

---

# Computer Programming: Skills & Concepts (CP1)

## Boolean Expressions; Increment and Decrement

5th November 2009



## Revision, Tuesday 3rd November

- Discussed functions, pointers, arrays.
- `pointers.c`
  - Two functions `areTheyPrimes` and `evenDivComments` for discussing pointer/array issues;
  - (on board) Added an extra function `evenDivComments2` - similar to `evenDivComments`, but does *not* modify its input array - instead it returns a new array as its return type. (**some details wrong**)

## Today's lecture

- Quick pointing out of two bugs (one small, one a bit bigger) in the on-the-board code developed in Tuesday's lecture.
- Boolean *expressions* in detail;
- Expressions for incrementing;
- *Precedence* for evaluating expressions.

## evenDivComments

```
int evenDivComments(int a[], int n) {
    int j;
    printf("For curiosity, note that address of j is %p.\n", &j);
    for (j = 0; j < n; j++) {
        if (a[j] % 2 == 0)
            a[j] = a[j] / 2;
        printf("Address of %d-index of array is %p\n", j, (a+j));
    }
    return EXIT_SUCCESS;
}
```

## evenDivComments2

**note:** return type of evenDivComments2 is int\*.

```
int* evenDivComments2(int a[], int n) {
    int *w, j;
    w = calloc(n, sizeof(int));
    if (w != NULL) {
        for (j = 0; j < n; j++) {
            if (a[j] % 2 == 0)
                w[j] = a[j] / 2;
            else w[j] = a[j];
        }
    }
    return w;
}
.....
```

```
int* p;                /* general pointer variable (to int) */
evenDivComments(b, 10);
printf("\n");
printf("After evenDivComments, b is: ");
for(y = 1; y < 10; y++)
    printf("%d, ", b[y]);
printf("\n");
/* evenDivComments2 does return an array. We store it in the
 * previously unassigned int pointer p. Can NOT assign this
 * return array DIRECTLY to b, because array pointers are static
 * and cannot change. Then we copy p into b on a cell-by-cell basis. */
p = evenDivComments2(b, 10);
for (y=0; y <10; y++)
    b[y] = p[y];
printf("After evenDivComments2, b is: ");
for(y = 1; y < 10; y++)
    printf("%d, ", b[y]);
```

# Booleans

&& (“and”):

- *usage* is  $d \ \&\& \ s$ , for  $d, s$  booleans.
- *meaning* is like ‘and’ in English, eg, “it is dry and it is sunny”.

|| (“or”):

- *usage* is  $t \ || \ s$ , for  $t, s$  booleans.
- *meaning* is like ‘or’ in English, eg “Tesco or Scotmid will be open”.
- NOT *exclusive* or:  $t \ || \ s$  also holds if *both*  $t$  and  $s$  hold.

! (“not”):

- $!p$  is true if and only  $p$  is false.

## Examples

```
char c='F';
```

```
const int false=0; true=1;
```

```
(1 < 9) || (2 == 5)
```

```
IsSunny(today) || true
```

```
('A' <= c) && (c <= 'Z')
```

```
false && (1 == 1)
```

## Boolean as int

- Booleans are represented as integers in C.
- 1 is the value of a true expression:  
`(x == x)` is 1
- 0 is the value of a false expression:  
`x < x` is 0
- Non-zero values are treated as true:  
`while(45){ };` `/* loop forever */`

# Truth Table

expr1	expr2	!expr1	expr1 && expr2	expr1    expr2
false	false	true	false	false
false	true	true	false	true
true	false	false	false	true
true	true	false	true	true

## Truth Table (as int)

expr1	expr2	!expr1	expr1 && expr2	expr1    expr2
0	0	1	0	0
0	non-zero	1	0	1
non-zero	0	0	0	1
non-zero	non-zero	0	1	1

## “short-circuit” to testing

&& and || expressions are evaluated in order:

- eg, first && second
- Arithmetic expressions DO NOT have this property

For Boolean expressions, *evaluation* ends as soon as the outcome is known:

- eg false && never
- eg (x == x) || never

## Testing elements of an array

```
int CheckRange(int max, int *array, int length) {
    int i = 0;
    while (i < length) {
        if (array[i] > max)
            break;
        i++;
    }
    if (i < length)        /* We broke out of the loop early */
        return 0;
    else return 1;
}
```

## Testing elements ... “short-circuit” version

```
int CheckRange2(int max, int *array, int length) {
    int i = 0;
    while ((i < length) && (array[i] <= max)) {
        i++;
    }
    if (i < length)        /* We broke out of the loop early */
        return 0;
    else return 1;
}
```

# Incrementing and Decrementing

There are 4 ways to increment a variable:

| `x = x+1;` | `x += 1;` | `++x;` | `x++;` |

4 ways to decrement a variable:

| `x = x-1;` | `x -= 1;` | `--x;` | `x--;` |

## Side-effects

`++x` (“pre-increment”):

Add 1, *then* return the result to the expression `++x`; is appearing in.

```
int x = 10;
printf("%d\n", ++x);
```

will print 11 to standard output.

`x++` (“post-increment”):

Return value of `x` to the expression `++x`; appears in, then add 1 to `x`.

```
int x = 10;
printf("%d\n", x++);
```

will print 10 to standard output.

## Use in practice

Function to read a line of characters:

```
ch = getchar();  
buffer[length] = ch;  
++length;
```

... is more commonly written as

```
ch = getchar();  
buffer[length++] = ch;
```

## Watch out . . .

Don't assume that *arithmetic* expressions will evaluate in order. For example:

```
x = 10;  
y = ++x + x;
```

In practice, depending on compiler, this could evaluate as either of the following:

```
y = 11 + 11;      /* ++x; y = x + x; */
```

```
y = 11 + 10;     /* y = x; ++x; y += x; */
```

Avoid writing code with these ambiguous interpretations.

## Precedence - highest to lowest

()	[]	(<var>++)	(<var>--)	
++(<var>)	--(<var>)			
*	/	%		
+	-			
<	<=	>	>=	
==	!=			
&&				
=	+=	--	*=	/= etc

# Precedence

Higher precedence (in one expression)  $\Rightarrow$  gets done first

Equal precedence (in one expression) - left to right.

## Watch out . . .

The common mathematical short-hand  $3 < j < 6$

. . . is evaluated as  $(3 < j) < 6$

Suppose  $j$  is 7. Then the sequence of evaluations is:

```
(3 < 7) < 5
```

```
= 1 < 5    /* 1 is the result (true) of 3 < 7 */
```

```
= 1        /* representing true */
```

Must be clear and write  $(3 < j) \ \&\& \ (j < 6)$

## Assigned Reading (Kelley and Pohl)

For Merge-Sort (version where length is a power-of-2): §6.9

For Boolean expressions, §4.1, §4.2, §4.3, §4.4.

For Increment/Decrement, §2.10

(if you don't have Kelley & Pohl, find another C-programming textbook and read similar sections in that.)