# Arithmetic Expressions

COP 3223C – Introduction to Programming with C

Fall 2025

Yancy Vance Paredes, PhD

# Arithmetic Expressions /1

- Recall your Algebra class?

- Solutions that you will write will involve some mathematical expressions

# Arithmetic Expressions /2

| Symbol | Operation |
|---|---|
| + | Addition |
| – | Subtract |
| * | Multiplication |
| / | Division |
| % | Modular Division (modulus; *remainder*) |

# Arithmetic Expressions /3

- Some courses would abbreviate modulus as **mod**

- Works for **whole numbers** only

- **Example:**

```
5 % 2 = ?
```

# Arithmetic Expressions /4

- So, what is the result of

$$\underbrace{3 * 2}_{6} + \underbrace{3 * 3}_{9} = 15$$

- How do you determine the order when to perform them?

- Important to remember the **order of precedence** of operators

# Arithmetic Expressions /5

- What if we really want to perform the **addition first**?

```
3 * 2 + 3 * 3


3 * (2 + 3) * 3
```

- Using **parentheses** can **override** the precedence

# Your Turn!

Assume the variables have already been declared. Rewrite each of the following formulas as valid C statements:

- Area of Rectangle = $l \cdot w$ l * w
- Area of Square = $s^2$ s * s
- Area of Triangle = $\frac{1}{2} b \cdot h$ 0.5 * b * h

~~1/2 * b * h~~

# Operators and Operands

The arithmetic operators we saw are examples of **binary operators**

Meaning, it requires two **operands** for it to work

# Evaluate the following

int long

4bytes 8bytes ✓

1 + 1

2.5 * 0.1 double

int double

3 + 4.5 7.5

10.5 + 2.5

int int

1 / 2 = 0 ✓ int 0.5

4.5 % 2

5 / 2 2 ✓

int + int = int

int double

3 + 4.5

3.0 + 4.5 double

double double 8bytes

4 bytes

# Notes

- Binary operators expect their operands have the **same data type**

- What if the data types are not the same?

# Type Casting /1

**Converting** a value from one data type into another data type

10.~~5~~

| | | | |
|---|---|---|---|
| double | 8 bytes | 10.5 | 10.0 |
| int | 4 bytes | 10 | 10 |
| char | 1 byte | | |

# Type Casting /2

**Implicit Type Casting**

Automatically done by the compiler

Usually, from **Smaller to Larger** data type

`char` < `int` < `float` < `double`

***Read as*:** `int` is greater than `char`, so on

Which scenarios do we see it?

# Type Casting /3

**Explicit Type Casting**

Manually done by the programmer

```
double num = 5.5;

int val = (int) num;

5
```

```
int val = (int) 5.5;
```

# Type Casting /4

The following is an example of an explicit* type casting

```
int n;
n = (int)(9 * 0.5);
```

(4.5)

$9.0 * 0.5 = \boxed{4.5}$

$\underline{(int)9} * 0.5$

What is the value of `n`?

4

# Type Casting /5

The following is an explicit* type casting

```
double n;
n = (int)(9 * 0.5);
```

4

What is the value of n?

4.0

# Type Casting /6

When doing an explicit type casting **from larger to smaller** data type, **we lose data**

Example:

```
    double n = 1.25 + 0.25;   = 1.50
    int m = (int)n;
                    1.5
              1
```

What is the value of `m`?

# Discussion /1

A typical pattern that you may encounter is when you want to perform **integer division**

For example, in the future you will be dealing with arrays that involves you knowing an **index**

This index is always a **whole number**

# Discussion /2

One common task that you may encounter would be to compute the index of the middle element of an array if there are 10 elements

# Notes

- Important to know that sometimes, a program’s behavior (or output) may be different from our expectation

- In short, **expectation != reality**

- There is an **error**!

# Types of Errors

- Compile Time Error (Compiler Error)
    - Syntax error!

- Run Time Error
    - It compiled but the program crashed while running!

