CS240: Programming in C

#include<stdio.h>

int main() {
    int fahr, celsius;
    const int lower = 10, upper = 300, step = 10;

    fahr = lower;
    while (fahr <= upper) {
        celsius = 5* (fahr - 32) / 9;
        printf("%d\t%d\n", fahr, celsius);
        fahr = fahr + step;
    }

    return 0;
}
# Example

```c
#include<stdio.h>

int main() {
    int fahr, celsius;
    const int lower = 10, upper = 300, step = 10;

    fahr = lower;
    while (fahr <= upper) {
        celsius = 5* (fahr - 32) / 9;
        printf("%d\t%d\n", fahr, celsius);
        fahr = fahr + step;
    }

    return 0;
}
```
Operators

- Arithmetic: +, -, *, /, %
- Relational: <, >, <=, >=, !=, ==
- Logical: ||, &&, !
- Increment/decrement: ++, --
- Bitwise: |, &, >>, <<, ^, ~
- Assignment: =, +=, -=, *=, /=, %=, <<=,
              >>=, &=, ^=, |=

Sunday, January 23, 2011
int bitcount(unsigned x) {
    int b;
    for (b=0; x != 0; x >>= 1)
        if (x & 01)
            b++;
    return (b);
}

what does bitcount(32) return?

bitcount(15)?

bitcount(34)?
Conditional expressions

\[
\begin{align*}
\text{if } (a > b) \\
&\quad z = a; \\
\text{else} \\
&\quad z = b; \\
\text{z} &= (a > b) \ ? \ a : b;
\end{align*}
\]

\textit{expression}_1 \ ? \ \textit{expression}_2 : \textit{expression}_3 \\
\textit{expression}_1 \text{ evaluated first, then} \\
\text{if true } \textit{expression}_2 \text{ is evaluated} \\
\text{if false } \textit{expression}_3 \text{ is evaluated}
If else

if (a > b)
    max = a;
else
    max = b;

if (expression)
    statement_1
else
    statement_2

If expression is true (non-zero) statement_1 is executed
Otherwise statement_2 is executed
Nested if-else: when things go wrong

```c
if ( n >= 0)
    printf (...);
    some_function(...);
if(n % 2 == 0) {
    ...
    ...
}
else  /* you mean n is negative*/

OOPS! The compiler will associate the else with the closest if (n%2)
Use braces to fix it!
```
Nested if-else

- To avoid ambiguity the else is associated with the closest else-less if

```c
if (n > 0)
  if (a > b)
    z = a;
  else
    z = b;
```

Always safer to use braces if you’re not sure!
Else-if

if (expression_1)
    statement_1
else if (expression_2)
    statement_2
else if (expression_3)
    statement_3
else if (expression_4)
    statement_4
else
    statement_5
## Precedence and associativity

<table>
<thead>
<tr>
<th>Operators</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>()  []  -&gt;  .</td>
<td>Left to right</td>
</tr>
<tr>
<td></td>
<td>~  ++  --  +  -  *  &amp;  (type)  sizeof</td>
</tr>
<tr>
<td>*  /  %</td>
<td>Left to right</td>
</tr>
<tr>
<td>+  -</td>
<td>Left to right</td>
</tr>
<tr>
<td>&lt;&lt;  &gt;&gt;</td>
<td>Left to right</td>
</tr>
<tr>
<td>&lt;  &lt;=  &gt;  &gt;=</td>
<td>Left to right</td>
</tr>
<tr>
<td>==  !=</td>
<td>Left to right</td>
</tr>
<tr>
<td>&amp;</td>
<td>Left to right</td>
</tr>
<tr>
<td>^</td>
<td>Left to right</td>
</tr>
<tr>
<td>l</td>
<td>Left to right</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>Left to right</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>?:</td>
<td>Right to left</td>
</tr>
<tr>
<td>=  +=  -=  *=  /=  %=  &amp;=  ^=</td>
<td>=  &lt;&lt;=  &gt;&gt;=</td>
</tr>
<tr>
<td>,</td>
<td>Left to right</td>
</tr>
</tbody>
</table>
Unpredictable results

- $s[i] = i++;$
- Is the subscript the old one or the new one?
- It is unspecified so different compilers treat this issue differently

$$x = f() + g()$$
It is not specified which is evaluated first -- $f()$ or $g()$
Order of evaluation ....

• Referential transparency revisited
• When does this matter?
• Is this a good or bad design feature?
  ▪ Implications for implementations
Example else-if

```c
int binsearch (int x, int v[], int n) {
    int low, high, mid;

    low = 0;
    high = n-1;
    while (low <= high) {
        mid = (low+high) / 2;
        if (x < v[mid])
            high = mid - 1;
        else if (x > v[mid])
            low = mid + 1;
        else
            return mid;
    }
    return -1;
}
```
Switch

switch (expression) {
    case const_expr: statements
    case const_expr: statements
    case const_expr: statements
    default: statements
}

Statements often ends with a break statement which causes the exit from switch.
Example switch

switch (day) {
    case MONDAY:
    case WEDNESDAY:
        prepare_class();
        break;
    default:
        other_stuff();
        break;
}

How do we generate the symbols MONDAY and WEDNESDAY?
Break and continue

- **break**: provides early exist from a `for`, `while`, `do`, and `switch`

- **continue**: continues the next iteration for a `for`, `while` or `do` to begin

```c
for (i=0; i<n; i++) {
    if(a[i]<0)
        continue /* skips negative elements*/
    ...
}
```
Go to

- **goto label**: makes the program jump to the label

```plaintext
for ()
    for ()
        ...
        if(failure)
            goto error;
error:
    clean up the mess
```
For a number of years I have been familiar with the observation that the quality of programmers is a decreasing function of the density of `go to` statements in the programs they produce.

... our intellectual powers are rather geared to master static relations and that our powers to visualize processes evolving in time are relatively poorly developed. For that reason we should do (as wise programmers aware of our limitations) our utmost to shorten the conceptual gap between the static program and the dynamic process, to make the correspondence between the program (spread out in text space) and the process (spread out in time) as trivial as possible.

Edsger W. Dijkstra

1968
Readings for this lecture

K&R Chapter 2 and 3