CPNM Lecture 5 - Expressions

Mridul Sankar Barik

Jadavpur University

2022

Expressions

- C expressions are formulas that show how to compute a value
- Simplest expressions: variables and constants
- Operators are basic tools for building expressions
  - Arithmetic operators
  - Relational operators
  - Logical operators

Variables

- Identifiers (variable names, function names etc.) are made up of letters and digits, and are case-sensitive
- The first character of an identifier must be a letter, which includes underscore (_)
- C language has 32 keywords which are reserved and may not be used as identifiers

Constants

- Can have different types and representations
- Integer Constants: decimal (1234), octal (0123), hex (0x89af)
- Floating Point Constants: 12.34f, 123456.7890L
- Character Constants: 'a', 'Z', '\n', '\t'

Arithmetic Operators

- Unary: unary plus (+), unary minus (-)
  - Example: `i = +1; j = -i;`
- Binary: addition (+), subtraction (-), multiplication (*), division (/), remainder or modulo (%)
  - Except %, all binary operators allow integer or floating point or mixed operands
  - For mixed operands result has type `float`
  - When both operands are integer, / operator truncates the result
  - % operator requires integer operands (else compiler error)
  - Zero as the right operand of either / or % operator causes undefined behavior

Operator Precedence And Associativity I

- Precedence rules are enforced when an expression contains more than one operator
- Precedence of arithmetic operators

```
Highest: + - (unary)
         * / %
Lowest: + - (binary)
```

- Example:

  `i + j * k` $\Rightarrow$ `i + ( j * k )`

  `-i * -j` $\Rightarrow$ `(-i) * (-j)`

  `+i + j / k` $\Rightarrow$ `(+i) + ( j / k )`
- Associativity rules are enforced when an expression contains one or more operators at the same level of precedence

Operator Precedence And Associativity II

- Left Associative: if an operator groups from left to right; Left operand must be unambiguous; Binary arithmetic operators (+, -, /, *, %) are all left associative
  `i * j / k` $\Rightarrow$ `((i * j) / k )`
- Right Associative: if an operator groups from right to left; Right operand must be unambiguous; Unary arithmetic operators (+, -) are both right associative
  `- + i` $\Rightarrow$ `(- ( +i))`
- Unambiguous: It must not be involved in evaluation of any other sub-expression

Assignment Operators I

- Simple assignment ($=$) operator is used to update a value of a variable
- `v = e` $\Rightarrow$ evaluate expression `e` and copy its value into `v`
- If type of `v` and `e` don't match, then value of `e` is converted to the type of `v`
- In C, assignment is an operator. Value of `v = e`, is the value of `v`
- An operator is said to have **side effects** if it modifies its operands
- Simple assignment operator has side effects as it modifies its left operand
  `i = 0` produces result 0 and as a side effect assigns 0 to `i`
- Example: To check whether `i==0`

Assignment Operators II

```
if(i=0)
    printf("Hello\n");
else
    printf("World\n");
```

It always prints "World", irrespective of the value of `i`

- Example: To check whether `i==5`

```
if(i=5)
    printf("Hello\n");
else
    printf("World\n");
```

It always prints "Hello", irrespective of the value of `i`

- Simple assignment operator is right associative
  `i = j = k = 0;` $\Rightarrow$ `i = (j = (k = 0));`

Assignment Operators III

- Example:

```
int i;
float f;
f = i = 33.3;
```

  i is assigned the value 33 and then f is assigned 33.0

- Example:

```
i = 1;
k = 1 + ( j = i );
printf("%d %d %d", i, j, k);
```

  prints "1 1 2"

Lvalues

- Most C operators allow their operands to be variables, constants, or expressions containing other operators
- Assignment operators requires an **lvalue** as its left operand
- An **lvalue** represents an object stored in memory. Variables are lvalues, expressions are not.
- Examples of wrong lvalues:

```
12 = i;
i + j = 0;
-i = j;
```

Compound Assignment Operators

- Assignments that use old value of a variable to compute its new value can be shortened using compound assignment operators
  `i = i + 2;` can be written as i += 2;
- +=, -=, /=, *=, %=
- Right associative
  `i += j += k` ⇒ `i += (j += k);`

