\n";
 }
```

<< és >> operátorok jelentése iostream esetén

- << Inserter (beszúrás) a kiírandó folyamba beszúrja a megjelenítendő adatokat.
- >> Extractor (kinyerés) a bejövő adatfolyamból kinyeri a soron következő adatot.

Előre definiált stream-ek.

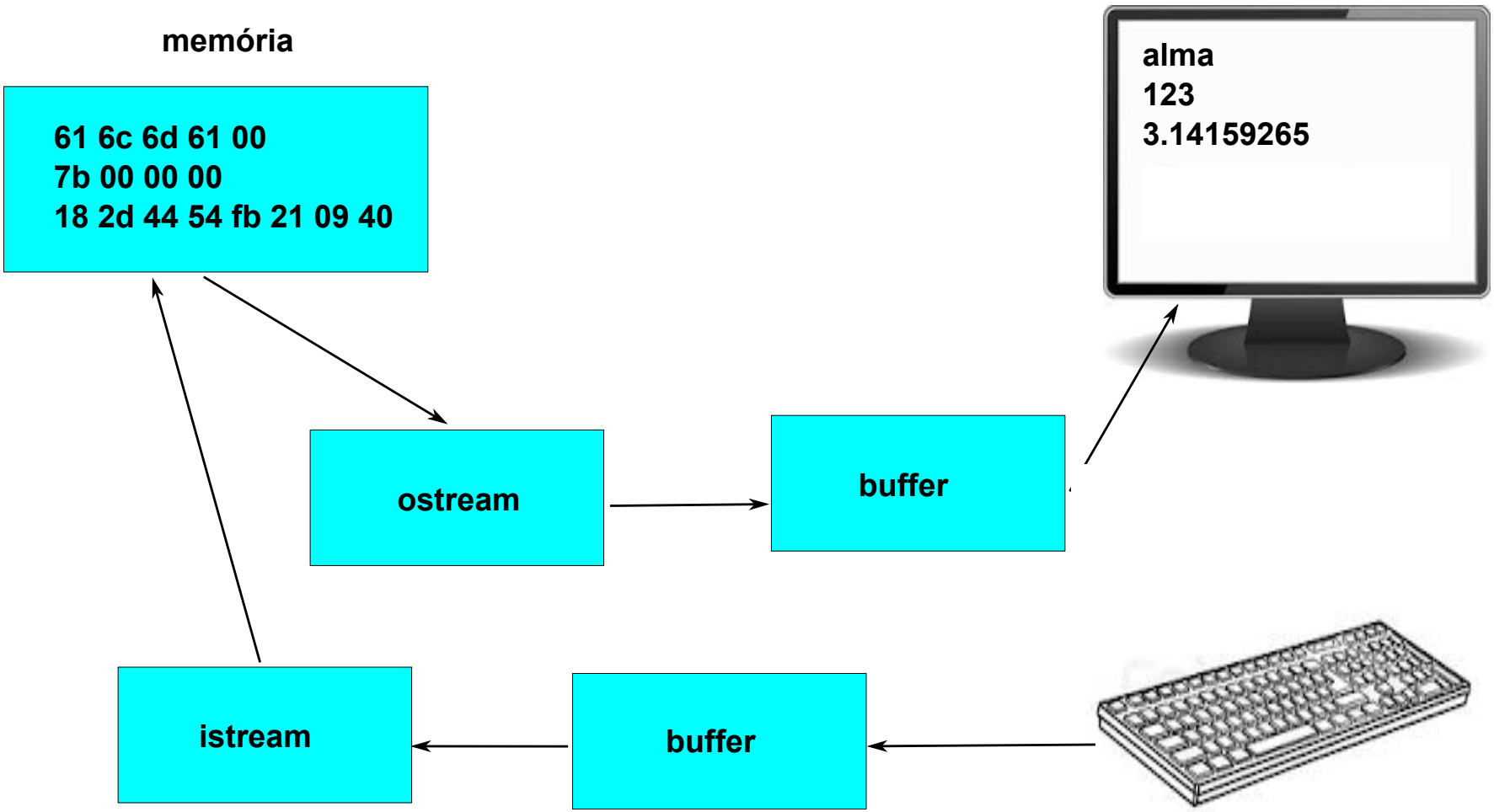
`cin` standard input (C megfelelője stdin)

`cout` standard output (C megfelelője stdout)

`cerr` standard error (C megfelelője stderr)

`clog` buffer-olt változata a standard errornak.

Stream modell



Section 2

Kiíratás

Felhasználó által definiált inserter

