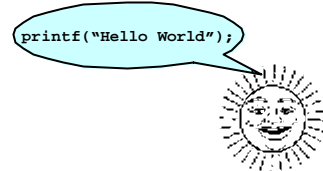


CSE1301
Computer Programming
Lecture 5:
C Primitives 2

Topics

- Expressions
- Precedence
- Function calls
- Comments



Expressions

- Combine values using operators and function calls
- Return a value of a known type

Arithmetic Expressions

- take arithmetic (numerical) values *and*
- return an arithmetic (numerical) value
- Are composed using the following operators:
 - + (unary plus)
 - (unary minus)
 - + (addition)
 - (subtraction)
 - * (multiplication)
 - / (division or quotient)
 - % (modulus or remainder)

Precedence in Expressions

- Defines the order in which an expression is evaluated

Precedence in Expressions -- Example

$1 + 2 * 3 - 4 / 5 =$
 $1 + (2 * 3) - (4 / 5)$

B.O.D.M.A.S.

B stands for brackets,
O for Order (exponents),
{ D for division, }
{ M for multiplication, }
{ A for addition, and }
{ S for subtraction. }



More on precedence

- *, /, % are at the same level of precedence
- +, - are at the same level of precedence
- For operators at the same "level", left-to-right ordering is applied.

$$2 + 3 - 1 = (2 + 3) - 1 = 4$$

$$2 - 3 + 1 = (2 - 3) + 1 = 0$$


$$2 * 3 / 4 = (2 * 3) / 4 = 6 / 4$$

$$2 / 3 * 4 = (2 / 3) * 4 = 0 * 4$$

Precedence in Expressions – Example (cont)

$$1 + 2 * 3 - 4 / 5 =$$


$$1 + (2 * 3) - (4 / 5)$$

6.2


Precedence in Expressions – Example (cont)

$$1 + 2 * 3 - 4 / 5 =$$

$$1 + (2 * 3) - (4 / 5)$$


6.2 ✗


Precedence in Expressions – Example (cont)

$$1 + 2 * 3 - 4 / 5 =$$

$$1 + (2 * 3) - \underbrace{(4 / 5)}$$


Integer division results in integer quotient



Precedence in Expressions – Example (cont)

$$1 + 2 * 3 - 4 / 5 =$$


$$1 + (2 * 3) - \underbrace{(4 / 5)}_{= 0}$$

D'oh


Precedence in Expressions – Example (cont)

$$1 + 2 * 3 - 4 / 5 =$$

$$1 + (2 * 3) - (4 / 5)$$

7 ✓


int-s and float-s

- **float** is a “communicable” type
- Example:

```
1 + 2 * 3 - 4.0 / 5
= 1 + (2 * 3) - (4.0 / 5)
= 1 + 6 - 0.8
= 6.2
```

int-s and float-s – Example 2

```
(1 + 2) * (3 - 4) / 5
= ((1 + 2) * (3 - 4)) / 5
= (3 * -1) / 5
= -3 / 5
= 0
```

int-s and float-s – Example 2 (cont)

```
(1 + 2.0) * (3 - 4) / 5
= ((1 + 2.0) * (3 - 4)) / 5
= (3.0 * -1) / 5
= -3.0 / 5
= -0.6
```

int-s and float-s – Example 3

```
(1 + 2.0) * ((3 - 4) / 5)
= (1 + 2.0) * (-1 / 5)
= 3.0 * 0
= 0.0
```

Unary operators

- Called *unary* because they require one operand.
- Example

```
i = +1; /* + used as a unary operator */
j = -i; /* - used as a unary operator */
```
- The unary + operator does nothing – just emphasis that a numeric constant is positive.
- The unary – operator produces the negative of its operand.

