• Code examples might have an accompanying link
  • Follow link to step through the example at http://www.pythontutor.com
  • Does a nice job of graphically showing variables in memory, the heap, and the stack

• You will also find examples in the course Lectures/C and Labs/C directory, for your use

• Request other examples, or, create your own
Basic Expressions
An *expression* is anything that evaluates to a value\(^1\).

- Literals are, of course, expressions.
- Variables which store values are expressions.
- Some (very few) examples:
  - `float`
  - `int`
  - `String`
  - `Color`
  - `Array`
  - `Person` (an aggregate, record type)
  - `Function` (depending on the language)

\(^1\) Even if it’s `void`
Function Calls
A function call is also an expression

Functions are either provided in a library, or defined by the user

Consider, e.g.:

```c
double foo( int, double );
```

- Takes 2 arguments, an int and a double
- Returns a double

Can be used in a larger expression\(^1\)

```c
int main() {
    int i = 13;
    double d = 23.372,
    answer ;

    answer = 5 * foo(i, d);

    return 0;
}
```\(^1\)

\(^1\)Not the void ones
Arguments to a Function

- *Arguments* are the actual values, data, passed into a function
  - Inside the function they are stored in the *formal parameters*
- Arguments can be simple literals, or larger expressions
- C guarantees that all arguments to a function are evaluated before the function is applied

Mind the side-effects!

Most languages (and C) say nothing about the order in which the arguments are evaluated
Operators
Operators

• An operator is a symbol which tells the compiler to perform a specific operation
• They are built-in, rather than user-defined¹
• Are often considered syntax
• They can be loosely categorised:
  • Arithmetic/string operations
  • Relational
  • Logical
• There are operators which aren’t so easily categorised. Importantly:
  • The assignment operator(s): =  +=  -=  *=  &=  . . .
  • Parentheses (brackets) – Used to force evaluation of any operator
  • Member and pointer operators: *  &  .  ->

¹Though some languages allow for overloading
Properties of Operators

- All operators have three properties. In no particular order:
  - **Arity** The number of operands operated upon
  - **Precedence** If two distinct operators are adjacent, who is evaluated first?
  - **Associativity** If two of the same operator bracket an operand, which is evaluated first?
Arithmetic Operators

- **PEMDAS (BODMAS)**

---

**Don’t Listen to Facebook “Experts”!**

2 + 3 \times 5 = 17 \textit{always}! No, it doesn’t “depend on the context”

- The four basic arithmetic operators associate left-to-right
- Often + is defined in a very natural way on strings
  - If the operands are known to be strings, e.g., we know that + indicates concatenation
  - While we might find operations for / or – on strings, they are not apparent
- It is better to avoid using operators for non-intuitive operations
Overloading Operators

Re-using Operators

Consider: \( A + B \)

- C, Python, etc., know the types of the operands, choose the appropriate operation.
- If one operand is a float type and the other is an integer, C will coerce the integer to the float’s type.
- In an untyped (or, loosely typed) language, if \( A \), e.g., is a number, and \( B \) is a string, the operation is ambiguous.
- Bash, Perl and others have separate arithmetic and relational operators for numbers and strings.
Operator Precedence

• If 2 operators are adjacent, the one with higher precedence is evaluated first
  • Parentheses are used to force evaluation. Compare:

\[ 2 + 3 \times 5 \text{ vs. } (2 + 3) \times 5 \]

• C has, I dunno, at least 40 operators
• I don’t expect you to memorise the precedence of all of them (I don’t)
• Know the arithmetic ones
  • Modulus (\%) has same precedence as division
  • Note, two operators might have equal precedence
• For others, use parentheses, even if you know the precedence
  • We prefer legible code to clever
• Maybe remember that logical AND, &&, is higher than logical OR, ||
Some observations:

- Parentheses are amongst the operators with the highest precedence.
- The assignment operators have some of the lowest. They happen last.
  - But for the increment, `++`, and decrement, `--`, operators, which have the highest (and 2nd highest).
- Array index, `[ ]`, and member access, `.`, `->` have the highest precedence.