```
struct Komplex{ double r; double i; };

ostream& operator<<(ostream& o, const Komplex& p) {
    o << p.r << std::showpos
      << p.i << std::noshowpos << 'i';
    return o;
}

void foo() {
    Komplex p1={2,5};
    cout << p1 << endl; // 2+5i
}
```

Azonnali output megjelenítése

```
cout << "Kérem az x értékét \n" << flush;
```

Ezzel ekvivalens:

```
cout << "Kérem az x értékét" << endl;
```

Bináris kiíratás

```
int c='A';
cout.put(c);       // egy karaktert ír ki
cout << (char)c; // egy karaktert ír ki

// egész bináris kiíratása
cout.write( (char*)&c, sizeof(c) );
```

Tetszőleges adatok konvertálása karakterfüzérré

```
#include <sstream>
#include <iostream>
using namespace std;
int main() {
    ostringstream os;
    os << 123 << '␣' << 456 << ends;
    cout << os.str() << endl;
    return 0;
}
```

Keep it simple:
as simple as possible,
but no simpler."

Albert Einstein

Section 3

Beolvasás

Adatfolyam állapot lekezelése

```
int i;

if( cin >> i ) { .... } // rendben volt a beolvasás
else { .... } // hibás input vagy file vége
```

- bool good() A beolvasás sikeres volt.
- bool eof() Fájl vége volt.
- bool fail () Az utolsó beolvasás sikertelen volt, de adatvesztés nem lépett fel.
- bool bad() Javíthatatlan hiba. Ilyenkor a fail () függvény is jelez.

Beolvasó ciklus szervezése

```
const int MAXPAIR=10;
Pair vp[MAXPAIR];
int x,y;
int npair=0;
while( npair< MAXPAIR && cin>>x && cin >>y )
{
    vp[npair].x=x;
    vp[npair++].y=y;
}
```

Felhasználó által definiált extractor

```
istream& operator>>(istream& i, Pair& pair)
{
    i >> pair.x >> pair.y;
    return i;
}
```

Saját hibaállapot állítás

```
istream& operator>>(istream& i, Pair& pair)
{
    i >> pair.x >> pair.y;
    if( !i ) return i; // hiba volt
    if( pair.x <0 || pair.y<0 )
    {
        i.clear(ios::badbit|i.rdstate() );
    }
    return i;
}
```

Előírt érték beolvasása

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

int main()
{
   cout << "Kerek 1-tol 10-ig egy egesz szamot:\n";
   int n = 0;
   while (cin>>n) { // read
      if   (1<=n && n<=10) break; // check range
      cout << "Bocs, de "  << n
           << " nem [1:10] kozott van; probald meg ujra \n";
   }
   cout << "Beolvasott ertek: " << n <<endl;
   return 0;
}
```

Bolondbiztos beolvasás

```
#include <iostream>
using namespace std;
int main()
{
   cout << "Kerek 1-tol 10-ig egy egesz szamot:\n";
   int n = 0;
   while (n==0) {
      cin>>n;
      if( cin ) {
         if(1<=n && n<=10) break; // check range
         cout << "Bocs, de "  << n << " nem [1:10] kozott van; probald meg ujra \n";
         n=0;
      } else if( cin.fail()) { // nem egész szám jött
            cin.clear(); // hibaállapot törlése
            cout << "Bocs de ez nem szam volt; probald meg ujra \n";
            char ch;
            // nem számjegyek átolvasása
            while (cin>>ch && !isdigit(ch)) ;
            if (!cin) {  // nem találtunk számjegyet
               cerr <<" feladtam"; return 5;
            }
            cin.unget(); // számjegy visszatétele
        } else { // eof vagy rossz input
            cerr <<" feladtam"; return 5;
      };
   }
   cout << "Beolvasott ertek: " << n <<endl;
   return 0;
}
```

Mi rossz az előző programban?

