Statements

• variable = something; or control action (for example, printf, return)
  • Examples: \( x = y + 9; \) return; printf(“%f %d\n”, f, g);

• Semicolon “;” ends statements; it does not separate them
  • Right: \( x = y + 9; \) printf(“%d\n”, x);
  • Wrong: \( x = y + 9; \) printf(“%d\n”, x) [compiler error]
  • Wrong: \( x = y + 9, \) printf(“%d\n”, x); [unexpected result]

• Expressions can be statements; they have value
  • Example: \( x = y = 0; \) is \( x = (y = 0); \) so both \( x \) and \( y \) are set to 0
Logical Constants and Operators

• In C, 0 is false and anything non-zero is true
  • If the compiler evaluates an expression that is true, the value is 1

• Operators
  • greater than: \( x > y \)
  • greater than or equal to: \( x \geq y \)
  • equal to: \( x == y \)
  • less than: \( x < y \)
  • less than or equal to: \( x \leq y \)
  • not equal to: \( x != y \)

• Example: \( x = 7; \ y = 19; \ z = (x >= y); \)  [here \( z \) is 0 (false)]
• Example: \( x = 7; \ y = 19; \ z = (x != y); \)  [here \( z \) is 1 (true)]
Logical Combination Operators

Logical and: $x \&\& y$ (1 if both $x$ and $y$ are true)
Logical or: $x \mid\mid y$ (1 if either $x$ or $y$ (or both) are true)
Logical not: $!x$ (1 if $x$ is false, 0 if $x$ is true)

<table>
<thead>
<tr>
<th>$x$</th>
<th>$y$</th>
<th>$x &amp;&amp; y$</th>
<th>$x \mid\mid y$</th>
<th>$!x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>
Precedence and Associativity

- ! has highest precedence, associates right to left
- && comes next, associates left to right
- || comes next, associates left to right
- ! comes before the arithmetic operators
- && and || come after
Lazy Evaluation

• C evaluates logical operators left to right
• It stops *as soon as it can determine the result*
• Examples: let $x = 12; y = 29; z = -1$; then
  • $(x > y \mid \mid (y < z \&\& x < z)) = 0$
    
    $[x > y$ is false, so evaluate the $\&\&; y < z$ is false, so $\&\&$ is false, so $\mid \mid$ is false, stop$]$
  
  • $(x > y \mid \mid y > z \&\& x > z) = 1$
    
    $[x > y$ is false, so evaluate the $\&\&; y > z, x > z$ are true, so $\&\&$ is true, so $\mid \mid$ is true, stop$]$
  
  • $x > y \&\& y > z = 0$
    
    $[x > y$ is false, $\&\&$ is false, stop$]$
Conditional Branching: if

```java
if (condition){
  statements
}
```

• Test `condition`
• If true, execute the `statements`
• If false, do not execute the `statements`
• Note: if there is only one `statement`, you can omit the `{ }`
Example

\(x = 12;\)
if (\(x == 12\))
    \(\text{printf(“x is 12!”);}\)
if (\(x < 12\))
    \(\text{printf(“x is less than 12!”);}\)

• \(x\) is indeed 12, so print “\(x\) is 12!”
• \(x\) is not less than 12, so the second if prints nothing
Conditional Branching: if/else

if (condition){
    \textit{if\_statements}
}
else {
    \textit{else\_statements}
}
\begin{itemize}
\item Test \textit{condition}
\item If true, execute the \textit{if\_statements}
\item If false, do not execute the \textit{else\_statements}
\item Note: if there is only one statement in the if or else, you can omit the \{ \}
\end{itemize}
Examples

\[
x = 12;
\]
if (x == 12)
  printf("x is 12!");
else
  printf("x is not 12!");

• x is indeed 12, so print
  "x is 12!"

\[
x = -3;
\]
if (x == 12)
  printf("x is 12!");
else
  printf("x is not 12!");

