

- **Tricks of the Masters**

- Please send me:

- your favourite programming C/C++/Java trick
- your favourite one-line perl/sed/awk script
- anything else you want to share

Trick 1: I/O Stream Manipulators

Summary A technique using the machinery of I/O stream manipulators together with a *cleanup proxy* class to handle multiple-levels of indentation.

Due to Bruno Preiss

```
char c;
```

```
int i;
```

```
double d;
```

```
cout << c << i << d;
```

```
ostream& operator<< (ostream&, char);  
ostream& operator<< (ostream&, int);  
ostream& operator<< (ostream&, double);
```

```
inline ostream& operator<< (  
    ostream& s, ostream& (*pf)(ostream&) )  
{  
    return (*pf)(s);  
}
```

```
ostream& endl (ostream& s)  
{  
    s << "\n";  
    s.flush ();  
    return s;  
}
```

```
cout << endl;
```

```
class Indent
{
    static unsigned int level;
public:
    Indent ()
        { ++level; }
    ~Indent ()
        { --level; }

    friend ostream& indent (ostream&);
};

unsigned int Indent::level = 0;
```

```
ostream& indent (ostream& s)
{
    for (unsigned int i = 0; i < level; ++i)
        s << "    ";
    return s;
}
```

```
void Foo ()
{
    Indent some;
    s << indent << "stuff" << endl;
    {
        Indent more;
        s << indent << "more stuff" << endl;
        {
            Indent evenmore;
            s << indent << "even more stuff" << endl;
        }
        s << indent << "back at more level" << endl;
    }
    s << indent << "back at some level";
}
```

Trick 2: Override Default Constructors

Summary Overriding the default constructors for the variables of a class in the constructor of that class is efficient.

Due to Chris Trudeau

```
class Stuff
{
public:
    Stuff() : x(1) {}           // default assignment
    Stuff( int _x ) : x(_x) {} // parameterized assignment

private:
    int x;
};
```

```
class Foo
{
    int const x;
public:
    Foo () : x(1) {} // OK
    Foo (int _x) : x(_x) {} // OK
    Foo (double _x) { x = (int) _x; } // NOT OK. x :
};
```

Trick 3: Constructors in Structs

Summary Putting constructors in your structs make them more robust.

Due to Ian Cunningham

```
struct st_person
{
    char* mpc_name;
    char* mpc_address;
    int mi_age;

    st_person()
    {
        // good default values
        mpc_name = NULL;
        mpc_address = NULL;
        mi_age = -1;
    }

    ~st_person()
    {
        // calling delete on a NULL pointer is defined to have no effect
        delete mpc_name;
        delete mpc_address;
    }
};
```

Trick 4: Special Variable Names (Hungarian?)

Summary Encode the type of a variable/function in

Due to Ian Cunningham

```
int i_an_int;  
unsigned int ui_an_unsigned_int;  
char c_a_char;  
unsigned char uc_a_unsigned_char;  
short s_a_short;  
float fl_a_float;  
double d_a_double;  
int ai_an_array_of_int[5];  
struct st_a_struct {}
```

```
class cl_my_class
{
public:
    int mfi_value()
    {
        return mi_value;
    }

    int mi_value;
};
```

Trick 5: Special Variable Names

Summary Use uppercase to identify the category of

Use uppercase when stringing several words together.

Avoid abbreviations.

Due to Bruno Preiss

```
class Foo { // capitalize class/struct/enum (type)
{
    void Bar () {} // capitalize functions
    int x; // lowercase variables
};
```

```
int thisIsALongName;
int unvOWloo; // yuck
int universityOfWaterloo; // too long?
```

Trick 6: Template Abominations

Summary Use a template to calculate results at com

Due to Bill Bishop

```
template<int N>
class Fibonacci {
    public:
        enum {
            value = Fibonacci<N-1>::value
                + Fibonacci<N-2>::value
        };
};
```

```
class Fibonacci<1> {  
    public:  
        enum { value = 1 };  
};
```

```
class Fibonacci<0> {  
    public:  
        enum { value = 1 };  
};
```

```
int main( )
{
    cout << "Fibonacci<0> = " << Fibonacci<0>::value;
    cout << "Fibonacci<1> = " << Fibonacci<1>::value;
    cout << "Fibonacci<2> = " << Fibonacci<2>::value;
    cout << "Fibonacci<3> = " << Fibonacci<3>::value;
    cout << "Fibonacci<4> = " << Fibonacci<4>::value;
    cout << "Fibonacci<5> = " << Fibonacci<5>::value;

    return( 0 );
}
```

Trick 7: Viral Program

Summary Self-replicating C code.

Due to Carey Wan

```
char*f="char*f=%c%s%c,q='%c',n='%cn',b='%c%c';%cmai
{printf(f,q,f,q,q,b,b,b,n,n);}%c",q='"',n='\n',b='\
main(){printf(f,q,f,q,q,b,b,b,n,n);}
```

Trick 8: Smart Pointers (Part 1)

Summary Smart pointers ease resource management exceptions gracefully.

Due to Paul Goswami

```
void processData (istream& datasource)
{
    while (datasource)
    {
        T* ptr = readData (datasource);
        ptr->processSomething ();
        delete ptr;
    }
}
```

```
void processData (istream& datasource)
{
    while (datasource)
    {
        T* ptr = readData (datasource);
        try
        {
            ptr->processSomething();
        }
        catch (...)
        {
            delete ptr;
            throw;
        }
        delete ptr;
    }
}
```

```
template<class T>
class auto_ptr
{
    T* ptr;
public:
    auto_ptr(T *p = 0) : ptr(p)
        {};
    ~auto_ptr()
        { delete ptr; };
    // ...
    T* operator->() const;
    T& operator*() const;
}
```

```
void processData (istream& datasource)
{
    while (datasource)
    {
        auto_ptr<T> ptr (readData (datasource));
        ptr->processSomething ();
    }
}
```

Trick 9: Smart Pointers (Part 2)

Summary Smart pointers simplify the handling of exceptions and destructors.

Due to Paul Goswami

```
class Entry
{
    Item* const item1;
    Item* const item2;
public:
    Entry () : item1 (new Item ()), item2 (new Item

    ~Entry () { cleanup (); }

    void cleanup()
    {
        delete item1;
        delete item2;
    }
}
```

```
Entry::Entry() :
    item1 (initItem ()),
    item2 (initItem ())
    {}
```

```
Item* Entry::initItem()
{
    try
    {
        return new Item ();
    }
    catch(...)
    {
        cleanup();
        throw;
    }
}
```

```
class Entry
{
    auto_ptr<Item> item1;
    auto_ptr<Item> item2;
public:
    Entry () :
        item1 (new Item ()),
        item2 (new Item ())
    {}

    ~BookEntry()
    {}
}
```

Trick 10: Commas in integers

Summary Insert commas into the textual representation of an integer. (Perl script).

Due to Larry Wall and Randal L. Schwartz

```
sub commas
{
    local ($_) = @_;
    1 while s/(.*\d)(\d\d\d)/$1,$2/;
    $_;
}
```

```
print &commas (1234567);
```

```
1,234,567
```

Request for Trick 1: Vararg Templates

Summary What do you do when you need a variable arguments in a template declaration? In the past, I had multiple template declarations—each with a different arguments. However, this is not satisfactory as there is a limit on the number of arguments.

Due to Ajit Singh

Request for Trick 2: Changing Operator Precedence

Summary In certain operator overloading situations, to change the built-in precedence or associativity of operators. What is the best way out?

Due to Ajit Singh