- Logical Error
    - It compiled and ran, but **does not** perform the task correctly
    - The **most annoying** to have

# Revisiting

Assume the variables have already been declared. Rewrite each of the following formulas as valid C statements:

implicit type casting solution

1.0/2   1/2.0   1.0/2.0   0.5

- Area of Rectangle = $l \cdot w$
- Area of Square = $s^2$

double area = 1/2 * base * height.

- Area of Triangle = $\frac{1}{2} b \cdot h$

(double)1/2 * b * h

# Your Turn! /1

Write a program that **accepts a whole number** `N`. It then prints the multiplication table of `N` from 1 to 10.

# Your Turn! /2

```
Enter Number: 5
1 * 5 = 5
2 * 5 = 10
3 * 5 = 15
4 * 5 = 20
5 * 5 = 25
6 * 5 = 30
7 * 5 = 35
8 * 5 = 40
9 * 5 = 45
10 * 5 = 50
```

# Discussion

- There are several shortcuts you can use as you write C programs
- For example, a common pattern involves “accumulating values”
- Another pattern is “to increment a variable by 1”

# Shortcuts /1

A common pattern would involve updating the value of an existing variable, such as:

```
int result = 0;
result = result + expr;
```

```
result += expr;
```

# Shortcuts /2

For example:

```
int sum = 0;
sum = sum + 5;
```

0 +5

5

Is the same as:

```
sum += 5;
```

+ − * / %

val = val * num;

val *= num;

# Shortcuts /3

sum -= num* 4

For example:

```
    int sum = 0, num = 5;
    sum = sum + (num * 4);
```

Is the same as:

```
    sum += (num * 4);
```

sum = sum + num * 4

# Shortcuts /4

- It also works for other arithmetic operations, these are called **compound assignment operators**
- For example, to perform subtraction, simply change it to **`-=`**
- Be careful with the grouping of the expression
- Use parentheses as much as possible

# Shortcuts /5

Another common pattern is incrementing or decrementing the value of a variable by 1

```
int num = 0, val = 10;
num = num + 1;
val = val - 1;

num++; val--;
```

10

⟺ num++;

postfix

# Notes

When using the increment or decrement shortcuts, there are two versions that you need to be aware of: **prefix** and **postfix**

```
int res1, res2;
int num = 5;

res1 = num++;
res2 = ++num;
```

5

res1 = 5

num = 6

num = 7

7

# Your Turn! /1

Trace the following code and determine the value of `result`

```
int num = 10;
int result = ++num + 10;
```

# Your Turn! /2

Trace the following code and determine the value of `result`

```
int num = 10;
int result = ++num + 10;
result = result + num++;
```

result = 21   num = 11

21   11

32

result = 32, num = 12

# Your Turn! /3

Trace the following code and determine the value of `result`

```
int num = 10;
int result = ++num + 10;
result = result + num++;
result = result + result / result - 5 * (2 - 4);
```

result = 32

32 + 1 - 5 * -2

32 + 1 - -10

33 - -10 = 43

# Notes

- Prefix (before; update first) vs. Postfix (after; update later)

- If you write a prefix increment or a postfix increment as a standalone statement, both will simply increase the value of the variable by 1, and there’s no observable difference

- The difference only shows up when the result of the expression is used in a larger context

# Order of Precedence

| Priority | Category | Examples |
|---|---|---|
| 1 (high) | Postfix | `expr++, expr--` |
| 2 | Prefix (Unary) | `++expr, --expr, +expr, -expr, &var` |
| 3 | Multiplicative | `*, /, %` |
| 4 | Additive | `+, -` |
| 5 | Assignment | `=, +=, -=, *=, /=, %=` |
| 6 (low) | Comma | `,` |

# Code Tracing

```
#include <stdio.h>

int main(void) {
    int a = 2, b = 3, c = 4, d;

    d = a++ + --b * 5 - ++c / 2 + a * b % c;

    printf("a = %d, b = %d, c = %d, d = %d\n", a, b, c, d);

    return 0;
}
```

# Notes

It is possible to override the precedence by using `(` `)` parentheses just as you would to it in math

# Questions?