Increment and decrement operators

- ++ is the *increment* operator

```
i++;
```

is equivalent to

```
i = i + 1;
```
- -- is the *decrement* operator

```
j--;
```

is equivalent to

```
j = j - 1;
```

(King, pp53-54)

| Example -- Simple Expressions | |
|--|---------------------------------------|
| Evaluate an expression | <code>#include <stdio.h></code> |
| set result to <code>1 + 2 * 3 - 4 / 5</code> output result | |

| Example -- Simple Expressions (cont) | |
|--|---|
| Evaluate an expression | <code>#include <stdio.h></code> |
| set result to <code>1 + 2 * 3 - 4 / 5</code> output result | <code>/* Evaluate an expression */</code> |

| Example -- Simple Expressions (cont) | |
|--|---|
| Evaluate an expression | <code>#include <stdio.h></code> |
| set result to <code>1 + 2 * 3 - 4 / 5</code> output result | <code>/* Evaluate an expression */</code> |
| | <code>int main()</code> |
| | <code>{</code> |
| | <code>return 0;</code> |
| | <code>}</code> |

| Example -- Simple Expressions (cont) | |
|--|---|
| Evaluate an expression | <code>#include <stdio.h></code> |
| set <u>result</u> to <code>1 + 2 * 3 - 4 / 5</code> output <u>result</u> | <code>/* Evaluate an expression */</code> |
| | <code>int main()</code> |
| | <code>{</code> |
| | <code>float result;</code> |
| | <code>return 0;</code> |
| | <code>}</code> |

| Example -- Simple Expressions (cont) | |
|--|---|
| Evaluate an expression | <code>#include <stdio.h></code> |
| set result to <code>1 + 2 * 3 - 4 / 5</code> output result | <code>/* Evaluate an expression */</code> |
| | <code>int main()</code> |
| | <code>{</code> |
| | <code>float result;</code> |
| | <code>result = 1 + 2 * 3 - 4 / 5;</code> |
| | <code>return 0;</code> |
| | <code>}</code> |

| Example -- Simple Expressions (cont) | |
|--|---|
| Evaluate an expression | <code>#include <stdio.h></code> |
| set result to <code>1 + 2 * 3 - 4 / 5</code> output result | <code>/* Evaluate an expression */</code> |
| | <code>int main()</code> |
| | <code>{</code> |
| | <code>float result;</code> |
| | <code>result = 1 + 2 * 3 - 4 / 5;</code> |
| | <code>printf("%f\n", result);</code> |
| | <code>return 0;</code> |
| | <code>}</code> |

Example -- Simple Expressions (cont)

| | |
|--|---|
| <p>Evaluate an expression</p> <p>set result to 1 + 2 * 3 - 4 / 5 output result</p> | <pre>#include <stdio.h> /* Evaluate an expression */ int main() { float result; result = 1 + 2 * 3 - 4 / 5; printf("%f\n", result); return 0; }</pre> |
| <p>Output: 7.000000</p> | |

Topics

- ✓ Expressions
- ✓ Precedence
- Function calls
- Comments

Function Calls

- Tell the computer to execute a series of C commands and (maybe) return a value
 - *In algorithm-speak:* An invocation of a named sequence of instructions
- Example: printf, scanf, sqrt

Example -- Find the square root

Find the square root of x

output "Enter a number: "
input x

set myResult to result of **squareRoot(x)**

output myResult

Example -- Find the square root (cont)

```
#include <stdio.h>
#include <math.h>
/* Find square root */
int main()
{
    /* declare variables */
    float x, myResult;

    /* output "Enter a number: " */
    printf("Enter a number\n");
    /* input x */
    scanf("%f", &x);

    /* set myResult to result of squareRoot(x) */
    myResult=sqrt(x);

    /* output myResult */
    printf("Result is %f\n",myResult);
    return 0;
}
```

Topics

- ✓ History of C
- ✓ Structure of a C program
- ✓ Values and variables
- ✓ Expressions
- ✓ Function calls
- Comments

Comments

- Essential for documenting programs
- Run from a /* to the next */
- Examples:

```
/* THIS IS A COMMENT */  
  
/* So is  
   this      */  
  
/*  
** ...and this.  
**  
*/
```

Comments (cont)

- Comments do not “nest”

```
/* Comments start with a "/*"  
   and end with a "*/"  
   but they don't nest! */
```

Topics

- ✓ Expressions
- ✓ Precedence
- ✓ Function calls
- ✓ Comments

Reading

- King
 - Chapter 2, 2.1 – 2.3
 - Chapter 4, 4.1
- D&D:
 - Chapter 2, Sections 2.1 to 2.5
- Kernighan & Ritchie
 - Chapter 1, 1.2
 - Chapter 2, 2.1 – 2.3, 2.5 – 2.7, 2.12