CSCI-UA.0201

Computer Systems Organization

C Programming – Basics (Part 2)

Thomas Wies  
wies@cs.nyu.edu  
https://cs.nyu.edu/wies
Now that we know about variables, let’s combine them to form expressions!

\[ X = 2 \times Y + Z; \]
How Expressions Are Evaluated?

Expressions combine Values using Operators, according to precedence.

1 + 2 * 2  →  1 + 4  →  5
(1 + 2) * 2  →  3 * 2  →  6

Comparison operators are used to compare values.
In C: 0 means “false”, and any other value means “true”.

```c
int x=4;
(x < 5)  →  (4 < 5)  →  <true>
(x < 4)  →  (4 < 4)  →  0
((x < 5) || (x < 4))  →  (<true> || (x < 4))  →  <true>
```

Not evaluated because first clause was true
Precedence

• Highest to lowest
  • ()
  • *, /, %
  • +, -

When in doubt, use parenthesis.
Comparison and Mathematical Operators

== equal to
< less than
<= less than or equal
> greater than
>= greater than or equal
!= not equal
&& logical and
|| logical or
! logical not

Beware in division:
If second argument is integer, the result will be integer (rounded):
5 / 10 → 0 whereas 5 / 10.0 → 0.5

& bitwise and
| bitwise or
^ bitwise xor
~ bitwise not
<< shift left
>> shift right

More on these in later lectures when we discuss binary numbers.
# Assignment Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x = y</code></td>
<td>assign <code>y</code> to <code>x</code></td>
<td><code>x = 10;</code></td>
</tr>
<tr>
<td><code>x++</code></td>
<td>post-increment <code>x</code></td>
<td><code>int x = 5; x++;</code></td>
</tr>
<tr>
<td><code>++x</code></td>
<td>pre-increment <code>x</code></td>
<td><code>int x = 5; ++x;</code></td>
</tr>
<tr>
<td><code>x--</code></td>
<td>post-decrement <code>x</code></td>
<td><code>int x = 5; x--;</code></td>
</tr>
<tr>
<td><code>--x</code></td>
<td>pre-decrement <code>x</code></td>
<td><code>int x = 5; --x;</code></td>
</tr>
<tr>
<td><code>x += y</code></td>
<td>assign <code>(x+y)</code> to <code>x</code></td>
<td><code>int x = 5; x += 3;</code></td>
</tr>
<tr>
<td><code>x -= y</code></td>
<td>assign <code>(x-y)</code> to <code>x</code></td>
<td><code>int x = 5; x -= 3;</code></td>
</tr>
<tr>
<td><code>x *= y</code></td>
<td>assign <code>(x*y)</code> to <code>x</code></td>
<td><code>int x = 5; x *= 3;</code></td>
</tr>
<tr>
<td><code>x /= y</code></td>
<td>assign <code>(x/y)</code> to <code>x</code></td>
<td><code>int x = 5; x /= 3;</code></td>
</tr>
<tr>
<td><code>x %= y</code></td>
<td>assign <code>(x%y)</code> to <code>x</code></td>
<td><code>int x = 5; x %= 3;</code></td>
</tr>
</tbody>
</table>

Note the difference between `++x` and `x++`:

```c
int x=5;
int y;
y = ++x;
/* x == 6, y == 6 */
```

```c
int x=5;
int y;
y = x++;
/* x == 6, y == 5 */
```

Don’t confuse `=` and `==`

```c
int x=5;
if (x==6) /* false */
{
    /* ... */
}
/* x is still 5 */
```

```c
int x=5;
if (x=6) /* always true */
{
    /* x is now 6 */
}
/* ... */
```
Evaluation Order of Expressions

• Unlike many other languages, the semantics of C does not specify the order in which operands are evaluated.
• So be careful when subexpressions have side effects!

Example:

```c
int x = 0;
x = x++ + (x + 1);
```
Can be evaluated as

```c
int x = 0;
int tmp1 = x++;
int tmp2 = x + 1;
int x = tmp1 + tmp2;
// x == 2
```

or

```c
int x = 0;
int tmp1 = x + 1;
int tmp2 = x++;
int x = tmp2 + tmp1;
// x == 1
```
Functions
What is a Function?

A Function is a series of instructions to run.
You pass Arguments to a function and it returns a Value.

“main()” is a Function. It’s only special because it always gets called first when you run your program.

```
#include <stdio.h>

/* The simplest C Program */

int main(int argc, char **argv)
{
    printf("Hello World\n");
    return 0;
}
```

“printf()” is just another function, like main(). It’s defined for you in a “library”, a collection of functions you can call from your program.
A More Complex Program: pow

#include <stdio.h>

float pow(float x, unsigned int exp)
{
    /* base case */
    if (exp == 0) {
        return 1.0;
    }
    /* "recursive" case */
    return x*pow(x, exp - 1);
}

int main(int argc, char **argv)
{
    float p;
    p = pow(10.0, 5);
    printf("p = %f\n", p);
    return 0;
}

“if” statement

/* if evaluated expression is not 0 */
if (expression) {
    /* then execute this block */
}
else {
    /* otherwise execute this block */
}

Tracing “pow()”:  
• What does pow(5,0) do?  
• What about pow(5,1)?
Recall scoping. If a variable is valid “within the scope of a function”, what happens when you call that function recursively? Is there more than one “exp”?

Yes. Each function call allocates a “stack frame” where Variables within that function’s scope will reside.

```c
#include <stdio.h>
#include <inttypes.h>

float pow(float x, unsigned int exp)
{
    /* base case */
    if (exp == 0) {
        return 1.0;
    }

    /* “recursive” case */
    return x*pow(x, exp - 1);
}

int main(int argc, char **argv)
{
    float p;
    p = pow(5.0, 1);
    printf("p = %f\n", p);
    return 0;
}
```

```
float x 5.0
uint32_t exp 0 Return 1.0
float x 5.0
uint32_t exp 1 Return 5.0
int argc 1
char **argv 0x2342
float p 5.0
```

Grows
The “for” loop

The “for” loop is just shorthand for this “while” loop structure.

float pow(float x, unsigned int exp)
{
    float result=1.0;
    int i;
    i=0;
    while (i < exp) {
        result = result * x;
        i++;
    }
    return result;
}

int main(int argc, char **argv)
{
    float p;
    p = pow(10.0, 5);
    printf(“p = %f\n”, p);
    return 0;
}
When to Use?

Different Loop-constructs

• while
• do-while
• for

Conditions

• if-else
• switch-case