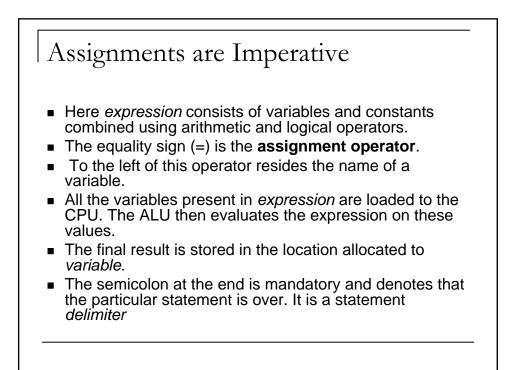


Assignments

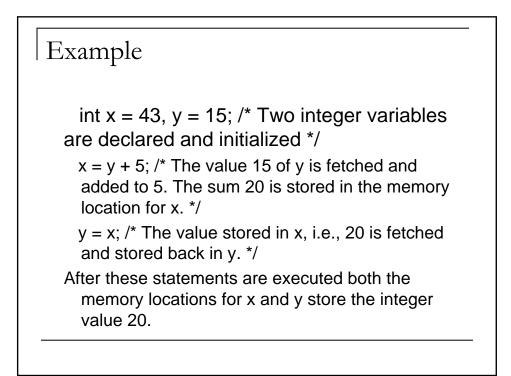
Assignments

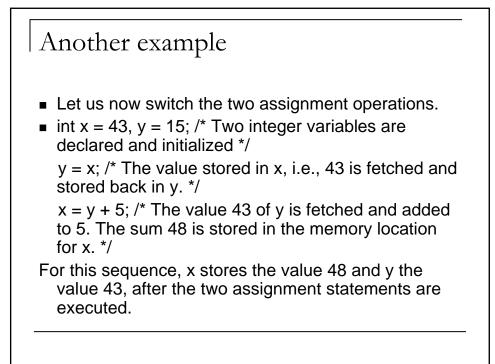
- Initialization during declaration helps one store constant values in memory allocated to variables.
 Later one typically does a sequence of the following:
 - Read the values stored in variables.
 - Do some operations on these values.
 - Store the result back in some variable.
- This three-stage process is effected by an assignment operation. A generic assignment operation looks like: variable = expression;

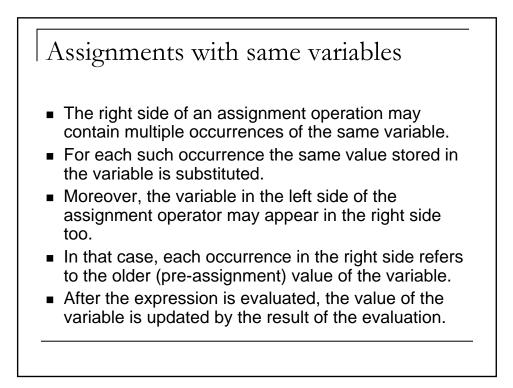


Imperative Programming

- A C program typically consists of a sequence of statements. They are executed one-by-one from top to bottom (unless some explicit jump instruction or function call is encountered). This sequential execution of statements gives C a distinctive **imperative** flavor.
- This means that the sequence in which statements are executed decides the final values stored in variables.