• x is not 12, so print
  "x is not 12!"
Conditional Branching: Nested ifs

```c
if (condition1){
    if1_statements
}
else {
    if (condition2){
        if2_statements
    }
    else {
        else_statements
    }
}
```

- Test `condition1`
- If true, execute the `if1_statements`
- If false, go to else and test `condition2`
- If true, execute the `if2_statements`
- If false, execute the `else_statements`
Example

```c
if (x == 12)
    printf("x is 12!");
else{
    if (x == 11)
        printf("x is 11!");
    else{
        if (x == 10)
            printf("x is 10!");
        else
            printf("x is not 10, 11, or 12!");
    }
}
```

- If x is 12, prints “x is 12!”
- If x is 11, prints “x is 11!”
- If x is 10, prints “x is 10!”
- If x is 28, prints “x is not 10, 11, or 12!”
Conditional Branching: A Cleaner Way

if (condition1) {
    if1_statements
}
else if (condition2) {
    if2_statements
}
else {
    else_statements
}

• Test condition1
• If true, execute the if1_statements
• If false, go to else and test condition2
• If true, execute the if2_statements
• If false, execute the else_statements
Example

if (x == 12)
    printf("x is 12!\n");
else if (x == 11)
    printf("x is 11!\n");
else if (x == 10)
    printf("x is 10!\n");
else
    printf("x is not 10, 11, or 12!\n");

• If x is 12, prints “x is 12!”
• If x is 11, prints “x is 11!”
• If x is 10, prints “x is 10!”
• If x is 28, prints
  “x is not 10, 11, or 12!”
Conditional Branching: switch Statement

```
switch(expression){
    case case1:
        statements1;
        break;
    case case2:
        statements2;
        break;
    default:
        statementsd;
        break;
}
```

- Evaluate `expression`
- If it evaluates to `case1`, execute `statements1` and leave the switch
- If it evaluates to `case2`, execute `statements2` and leave the switch
- Otherwise, execute `statementsd` and leave the switch
- Each of the cases must be different
- `case1, case2` must be a constant — no variables or expressions
Example

```
switch(x){
case 12:
    printf("x is 12!");
    break;
case 11:
    printf("x is 11!");
    break;
case 10:
    printf("x is 10!");
    break;
default:
    printf("x is not 10, 11, or 12!");
}
```

- If x is 12, prints “x is 12!”
- If x is 11, prints “x is 11!”
- If x is 10, prints “x is 10!”
- If x is 28, prints “x is not 10, 11, or 12!”
Example, But Omitting break

switch(x){
case 12:
    printf(“x is 12!”);
break;
case 11:
    printf(“x is 11!”);
    break;
case 10:
    printf(“x is 10!”);
    break;
default:
    printf(“x is not 10, 11, or 12!”);
}

- If x is 12, prints “x is 12!x is 11”
- If x is 11, prints “x is 11!”
- If x is 10, prints “x is 10!”
- If x is 28, prints “x is not 10, 11, or 12!”

Note: leaving off the “break” at the end works, but is very bad form (because someone may add a case after it and not notice there is no break in the one above)
Loops in C

• for loop
  • When you know where you will stop

• while loop

• do ... while loop
  • When termination depends on a condition being satisfied
for loop

for (initialization; condition; increment)

• Examples:
  
  for (i = 1; i < 10; i++)
  for (j < 10; j += 3)
  for (x < 10; )
  for (; ; )
while loop

while (condition)
    ...

• Examples:
    while (i < 10)
        i = i + 1;
    while (j != 13)
        j = j - 1;
    while (1)
        ;

• condition goes at top of loop; if condition is initially false, the loop is skipped
do ... while loop

do{
    ... 
} while (condition)

• Examples:
  do{
    i = i + 1;
  } while (i != 13);
  do{
    ;
  } while (1);
• condition goes at bottom of loop, which is always executed at least once