CSE 220 - C Programming
Fall 2015

Expressions and Data Types
Expressions in C

• **Expressions**: Formulas to compute a value

  - \( d = (a + b) \times c \)
  - \( a = 2 \)
  - \( c = a \& b \)
Expressions in C

• Expressions = Operator (s) + Operand (s)
Expressions in C

• **Expressions** = Operator (s) + Operand (s)

• **Operators**: symbols to represent specific action:
  – **Arithmetic**: +, -, *, /, %
  – **Relational for comparisons**: >, <, >=, <=, ==, !=
  – **Logical to perform logical operations**: &&, ||, etc
  – **Assignment**
  – **Bitwise**
  – etc
Expressions in C

• **Expressions** = Operator (s) + Operand (s)

• **Operators**

• **Operands**: objects (variables) that are manipulated
  – Variables (x + y)
  – Constants (2 * x)

Expression can result in a value and can produce side effects.
Arithmetic Operators
Arithmetic Operators

- **Unary**: involves one operand
  - \[ i = +1; \quad j = -1; \]

<table>
<thead>
<tr>
<th>Unary Operators</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td><strong>Unary plus</strong> maintains the value of the operand. Any plus sign in front of a constant is not part of the constant.</td>
<td>+aNumber</td>
</tr>
<tr>
<td>−</td>
<td><strong>Unary minus</strong> operator negates the value of the operand. For example, if <code>num</code> variable has the value 200, (-num) has the value -200. Any minus sign in front of a constant is not part of the constant.</td>
<td>-342</td>
</tr>
</tbody>
</table>
Arithmetic Operators

- **Unary**: involves one operand

- **Binary**: requires two operands and are used to perform **mathematical calculations**
  - addition: +
  - subtraction: -
  - multiplication: *
  - division: /
  - remainder: % (11 % 3 evaluates to 2)
### Binary Arithmetic Operators

#### Multiplicative Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>*</code></td>
<td>The multiplication operator causes its two operands to be multiplied.</td>
<td>( p = q \times r );</td>
</tr>
</tbody>
</table>
| `/` | The division operator causes the first operand to be divided by the second. If two integer operands are divided and the result is not an integer, it is truncated according to the following rules:  
1. The result of division by 0 is undefined according to the ANSI C standard. The Microsoft C compiler generates an error at compile time or run time.  
2. If both operands are positive or unsigned, the result is truncated toward 0.  
3. If either operand is negative, whether the result of the operation is the largest integer less than or equal to the algebraic quotient or is the smallest integer greater than or equal to the algebraic quotient is implementation defined. | \( a = b / c \); |
| `%` | The result of the remainder operator is the remainder when the first operand is divided by the second. When the division is inexact, the result is determined by the following rules:  
1. If the right operand is zero, the result is undefined.  
2. If both operands are positive or unsigned, the result is positive.  
3. If either operand is negative and the result is inexact, the result is implementation defined. | \( x = y \% z \); |
Arithmetic Operators Example

```c
1 //using arithmetic operator
2 #include <stdio.h>
3 int main()
4 {
5     int a, b, sum, diff, product;
6     printf("Enter a:");
7     scanf("%d", &a);
8     printf("Enter b:");
9     scanf("%d", &b);
10    sum = a + b;
11    printf("a+b =\%d", sum);
12    diff = a - b;
13    printf("a-b =\%d", diff);
14    product = a* b;
15    printf("a*b =\%d", product);
16
17    return 0;
18 }
```
Arithmetic Operators Example

```c
//using arithmetic operator
#include <stdio.h>
int main()
{
    int a, b, sum, diff, product;
    printf("Enter a:");
    scanf("%d",&a);
    printf("Enter b:");
    scanf("%d",&b);
    sum = a + b;
    printf("a+b =\%d", sum);
    diff = a - b;
    printf("a-b =\%d", diff);
    product = a* b;
    printf("a*b =\%d", product);
    return 0;
}
```

Ranas–MacBook–Pro:CSE 220 ranaforsati$ ./arithmetic.e
Enter a:12
Enter b:10
a+b =22a-b =2a*b =120Ranas–MacBook–Pro:CSE 220 ranaforsati$
Arithmetic Operators

• +, -, *, /, %:
  – allow int and float operands or constants
  – If both of same type: evaluates as given type
  – If mixed: evaluates as float
    – 1.0 / 2 = ?
    – 1 / 2 = ?