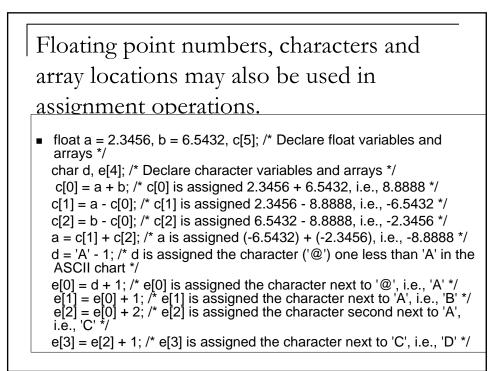






Example

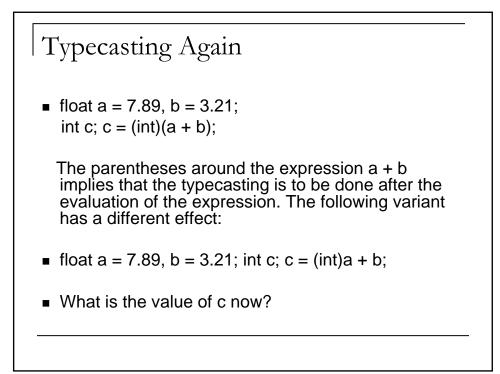
- int x = 5; x = x + (x * x);
- The value 5 stored in x is substituted for each occurrence of x in the right side, i.e., the expression 5 + (5 * 5) is evaluated.
- The result is 30 and is stored back to x.
- Thus, this assignment operation causes the value of x to change from 5 to 30.
- The equality sign in the assignment statement is not a mathematical equality, i.e., the above statement does <u>not</u> refer to the equation x = x + x² (which happens to have a single root, namely x = 0).

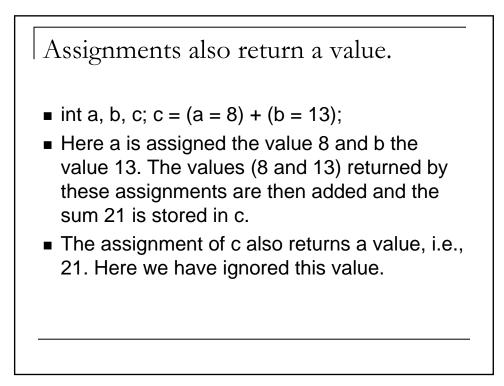


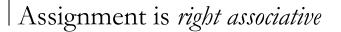
Implicit Conversion

- An assignment does an implicit type conversion, if its left side turns out to be of a different data type than the type of the expression evaluated.
- float a = 7.89, b = 3.21; int c; c = a + b;
- Here the right side involves the floating point operation 7.89 + 3.21. The result is the floating point value 11.1. The assignment plans to store this result in an integer variable.
- The value 11.1 is first truncated and subsequently the integer value 11 is stored in c.

```
Example
#include<stdio.h>
main()
{
    float a = -7.89., b = 3;
    int c;
    typedef unsigned long newlong;
    newlong d;
    c = (int) a + b;
    d=c;
    printf("%d\n",c);
    printf("%x\n",d);
}
```





