CENG 230

Introduction to C Programming

Week 4 – Overview of C

Sinan Kalkan

Some slides/content are borrowed from Tansel Dokeroglu, Nihan Kesim Cicekli.
<table>
<thead>
<tr>
<th>Algebraic equality or relational operator</th>
<th>C equality or relational operator</th>
<th>Example of C condition</th>
<th>Meaning of C condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equality operators</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>=</td>
<td>==</td>
<td>x == y</td>
<td>x is equal to y</td>
</tr>
<tr>
<td>≠</td>
<td>!=</td>
<td>x != y</td>
<td>x is not equal to y</td>
</tr>
<tr>
<td>Relational operators</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt;</td>
<td>&gt;</td>
<td>x &gt; y</td>
<td>x is greater than y</td>
</tr>
<tr>
<td>&lt;</td>
<td>&lt;</td>
<td>x &lt; y</td>
<td>x is less than y</td>
</tr>
<tr>
<td>≥</td>
<td>≥</td>
<td>x ≥ y</td>
<td>x is greater than or equal to y</td>
</tr>
<tr>
<td>≤</td>
<td>≤</td>
<td>x ≤ y</td>
<td>x is less than or equal to y</td>
</tr>
</tbody>
</table>

Fig. 2.12 | Equality and relational operators.

! exclamation mark

= is assignment and == is an equality operator
**Increment, Decrement Operators**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Sample expression</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>++</td>
<td>++a</td>
<td>Increment <code>a</code> by 1, then use the new value of <code>a</code> in the expression in which <code>a</code> resides.</td>
</tr>
<tr>
<td>++</td>
<td><code>a++</code></td>
<td>Use the current value of <code>a</code> in the expression in which <code>a</code> resides, then increment <code>a</code> by 1.</td>
</tr>
<tr>
<td>--</td>
<td><code>--b</code></td>
<td>Decrement <code>b</code> by 1, then use the new value of <code>b</code> in the expression in which <code>b</code> resides.</td>
</tr>
<tr>
<td>--</td>
<td><code>b--</code></td>
<td>Use the current value of <code>b</code> in the expression in which <code>b</code> resides, then decrement <code>b</code> by 1.</td>
</tr>
</tbody>
</table>

*Fig. 3.12* | Increment and decrement operators

- `++a, --a`
- `vs`
- `a++, a--`
Assignment operators

= 
+= 
-= 
*= 
/= 
%= 

a+=10; is the same with 
a=a + 10;
Compound Assignment Operators

- `var op= expr`
- `+=  -=  *=  /=  %=`
Some examples

• i += j = k;
• i = j += k;
Relational Operators

- `<=`, `>`, `>=`, `==`, `!=`
- False means 0 (zero)
- True means anything that is not False (i.e., non-zero)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Type</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>+</code> <code>-</code></td>
<td>Unary</td>
<td>Right to left</td>
</tr>
<tr>
<td><code>++</code> <code>--</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>*</code> <code>/</code> <code>%</code></td>
<td>Binary</td>
<td>Left to right</td>
</tr>
<tr>
<td><code>+</code> <code>-</code></td>
<td>Binary</td>
<td>Left to right</td>
</tr>
<tr>
<td><code>&lt;</code> <code>&lt;=</code> <code>&gt;</code> <code>=&gt;</code></td>
<td>Binary</td>
<td>Left to right</td>
</tr>
<tr>
<td><code>==</code> <code>!=</code></td>
<td>Binary</td>
<td>Left to right</td>
</tr>
<tr>
<td><code>=</code> <code>*=</code> <code>/=</code> <code>%=</code> <code>+=</code> <code>-=</code></td>
<td>Binary</td>
<td>Right to left</td>
</tr>
</tbody>
</table>

Example: `a = b + c <= d + e == c - d`
Today

• Finish up operators
• Type conversion
• Defining macros
• Examples
• Changing the flow of the program
## Logical Operators

