UPPSALA UNIVERSITET

Methods of Programming DV2

Elegant code

What perfection?

Perfection is achieved, not when there is nothing more to add, but when there is nothing left to take away.

Antoine de Saint-Exupery

Everything should be made as simple as possible, but not simpler.

Albert Einstein

He adds function by deleting code.

Anonymous

What is elegance?

Simplicity.
Consistency.
Readbility.

Put readability ahead of ”clever” constructs.
Use consistent coding style. If working with existing code, use the existing style. (Coding standards…)
Use available language features, but don’t over-use.
What does `if (foo = bar == baz) {…}` mean?
Abstract away from specific cases. E.g.
If copying 10 array elements, use a loop – not 10 copy statements!
Put similar code in a procedure (or macro…)
Use named constants.

Some constructs in C

What about this common C idiom?

```
*to++ = *from++
```

Or even worse: (copy a zero-terminated character string)

```
while(*to++ = *from++) {}
```

Clever? Language overuse?
What if they were not common idioms?
What about `x++` vs. `++x`?
Conditional expressions:

```
x = y+(z!=0?w/z:99999999)
```


A new language construct

```
struct list {
  int hd;
  struct list *tl;
};
typedef struct list *listref;
#define list_hd(x) (x->hd)
#define list_tl(x) (x->tl)
```

An *iterator* for lists (not very well done, but illustrates the idea)

```
#define foreach(l,x)                                \
   listref ll;                                      \
   for(ll=l;                                        \
       ll!=NULL && (x=list_hd(ll),1);               \
       ll=list_tl(ll))
```

Add the elements of a list:

```
sum=0;
foreach(p,i) {sum=sum+i;};
```


Jensen’s device

(Jørn Jensen, Regnecentralen Copenhagen 1960)

```
#define Jensen(var,from,to,step,x,sum)          \
  {sum = 0;                                     \
   for (var=from; var<=to; var=var+step) {      \
     sum = sum+x;                               \
   };                                           \
  }
```

This is a generalised summing procedure. *Call by name* parameter passing is crucial. Examples:

Sum the integers from 1 to 10: `Jensen(i,1,10,1,i,sum)`
Sum the array `a` with elements 0 to 9:

```
Jensen(i,0,9,1,a[i],sum)
```

Approximate the integral of x^2 from 0 to 2:

```
Jensen(x,0,2,0.01,x*x*0.01,result)
```

Time critical loop

Copy to hardware i/o register (the loop overhead made it too slow):

```
do
  *to = *from++;
while(--count>0);
```

Optimised version (loop unrolling):

```
n = count/8;
switch(count%8){
  case 7: *to = *from++;
  case 6: *to = *from++;
  case 5: *to = *from++;
  case 4: *to = *from++;
  case 3: *to = *from++;
  case 2: *to = *from++;
  case 1: *to = *from++;
  case 0: ;
 };
do{*to = *from++;*to = *from++;*to = *from++;*to = *from++;
   *to = *from++;*to = *from++;*to = *from++;*to = *from++;
 } while(--n>0);
```


Duff's device

(Tom Duff, Lucasfilm 1983)

```
n=(count+7)/8;
switch(count%8){
case 0: do{     *to = *from++;
case 7:         *to = *from++;
case 6:         *to = *from++;
case 5:         *to = *from++;
case 4:         *to = *from++;
case 3:         *to = *from++;
case 2:         *to = *from++;
case 1:         *to = *from++;
        }while(--n>0);
 };
```

”Many people have said that the worst feature of C is that switches don't break automatically before each case label. This code forms some sort of argument in that debate, but I'm not sure whether it's for or against.”

Recall functional programming

```
load "Math"; open Math;
fun square x = x*x:real;
fun twice x = x*2.0;
square(twice(sqrt 9.0))
```
computes to `36.0`

Functional composition operator `o`:

```
infix o;
fun (f o g) x = f(g x);
```
(built-in function in ML)

```
(square o twice o sqrt) 9.0
```
computes to `36.0`

```
(square o twice o sqrt) ~9.0
```
raises exception!

Can this be handled more gracefully? How do we program a ”stop” in the middle of a function composition sequence?

```
(SOME o square o twice o sqrt) 9.0
```
computes to
```
  SOME 36.0;
```


Continuations

Make explicit what should be done with the computed function value:

```
fun csquare(c,x) = c(x*x:real);
fun ctwice(c,x) = c(x*2.0);
fun csqrt(c,x) = if x < 0.0
                 then NONE
                else
                 c(sqrt x);

infix %;
fun (f%g)x = g(f,x);
```

```
(SOME % csquare % ctwice % csqrt) 9.0
```
computes to
```
  SOME 36.0;
```

```
(SOME % csquare % ctwice % csqrt) ~9.0
```
computes to
```
  NONE;
```