Subject: Re: REDUCE
Posted by Richard Younger on Mon, 02 Apr 2001 03:31:47 GMT
View Forum Message <> Reply to Message

JD Smith wrote:
>
> Richard Younger wrote:
>>
>> John-David Smith wrote:
>>>
>>> Kenneth Mankoff wrote:
>>>>

[SNIP]

> generated code. Granted, this is a very simple example, but what I am
> looking for is a solution which makes use of the redundancy in this code
> to avoid generating most of it. I may be asking more out of compilers
> than they can offer.

I was thinking about this a little more, and then it hit me: You
probably don't want templates.

Templates are a means for providing ~compile time~ polymorphism. With
compile time polymorphism, you'll always need some sort of switch()
statement or a decision table to decide which compiled path to take at
runtime. What you're really looking for is ~run time~ polymorphism.

Run time polymorphism is provided for with function pointers in C and
virtual functions (and parent class / derived class hierarchies) in
C++. Now I suppose I could try to cook up a C++ example using
virtuality, but outside of a brief exercise or two, I've never actually
used it. So here's an example of a function pointer table. Tell me if
you like it. No C++ required.

WARNING -- large code block follows -- see bottom for more actual text.

```
/* myheaderfile.h ---------------------------------------*/

#include <stdio.h>
#include "export.h"

void NULL_process (char* out, IDL_VARIABLE* arg, int new_nel, int skip,
int atom, int n_cdim )
{
 /*IDL_MESSAGE(... "She canna handle this type, cap'n!" ... )*/
 return;
}

/* or use a macro to generate these in the header file */
void BYTE_process (char* out, IDL_VARIABLE* arg, int new_nel, int skip,
int atom, int n_cdim )
{
 int i, j, ind, base;

     UCHAR *tin,*tout,tmp;
     tout=( UCHAR *)out;
     tin=( UCHAR *)arg->value.arr->data;
     for(i=0,base=0; i<new_nel; base+=skip) {
       for(j=0;j<atom;j++) {
         tmp=tin[j+base];
         for(ind=j+base; ind<j+base+atom*n_cdim; ind+=atom) {
           if(tin[ind]>tmp)
      tmp=tin[ind];
         }
         tout[i++]=tmp;
       }
     }
 return;
}

void INT_process (char* out, IDL_VARIABLE* arg, int new_nel, int skip,
int atom, int n_cdim )
{
 int i, j, ind, base;
     short *tin,*tout,tmp;
     tout=( short *)out;
     tin =( short *)arg->value.arr->data ;
     for(i=0,base=0;i<new_nel;base+=skip) {
       for(j=0;j<atom;j++) {
         tmp=tin[j+base];
         for(ind=j+base;ind<j+base+atom*n_cdim;ind+=atom) {
           if(tin[ind]>tmp)tmp=tin[ind];
         }
         tout[i++]=tmp;
       }
     }
}

/* etc ... */
/*----end of header-------------------------------------------------*/

/*---start of main---------------*/
#include "myheaderfile.h"
```

```
/* typedef pointer to processing function */
typedef void (*PROC_FUNC) (char* , IDL_VARIABLE*, int, int, int, int);

void main (void)
{
 PROC_FUNC process_table[] = {
  NULL_process,
  BYTE_process,
  INT_process
  /* etc ...
   to fill out this table, you'll need to look at
   the macro IDL_TYP_ values in export.h and put in
   placeholders for those types that you can't handle.*/
 };

 /*RSI will scowl at this method, because the table relies on
   the values of the macro-defined type indices (e.g. #define
IDL_TYP_INT 2)
   staying the same, which they do not guarantee.
 */

 /*dummy arguments*/
 char* out = NULL;
 int new_nel=0, skip=0, atom=0, n_cdim=0;
 IDL_VARIABLE* arg[5];
 int type = 1;

 process_table[type](out, arg[0], new_nel, skip, atom, n_cdim );

 return;
}
/*-----------------------*/
```

... now I ~know~ this compiles, and will run the relevant functions (I tested with print statements), but I may have mangled your math. Double check before cut and pasting.

You could also stick the typedef and the table in the header, making it global, if that suited your sense of style better.

for completeness, here's a template function if you want to take a look at it. Notice it still requires the switch, because there must be an argument of the templated type.

```
#include <stdio.h>
```

```
#include "export.h"


template <class Item>
void process (Item tmp, IDL_VARIABLE* arg[], void* out, int new_nel, int
skip, int atom, int n_cdim )
{
 int i, j, ind, base;

 Item *tin, *tout;
    tout=( Item *)out;
     tin =( Item *)arg[0]->value.arr->data ;
     for(i=0,base=0; i<new_nel; base+=skip) {
       for(j=0; j<atom; j++) {
         tmp=tin[j+base];
         for(ind=j+base; ind<j+base+atom*n_cdim; ind+=atom) {
           if(tin[ind]>tmp)tmp=tin[ind];
         }
         tout[i++]=tmp;
       }
     }
  return;
}

void main(void)
{
 int type = 1;

 void* out = NULL;
 int new_nel=0, skip=0, atom=0, n_cdim=0;

 IDL_VARIABLE* arg[5];

 switch( type ) {
 case IDL_TYP_BYTE: {
  UCHAR tmp = 0;
  process (tmp, arg, out, new_nel, skip, atom, n_cdim );
  }
  break;
 case IDL_TYP_INT: {
  short tmp = 0;
  process (tmp, arg, out, new_nel, skip, atom, n_cdim );
  }
  break;
 //etc...
 }

 return;
```

}

```
> I.e. an implicit double sum over the second and third indices?  I
> wouldn't do this with for loops.  I would use rebin, and total, like:
>
> res=total(total(epsilon*rebin(reform(e_one,1,1,s[2]),s)*  $
>               rebin(1#e_two,s),1),2)
>
> Admittedly, your notation is somewhat cleaner.  If these are confusing,
> see my tutorial (to be posted) concerning rebin+reform in action.
>
> JD
```

I use reform commonly, but I'll have to side with David and say that the
use of REBIN and HISTOGRAM for anything but their most obvious uses is a
black art that intimidates me. *shudder* :-) Hey, wait a second! You
just want to put together this REDUCE package so you can put REBIN and
HISTOGRAM to more arcane uses than before, don't you?

My notation also allows almost direct copying of formulas from a large
number of textbooks, which is one major reason I'm partial to it.
Unfortunately, it's wishful thinking, and REBIN is not. :-)

Rich

--
Richard Younger
Assistant Technical Staff   MIT Lincoln Laboratory
Electro-Optical Materials and Devices   Mail Stop C-130
Email: younger@ll.mit.spmdecoy.edu   244 Wood St.
Phone: (781)981-4464   Lexington, MA 02144-9108

(My opinions on this forum are not endorsed, warrantied, etc. by my
employer. Heck, they probably own the copyright anyway, though.)