- Összekeveredtek a feladatok, úgymint
  - érték olvasása
  - tevékenység kiírása
  - hiba kiírás
  - rossz input átolvasása
  - input tesztelése
- Szét kell választani a különböző logikai részeket.

Bolondbiztos beolvasás újra

Logikailag különböző részek:

- `int get_int(int low, int high);`
- `int get_int();`
- `void skip_to_int();`

Hibás adatok átolvasása

```
void skip_to_int()
{
    if( cin.fail()) { // nem egész szám jött
        cin.clear(); // hibaállapot törlése
        char ch;
        // nem számjegyek átolvasása
        while (cin>>ch ) {
            if( isdigit(ch)) {
                cin.unget();
                return;
            }
        };
    }
    throw "␣feladtam!"; // jobb híján string
}
```

Egész érték beolvasás

```
int get_int()
{
    int n = 0;
    while (true) {
        if( cin>>n ) return n;
        cout << "Bocs de ez nem szam volt. ";
        cout << "Probald meg ujra \n";
        skip_to_int();
    }
    return 0; // ide nem jut a vezérlés
}
```

Megadott intervallumba eső érték beolvasása

```
int get_int(int low, int high) {
  cout << "Kerek␣" << low "-tol␣" << high
       << "-ig␣egy␣egesz␣szamot:\n";
  while (true) {
    int n = get_int();
    if (low<=n && n<=high) return n;
    cout << "Bocs␣de␣a␣szam␣nem␣esett␣az␣intervallumba.";
    cout << "Probald␣meg␣ujra␣\n";
    skip_to_int();
  }
  return 0; // ide nem jut a vezérlés
}
```

Használat:

```
int n = get_int(1,10);
cout << "n:␣" << n << endl;

int m = get_int(2,300);
```

Section 4

Formázás

Maximálisan megengedett karakterszám beállítása

String `char*` beolvasása esetén vigyázni kell, nehogy több adatot olvassunk, mint amennyi memória rendelkezésre áll.

```
void foo()
{
    char p[5];
    cin.width( sizeof(p) );
    cin >> p;
}
```

másként:

```
void foo()
{
    char p[5];
    cin >> setw(sizeof(p) >> p;
}
```

Sor olvasás

```
void foo()
{
    char line[100];
    cin.getline( line, sizeof(line) );
}
```

Szóközök beolvasása

```
void foo()
{
    char c;
    cin >> noskipws;
    while( cin >> c && c !='\n' ) cout<<c;
    cin >> skipws;

    // a getc() minden karaktert beolvas
    cin.get(c);
    ....
    while( cin.get(c) ) { ..... }
}
```

Bináris input

```
void foo()
{
    int v[100];
    ....
    cin.read( (char*)v, sizeof(v) );
}
```

Karakterfüzér konvertálása tetszőleges adatra

```
#include <sstream>

void foo()  {
  int iv1, iv2;
  istringstream is("123 456");
  is >> iv1 >> iv2; // konvertálás
}
```

Numerikus formátumok

- Integer érték
  - 1234 (decimal)
  - 2322 (octal)
  - 4d2 (hexadecimal)
- Lebegőpontos szám
  - 1234.57 (general)
  - 1.2345678e+03 (scientific)
  - 1234.567890 (fixed)
- Lebegőpontos szám pontossága
  - 1234.57 (precision 6)
  - 1234.6 (precision 5)
- Mezőszélesség
  - |12| (alapértelmezés)
  - | 12| (12 4 karakter szélesen)

Mezőszélesség

```
cout.width(5);
cout << 12 << '␣' << 34 << '␣' << 56 << endl;
```

egyszerűbben:

```
cout << setw(5) << 12 << '␣' << 34 << '␣'
     << 56 << endl;
```