Interview Tip:

The comma, `,`, has the very lowest precedence in C.
Associativity of Operators

- **Associativity** is either left-to-right, or right-to-left, for a given operator.
- Associativity applies when two operators *with the same precedence* are adjacent.
- It doesn’t matter for operators that are associative:
  \[(2 \times 3) \times 5 = 2 \times (3 \times 5)\]
- But consider these:
  \[8 - 3 - 2 \neq 8 - (3 - 2)\]
  \[8 - 3 + 2 \neq 8 - (3 + 2)\]
  \[8 - 3 - 2 \neq 8 - (3 - 2)\]
  \[30 \div 3 \times 5 \neq 30 \div (3 \times 5)\]
- The four basic arithmetic operators associate left-to-right.
• Assignment associates right-to-left
• Consider the statement:
  \[ x = y = z = 7 \]
• When associativity is applied, it groups this way:
  \[ ( x = ( y = ( z = 7 ))) \]
• 7 is assigned into \( z \), and that expression evaluates to 7
• We then assign that value into \( y \), and the expression evaluates to 7 again
• Finally we assign 7 into \( x \), the expression evaluates to 7, which we ignore, this time
• In Python this expression makes sense:
  \( l < x < h \)
  - True if \( x \) is greater than \( l \) and less than \( h \)
  - Python really reads the above as
    \( l < x \) and \( x < h \)
  - There is no associativity
• C reads this literally
  - All relationals associate left-to-right
  - This becomes
    \( (l < x) < h \)
  - If \( l < x \), then that expression evaluates to 1, which is compared to \( h \)
  - If \( l \not< x \), then that expression evaluates to 0, which is compared to \( h \)
Evaluating Operands

Be careful applying associativity

- C only guarantees that operands will be evaluated before operator is applied
- C says nothing about the order in which operands are evaluate
  - Math doesn’t care, either!
  - There are professors, even as close as Ramapo, who get this wrong
- Many of us learned to evaluate expressions left-to-right
  - This simply describes associativity of the four basic operations
  - It’s not really true. Just a handy aid
  - Exponentiation, e.g., associates right-to-left:

\[ 2^{3^2} \neq (2^3)^2 \]

\(^a\)There are two exceptions, we’ll see shortly
Example

- Consider the expression $7 \times 5 - 12 \div 3$
  - It doesn’t matter if the multiplication happens first, or the division
  - You should have 31, whichever way you went
  - Precedence dictates that both must happen before the subtraction
  - They are not adjacent, so associativity doesn’t apply
- The compiler will evaluate whichever term is most convenient first
- In the absence of side-effects, the answer is the same
Logical AND and OR

- Operands are evaluated left-to-right
- C applies *short-circuit logic*
- Consider $A \land B$
  - If $A$ is false, entire expression is false
  - No need to evaluate $B$
  - Handy to test for divide-by-zero:
    $X \neq 0 \land y/x > 12$
- Consider $A \lor B$
  - Similarly, if $A$ is true, entire expression is true
  - No need to evaluate $B$
Side-Effects in Expressions

Remember, after precedence and associativity are obeyed, the order of operations is irrelevant to the value if there are no side-effects (if we have referential transparency).

Mind the Side-Effects!

Do not write expressions such as:

\[ i = 4; \]
\[ j = ++i + i--; \]

- Tough on the eyeballs
- **Not well-defined!**
- \( j \) could be 10, or it could be 8
- Parentheses won’t fix this
  - Since precedence is not the problem here
  - Nor is associativity!
The Ternary If Expression, ?: 

\[ \text{cond} \ ? \ \text{cons} : \ \text{alt} \]

- An inline if statement
- The only other operator whose operands are evaluated specially:
  - The \textit{condition} is evaluated
  - If true, then the \textit{consequence} is evaluated
  - Otherwise, the \textit{alternate} is evaluated
- C’s only ternary operator
- Can be nested
  - Don’t. Switch over to regular if statements, for legibility

\[ \text{max} = (a>b) \ ? \ a : b \ ; \]