Increment and Decrement Operators I

- `++` (increment), `--` (decrement)
- Both can be used as either prefix (`++i`, `--i`) or postfix (`i++`, `i--`) operators
- Both `++` and `--` have side effects
  - `++i` yields `i+1` and as a side effect increments `i`
  - `i++` yields `i` and as a side effect increments `i`
- Operates on l-values only
- `++i++` or `(j=i++)--` generate compiler error
- Example:

```
i = 1;
printf("i is %d\n", ++i); /* prints 2*/
printf("i is %d\n", i); /*prints 2*/

i = 1;
printf("i is %d\n", i++); /* prints 1*/
printf("i is %d\n", i); /*prints 2*/
```

Expression Evaluation I

Precedence	Name	Symbol(s)	Associativity
1	increment (postfix)	++	left
	decrement (postfix)	--	
2	increment (prefix)	++	right
	decrement (prefix)	--	
	unary plus	+	
	unary minus	-	
3	multiplicative	* / %	left
4	additive	+ -	left
5	assignment	= *= /= %= += -=	right

Table 1: Partial List of C Operators

Example: Evaluate the expression

Expression Evaluation II

```
a = b += c++ - d + --e / -f
    /* ++ has highest precedence */

a = b += (c++) - d + --e / -f
    /* prefix -- and unary - have precedence 2 */

a = b += (c++) - d + (--e) / (-f)
    /* / operator has precedence 3*/

a = b += (((c++) - d) + ((--e) / (-f)))
    /* two operators with precedence 4; left associativity */

(a = (b += (((c++) - d) + ((--e) / (-f)))))
    /* remaining two assignment operators; right associativity*/
```

Expression Statement

In C any expression can be used as a statement (by appending a ;)

Example:

```
i++;
i + 1;
i + j;
```

Logical Expressions I

- Have either true (1) or false (0) value
- Relational operators: $<, >, <=, >=, ==, !=$
- Precedence of relational operators is lower than that of the arithmetic operators. Example: `i + j < k - 1` means `(i + j) < (k - 1)`
- The relational operators are left associative
- Example:

```
int i=15, j=9;
if(i>j)
    printf("Hello\n");
else
    printf("World\n");
```

  Here, `(i>j)` evaluates to true (1), "Hello" is printed
- Example:

Logical Expressions II

```
int i=15, j=9, k=5;
if(i>j>k)
    printf("Hello\n");
else
    printf("World\n");
```

Here, `(i>j>k)` is evaluated as `((i>j)>k)` $\Rightarrow$ `(1>k)` $\Rightarrow$ false (0), "World" is printed

- Logical Operators: ! (Logical Negation), && (Logical AND), || (Logical OR)
- Logical operators have lower precedence than relational operators, except NOT (!) operator which has precedence equal to that of other unary operators. AND has higher precedence than OR.

Precedence Table

	Operator	Associativity	Precedence
()	Function call	Left-to-Right	Highest 14
[]	Array subscript		
.	Dot (Member of structure)		
->	Arrow (Member of structure)		
!	Logical NOT	Right-to-Left	13
~	One's-complement		
-	Unary minus (Negation)		
++	Increment		
--	Decrement		
&	Address-of		
*	Indirection		
(type)	Cast		
sizeof	Sizeof		
*	Multiplication	Left-to-Right	12
/	Division		
%	Modulus (Remainder)		
+	Addition	Left-to-Right	11
-	Subtraction		
<<	Left-shift	Left-to-Right	10
>>	Right-shift		
<	Less than	Left-to-Right	8
<=	Less than or equal to		
>	Greater than		
>=	Greater than or equal to		
==	Equal to	Left-to-Right	8
!=	Not equal to		
&	Bitwise AND	Left-to-Right	7
^	Bitwise XOR	Left-to-Right	6
|	Bitwise OR	Left-to-Right	5
&&	Logical AND	Left-to-Right	4
||	Logical OR	Left-to-Right	3
? :	Conditional	Right-to-Left	2
=, += *=, etc.	Assignment operators	Right-to-Left	1
,	Comma	Left-to-Right	Lowest 0