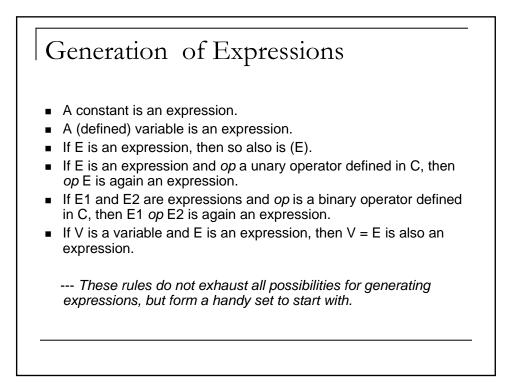


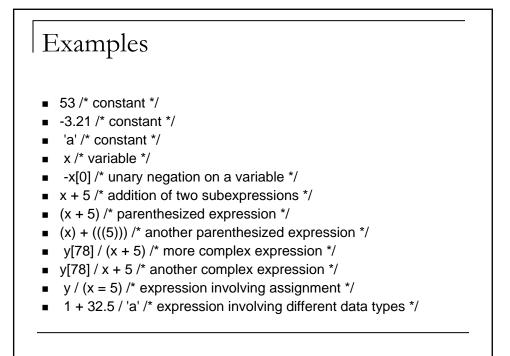
For example,

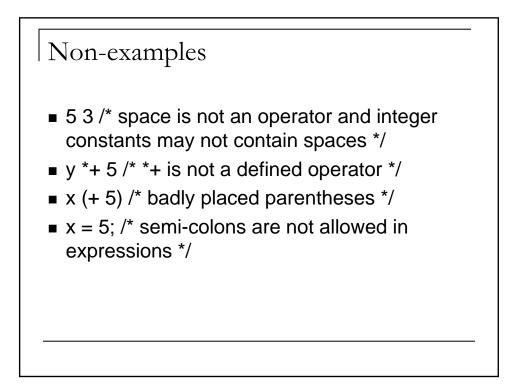
a = b = c = 0;

is equivalent to a = (b = (c = 0));

Here c is first assigned the value 0. This value is returned to assign b, i.e., b also gets the value 0. The value returned from this second assignment is then assigned to a. Thus after this statement all of a, b and c are assigned the value 0.

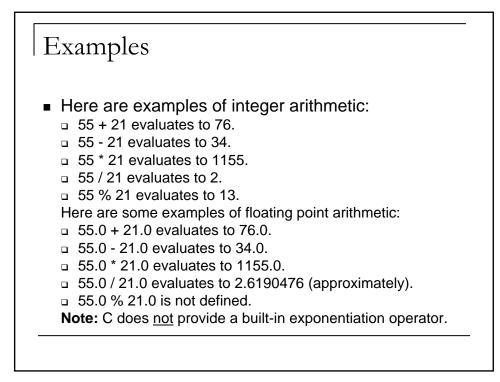


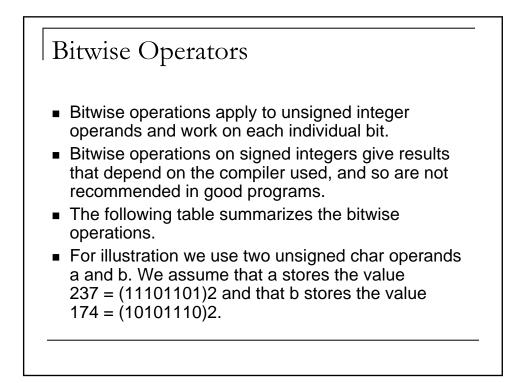




Oper ator	Meanin g	Description
-	unary negatio n	Applicable for integers and real numbers. Does not make enough sense for unsigned operands.
+	(binary) addition	Applicable for integers and real numbers.
-	(binary) subtract ion	Applicable for integers and real numbers.
*	(binary) multiplic ation	Applicable for integers and real numbers.

Op	erato:	rs in C
/	(binary) division	For integers division means "quotient", whereas for real numbers division means "real division". If both the operands are integers, the integer quotient is calculated, whereas if (one or both) the operands are real numbers, real division is carried out.
%	(binary) remaind er	Applicable only for integer operands.





Operator	Meaning	Example									
		a = 237	1	1	1	0	1	1	0	1	
&	AND	b = 174	1	0	1	0	1	1	1	0	
		a & b is 172	1	0	1	0	1	1	0	0	
Ι		a = 237	1	1	1	0	1	1	0	1	
	OR	b = 174	1	0	1	0	1	1	1	0	
		a b is 239	1	1	1	0	1	1	1	1	
^		a = 237	1	1	1	0	1	1	0	1	
	EXOR	b = 174	1	0	1	0	1	1	1	0	
		a ^ b is 67	0	1	0	0	0	0	1	1	
	Complement	a = 237	1	1	1	0	1	1	0	1	
~	Complement	~a is 18	0	0	0	1	0	0	1	0	
>>	Dight shift	a = 237	1	1	1	0	1	1	0	1	
	Right-shift	a >> 2 is 59	0	0	1	1	1	0	1	1	
	Left-shift	$\mathbf{b} = 174$	1	0	1	0	1	1	1	0	
<<		b << 1 is 92	0	1	0	1	1	1	0	0	

.			