Kitöltés és jobbra, balra igazítás

```
cout.fill('*');
cout << left << setw(5) << 13 << ",";
cout << setw(5) << 25 << ",";
cout.fill('#');
cout << right << setw(5) << 14 << "," << endl;
```

output: `13***,25***,###14,`

Számrendszer megadása

```
int x=64;
cout << dec << x << ' '
     << hex << x << ' '
     << oct << x << ' ' << endl;
cout << showbase
     << dec << x << ' '
     << hex << x << ' '
     << oct << x << ' ' << endl;
```

output :

```
64 40 100
64 0x40 0100
```

Lebegőpontos szám pontossága

```
cout << 1234.56789 << '\t' << fixed << 1234.56789 << '\t'
     << scientific << 1234.56789 << '\n';
cout << general << setprecision(5)
     << 1234.56789 << '\t' << fixed << 1234.56789 << '\t'
     << scientific << 1234.56789 << '\n';
cout << general << setprecision(8)
     << 1234.56789 << '\t' << fixed << 1234.56789 << '\t'
     << scientific << 1234.56789 << '\n';
```

output :

```
1234.57 1234.567890 1.234568e+03
1234.6 1234.56789 1.23457e+03
1234.5679 1234.56789000 1.23456789e+03
```

Manipulátorok összefoglalása 1

név	jelentés
boolalpha	bool típus szövegesen (true, false)
dec	tízes számrendszer
endl	newline és flush
ends	null karakter
fixed	tizedes tört alak
flush	stream buffer kiürítés
hex	hexadecimális számrendszer
left	balra igazítás
noboolalpha	bool formátuma szám (0,1)
noshowbase	ne mutassa a számrendszer alakját
noshowpoint	ne írjon tizedesvesszőt (pontot)
noshowpos	ne írjon pozitív előjelet
noskipws	ne olvassa át a whitespace karaktereket
nouppercase	számoknál (E) és hexa értékeknél (A-F) ne írjon nagybetűt
oct	nyolcas számrendszer

Manipulátorok összefoglalása 2

név	jelentés
resetiosflags	formátum flagek törlése
right	jobbra igazít
scientific	exponenciális alak
setfill	kitöltő karakter beállítás
setiosflags	beállítja a megadott jelzőbiteket
setprecision	kiírandó tizedesjegyek beállítása
setw	mezőszélesség
showbase	mutassa a számrendszer alakját
showpoint	írjon tizedesvesszőt (pontot)
showpos	írjon pozitív előjelet is
skipws	whitespacek átolvasása
uppercase	számoknál (E) és hexa értékeknél (A-F) nagybetűt használjon

Section 5

Fájlkezelés

Fájlok megnyitása

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

int main ()
{
   fstream os("test.txt", fstream::in | fstream::out);
   if( os )
   {
      os << "PI:" << 3.14 << endl;
      .....
      os.close();
   }
   else cerr <"Hiba a fájl megnyitásakor" <<endl;
   return 0;
}
```

Megnyitási módok

flag	megnyitási mód
app (append)	minden egyes kiírás előtt a fájl végére pozícionál
ate (at end)	megnyitás után a fájl végére pozícionál
binary (binary)	bináris mód
in (input)	olvasás engedélyezése
out (output)	írás engedélyezése
trunc (truncate)	megnyitáskor a fájl tartalmának a törlése

Section 6

Gyakorló feladatok

Feladat 1.

Írja át az alábbi C programrészletet C++-ban szokásos iostream felhasználásával!

```
#include <stdio.h>
 int main() {
  #define LEN 20
  char str[LEN];
  int i = 10;
  scanf("%19s", str);
  printf("%s\n", str);
  fprintf(stdout, "Hello\n");
  fprintf(stderr, "Hello\n");
  printf("%d\n",i);
  printf("%04d\n",i);
  printf("%+d\n",i);
  printf("%x\n", i);
  printf("%X\n", i);
  printf("%02X\n",i);
  printf("%o\n",i);
}
```

Feladat 2. Mit ír ki a program?

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

int main () {
  fstream fs("alma.txt", fstream::out | fstream::trunc);

  int i = 16;
  fs.width(4); fs << hex << i;
  fs.width(8); fs << oct << i << '\n';
  double pi = 3.14159;
  fs.fill('.');
  fs.width(20); fs.precision(4);
  fs << pi << '\n';
  fs.close();
  fs.open("alma.txt", fstream::in);
  char buf[100];
  while (fs.getline(buf, sizeof(buf)))
     cout << buf << endl;

  fs.close();
  cin.ignore(1);
  return 0;
}
```