CS113: Lecture 12

Topics:

- C's dustier corners

goto

- The first rule of `goto` is, you do not use `goto`.

  To use: `goto identifier;` where `identifier:` is placed somewhere in program.

```
if( a == 3 ) goto three;
printf( "a is not 3.\n" );
exit( 0 );

three:
printf( "a is 3.\n" );
```

- When is using `goto` considered acceptable? To "jump out" of more than one layer of nesting. For example:

```
for( ... )
{
   for( ... )
   {
      if( disaster )
         goto error:
   }
}

error:
   /* clean up the mess */
```

- Note that use of `goto` is always avoidable, e.g. by using "flag" variables. However, occasionally it is convenient.

The ternary ? : operator

- A type of conditional expression. Form:

  `test ? expr1 : expr2`

  `test` is evaluated first. If it is non-zero ("true"), then `expr1` is evaluated, and the entire expression has value `expr1`. Otherwise, `expr2` is evaluated, and the entire expression has value `expr2`.

- Example. Instead of

```
if( a > b )
   z = a;
else
   z = b;
```

  We can write

```
z = ( a > b ) ? a : b;
```

- A little trick...

  Can something like the following be done (without duplicating `complex_expression`)?

```
((condition) ? a : b ) = complex_expression;
```

  Yes!

```
*((condition) ? &a : &b ) = complex_expression;
```

The comma operator

- Form: `expr1, expr2`.
- Most common use: in `for` loop.

```
void reverse( char *s )
{
   int temp, i, j, len;
   len = strlen( s );
   for( i = 0, j = len - 1; i < j; i++, j-- )
   {
      temp = s[i];
      s[i] = s[j];
      s[j] = temp;
   }
}
```

- Evaluated left-to-right. All side-effects resulting from evaluation of left expression are completed before right expression evaluated.
- Type and value of the result are the type and value of the right operand.
- Example:

```
int a = 3, b = 6, c;
c = (a++, (b++) + a);
printf( "a is %d, b is %d, c is %d.\n", a, b, c );
```

(Prints 4, 7, 10.)

Note on array notation

- Does the seemingly insane expression `5["0abcdefgh"]` make sense?

- Yes, it does! Array subscripting in C is "commutative", i.e., `a[e]` is identical to `*((a) + (e))` for any two expressions `a` and `e`.

- Thus, the following are all equal.

```
a[e]
*((a) + (e))
*((e) + (a))
e[a]
```

- ...and `5["0abcdefgh"]` is equal to `"0abcdefgh"[5]`, which is `'e'`.

Loop Unrolling

Which is faster?

```
for( i = 0; i < 8 * n; i++ )
{
   a[i] = i;
}

for( i = 0; i < n; i += 8 )
{
   a[i] = i;
   a[i+1] = i+1;
   a[i+2] = i+2;
   a[i+3] = i+3;
   a[i+4] = i+4;
   a[i+5] = i+5;
   a[i+6] = i+6;
   a[i+7] = i+7;
}
```

Tom Duff (while at Lucasfilm) wanted to copy chunks memory, *quickly*. Original code:

```
send(to, from, count)
register short *to, *from;
register count;
{
   do
      *to = *from++;
   while(--count>0);
}
```

Duff's Device

"Many people (even bwk?) 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."

– Tom Duff

```
send(to, from, count)
register short *to, *from;
register count;
{
   register 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);
   }
}
```