• %: both operands must be integers

• Can NOT use 0 as right hand side of / and %
Arithmetic Operators

```c
//using arithmetic operator
#include <stdio.h>
int main()
{
    int a, b, sum, diff, product, c;
    printf("Enter a:\n");
    scanf("%d",&a);
    printf("Enter b:\n");
    scanf("%d",&b);
    sum = a + b;
    printf("a+b =%d\n", sum);
    diff = a - b;
    printf("a-b =%d\n", diff);
    product = a* b;
    printf("a*b =%d\n", product);
    c = a / b;
    printf("a/b =%d\n", c);

    return 0;
}
```
Arithmetic Operators

```c
//using arithmetic operator
#include <stdio.h>
int main()
{
    int a, b, sum, diff, product, c;
    printf("Enter a: \n");
    scanf("%d", &a);
    printf("Enter b: \n");
    scanf("%d", &b);
    sum = a + b;
    printf("a+b = %d\n", sum);
    diff = a - b;
    printf("a-b = %d\n", diff);
    product = a * b;
    printf("a*b = %d\n", product);
    c = a / b;
    printf("a/b = %d\n", c);
    return 0;
}
```

```
Enter a: 12
Enter b: 5
a+b = 17
a-b = 7
a*b = 60
a/b = 2
```
Arithmetic Operators

```c
#include <stdio.h>
int main()
{
    float a, b, sum, diff, product, c;
    printf("Enter a:\n");
    scanf("%f", &a);
    printf("Enter b:\n");
    scanf("%f", &b);
    sum = a + b;
    printf("a+b =%.2f\n", sum);
    diff = a - b;
    printf("a-b =%f\n", diff);
    product = a* b;
    printf("a*b =%f\n", product);
    c = a / b;
    printf("a/b =%f\n", c);

    return 0;
}
```
Arithmetic Operators

```c
//using arithmetic operator
#include <stdio.h>
int main()
{

    float a, b, sum, diff, product, c;
    printf ("Enter a:\n");
    scanf("%f", &a);
    printf ("Enter b:\n");
    scanf("%f", &b);
    sum = a + b;
    printf("a+b =%.2f\n", sum);
    diff = a - b;
    printf("a-b =%f\n", diff);
    product = a* b;
    printf("a*b =%f\n", product);
    c = a / b;
    printf("a/b =%f\n", c);

    return 0;
}
```

Enter a:
12
Enter b:
7
a+b =19.00
a-b =5.000000
a*b =84.000000
a/b =1.714286
Arithmetic Operators

```c
#include <stdio.h>
int main()
{
    int a, b, sum, diff, product, quotient, remind;
    printf("Enter a:\n");
    scanf("%d", &a);
    printf("Enter b:\n");
    scanf("%d", &b);
    sum = a + b;
    printf("a+b =%d\n", sum);
    diff = a - b;
    printf("a-b =%d\n", diff);
    product = a* b;
    printf("a*b =%d\n", product);
    quotient = a / b;
    printf("a/b =%d\n", quotient);
    remind = a % b;
    printf("a%b =%d\n", remind);

    return 0;
}
```
Arithmetic Operators

```c
//using arithmetic operator
#include <stdio.h>
int main()
{
    int a, b, sum, diff, product, quotient, remind;
    printf("Enter a:\n");
    scanf("%d",&a);
    printf("Enter b:\n");
    scanf("%d",&b);
    sum = a + b;
    printf("a+b =%d\n", sum);
    diff = a - b;
    printf("a-b =%d\n", diff);
    product = a* b;
    printf("a*b =%d\n", product);
    quotient = a / b;
    printf("a/b =%d\n", quotient);
    remind = a % b;
    printf("a%b =%d\n", remind);
    return 0;
}
```

Enter a:
12
Enter b:
4
a+b  =16
a-b  =8
a*b  =48
a/b  =3
ab   =0
Arithmetic Operators

```c
//using arithmetic operator
#include <stdio.h>
int main()
{
  int a, b, sum, diff, product, quotient, remind;
  printf("Enter a:\n");
  scanf("%d", &a);
  printf("Enter b:\n");
  scanf("%d", &b);
  sum = a + b;
  printf("a+b =\%d\n", sum);
  diff = a - b;
  printf("a-b =\%d\n", diff);
  product = a* b;
  printf("a\*b =\%d\n", product);
  quotient = a / b;
  printf("a/b =\%d\n", quotient);
  remind = a % b;
  printf("a%%b =\%d\n", remind);

  return 0;
}
```
Arithmetic Operators

```c
#include <stdio.h>

int main()
{
    int a, b, sum, diff, product, quotient, remind;
    printf("Enter a:\n");
    scanf("%d", &a);
    printf("Enter b:\n");
    scanf("%d", &b);
    sum = a + b;
    printf("a+b =%d\n", sum);
    diff = a - b;
    printf("a-b =%d\n", diff);
    product = a* b;
    printf("a*b =%d\n", product);
    quotient = a / b;
    printf("a/b =%d\n", quotient);
    remind = a % b;
    printf("a%b =%d\n", remind);

    return 0;
}
```

Enter a: 
12
Enter b: 
5
a+b  =17
a-b  =7
a*b  =60
a/b  =2
a%b  =2
Float Numbers in %

You will get a **compiler error** if use % with float numbers! Only integers

```c
#include <stdio.h>
int main()
{
    printf("The result of 10.5%5 is: %f", 10.5%5);
}
```
Division by 0

- Caution: No compiler error!