- `&&`  
- `||`  
- `!`

<table>
<thead>
<tr>
<th>Operator</th>
<th>Type</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>+ - ++ -- !</code></td>
<td>Unary</td>
<td>Right to left</td>
</tr>
<tr>
<td><code>* / %</code></td>
<td>Binary</td>
<td>Left to right</td>
</tr>
<tr>
<td><code>+ -</code></td>
<td>Binary</td>
<td>Left to right</td>
</tr>
<tr>
<td><code>&lt; &lt;= &gt; &gt;=</code></td>
<td>Binary</td>
<td>Left to right</td>
</tr>
<tr>
<td><code>== !=</code></td>
<td>Binary</td>
<td>Left to right</td>
</tr>
<tr>
<td><code>&amp;&amp;</code></td>
<td>Binary</td>
<td>Left to right</td>
</tr>
<tr>
<td>`</td>
<td></td>
<td>`</td>
</tr>
<tr>
<td><code>= *= /= %= += -=</code></td>
<td>Binary</td>
<td>Right to left</td>
</tr>
</tbody>
</table>
Type conversion (casting)
Type conversions (casting)

```c
float a = 5.25;
int b = a;
/*Casting from float to int. The value of b here is 5*/

char c = ‘A’;
int x = c;
/*Casting from char to to int.
The value of x here is 65: the ASCII code of ‘A’*/

int x=7, y=5;
float z;
z=x/y;
/* the value of z is 1.00 */

int x=7, y=5;
float z;
z = (float)x/(float)y;
/ the value of z is 1.4*/
```
Type conversions (casting)

```c
int sum = 17, count = 5;
double mean;
mean = (double) sum / count;
printf("Value of mean : \%f\n", mean);

Value of mean : 3.400000
```

Output is: 1 and **fraction** part of the number is lost

```c
int sum = 17, count = 5;
double mean;
mean = (double) sum / count;
printf("Value of mean : \%f\n", mean);

Value of mean : 3.400000
```

```c
int i = 17;
char c = 'c'; /* ascii value is 99 */
int sum;
sum = i + c;
printf("Value of sum : \%d\n", sum);

Value of sum : 116
```
What is the result of `printf("%d", 'd' - 'a');`?

TABLE 2.7  ASCII Codes for Characters

<table>
<thead>
<tr>
<th>Character</th>
<th>ASCII Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>' '</td>
<td>32</td>
</tr>
<tr>
<td>'*'</td>
<td>42</td>
</tr>
<tr>
<td>'A'</td>
<td>65</td>
</tr>
<tr>
<td>'B'</td>
<td>66</td>
</tr>
<tr>
<td>'Z'</td>
<td>90</td>
</tr>
<tr>
<td>'a'</td>
<td>97</td>
</tr>
<tr>
<td>'b'</td>
<td>98</td>
</tr>
<tr>
<td>'z'</td>
<td>122</td>
</tr>
<tr>
<td>'0'</td>
<td>48</td>
</tr>
<tr>
<td>'9'</td>
<td>57</td>
</tr>
</tbody>
</table>
Type Conversion

Automatic Type Conversion Rules

char, short → int → long → float → double → long double

* Advice: Avoid automatic type conversion!
int m, n;

double p, x, y;

m = 3;
n = 2;
p = 2.0;
x = m / p;  /* 3/2.0 */
y = m / n;  /* 3/2 */

m  n  p  x  y
3   2  2.0 1.5 1.0

x = 9 * 0.5;
n = 9 * 0.5;

evaluates to the real number 4.5. If x is of type double, the number 4.5 is stored in x, as expected. If n is of type int, only the integral part of the expression value is stored in n, as shown.
Changing the flow of the program

*If statements*
Changing the flow of the program

• if statements
  
  ```
  if(expr)
  {
    ....
  }
  else if(expr)
  {
    ...
  }
  else
  {
    ...
  }
  ```

  ```
  if(a > b)
  printf("a is bigger");
  else if(a < b)
  printf("b is bigger");
  else
  printf("a = b");
  ```
Changing the flow of the program

- Common mistake with if statements
  - `if( a == 10) { ... }`
  - `if( a == 10); { ... }`
Nested if statements

• if( ... )
  
  if( ... )
  
  {.....}

else

{.....}