- It throws a runtime error!

inf: infinity (undefined)

indicate that you reached the boundaries of the range that your floating point values can assume. Depending on your architecture (hardware) and operating system the range can vary.
Operator Precedence

What is output of following code?

```c
#include <stdio.h>

int main()
{
    int a = 1, b = 2, c = 8, d = 4;
    float r = a + b * c - d;
    printf("The result is: \%f\n", r);
}
```
Operator Precedence

1. \(((1+2)\times8)-4\) = 20
2. \((1+2)\times(8-4)\) = 12
3. \((1+(2\times8))-4\) = 13

```c
#include <stdio.h>

int main()
{
    int a = 1, b = 2, c = 8, d = 4;
    float r = a + b * c - d;
    printf("The result is: %f\n", r);
}
```

```c
#include <stdio.h>

int main()
{
    int a = 1, b = 2, c = 8, d = 4;
    float r = a + b * c - d;
    printf("The result is: %f\n", r);
}
```
Operator Precedence

1. \(((1+2)*8)-4\) = 20
2. \((1+2)*(8-4)\) = 12
3. \((1+(2*8))-4\) = 13

The true answer: 13

```c
#include <stdio.h>

int main()
{
    int a = 1, b = 2, c = 8, d = 4;
    float r = a+b*c-d;
    printf("The result is: \%f\n", r);
}
```

```
C++ $g++ precedence.cc -o precedence.exe
C++ $./precedence.exe
The result is: 13.000000
C++ $
```
Operator Precedence

Operator precedence describes the order in which C evaluates expressions

- **Precedence rules:**
  1: unary +, unary -
  2: *, /, %
  3: binary +, binary -
  4: equal precedence: left to right
Operator Precedence

\[ r = -a + b \times c \quad = \quad ? \]
\[ z = a + -b / c \times d \quad = \quad ? \]
Operator Precedence: Summary

- Operators in expressions contained within pairs of **parentheses** are evaluated **first** (inner to outer)

- **Multiplication**, **division** and **remainder** operations are applied first.

- **Addition and subtraction** operations are evaluated next

- Operators with **equal precedence** such as + and –, evaluation proceeds from left to right

<table>
<thead>
<tr>
<th>Operator(s)</th>
<th>Operation(s)</th>
<th>Order of evaluation (precedence)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( )</td>
<td>Parentheses</td>
<td>Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses “on the same level” (i.e., not nested), they’re evaluated left to right.</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>Evaluated second. If there are several, they’re evaluated left to right.</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td></td>
</tr>
<tr>
<td>%</td>
<td>Remainder</td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>Addition</td>
<td>Evaluated last. If there are several, they’re evaluated left to right.</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td></td>
</tr>
</tbody>
</table>

[Image: C hot to program, Deitel and Deitel, 2010]
Operator Precedence

• Suggestion:
  Use extra parentheses to enhance readability

  \[ i + \frac{-j}{y} \cdot x \]

  \[ i + \frac{(-j)}{y} \cdot x \]
Let’s Practice

\[
\begin{align*}
\text{a + b + c + d + e} & \quad \square \quad \square \quad \square \quad \square \\
\text{a + b \times c - d \div e} & \quad \square \quad \square \quad \square \quad \square \\
\text{a \div (b + c) + d \mod e} & \quad \square \quad \square \quad \square \quad \square \\
\text{a \div (b \times (c + (d - e))))} & \quad \square \quad \square \quad \square \quad \square \\
\end{align*}
\]

\[
\begin{align*}
\text{a + b + c + d + e} & \quad 1 \quad 2 \quad 3 \quad 4 \\
\text{a + b \times c - d \div e} & \quad 3 \quad 1 \quad 4 \quad 2 \\
\text{a \div (b + c) + d \mod e} & \quad 2 \quad 1 \quad 4 \quad 3 \\
\text{a \div (b \times (c + (d - e))))} & \quad 4 \quad 3 \quad 2 \quad 1 \\
\end{align*}
\]
Assignment Operators