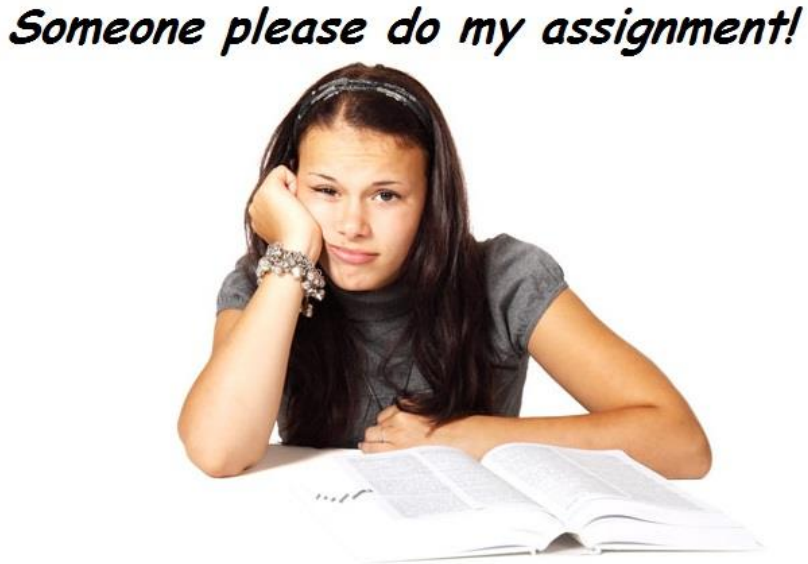
Assignment Statement

Changing a Variable Value



Anatomy of an Assignment Statement

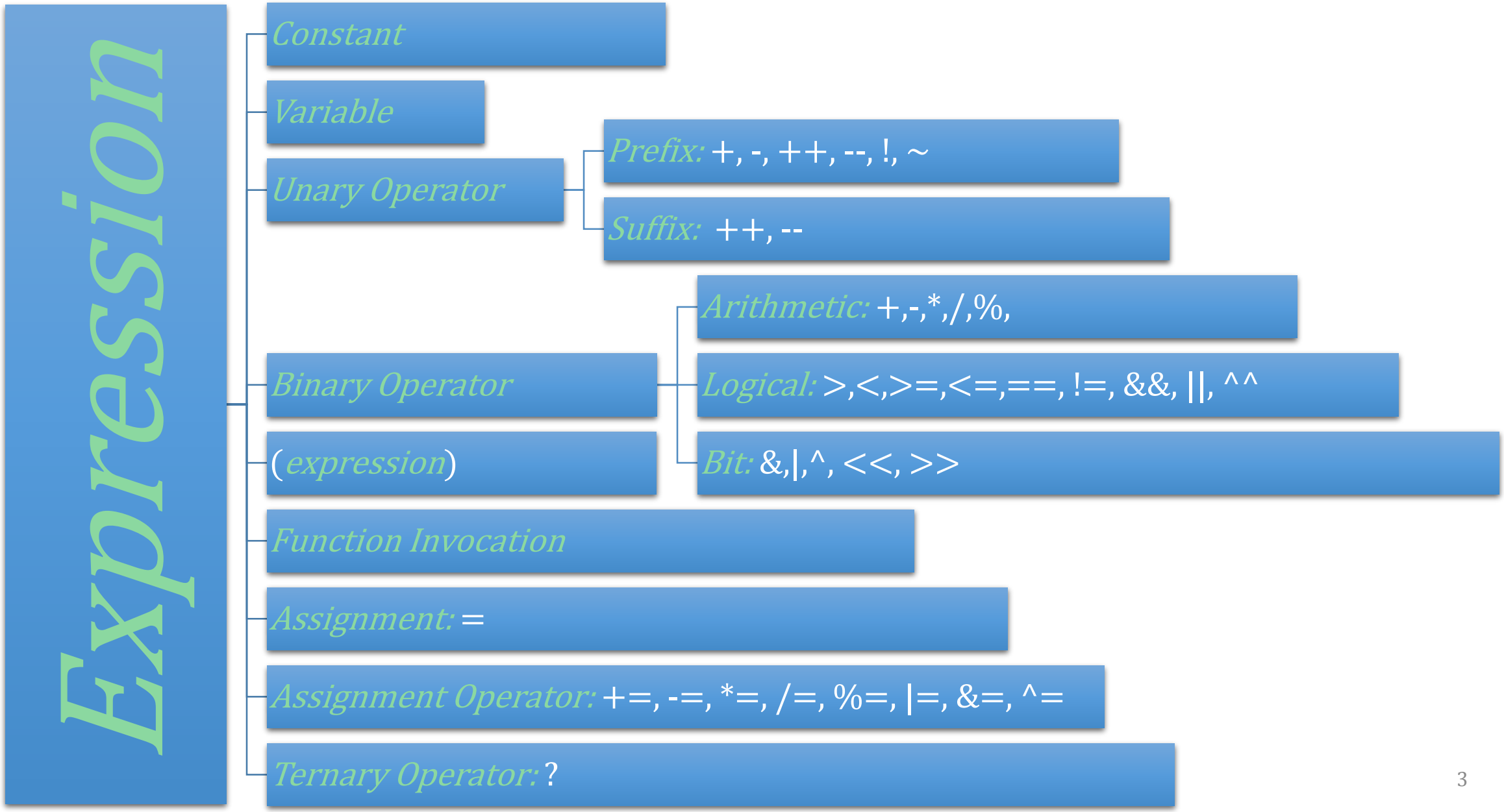
<lhs> = <rhs>;

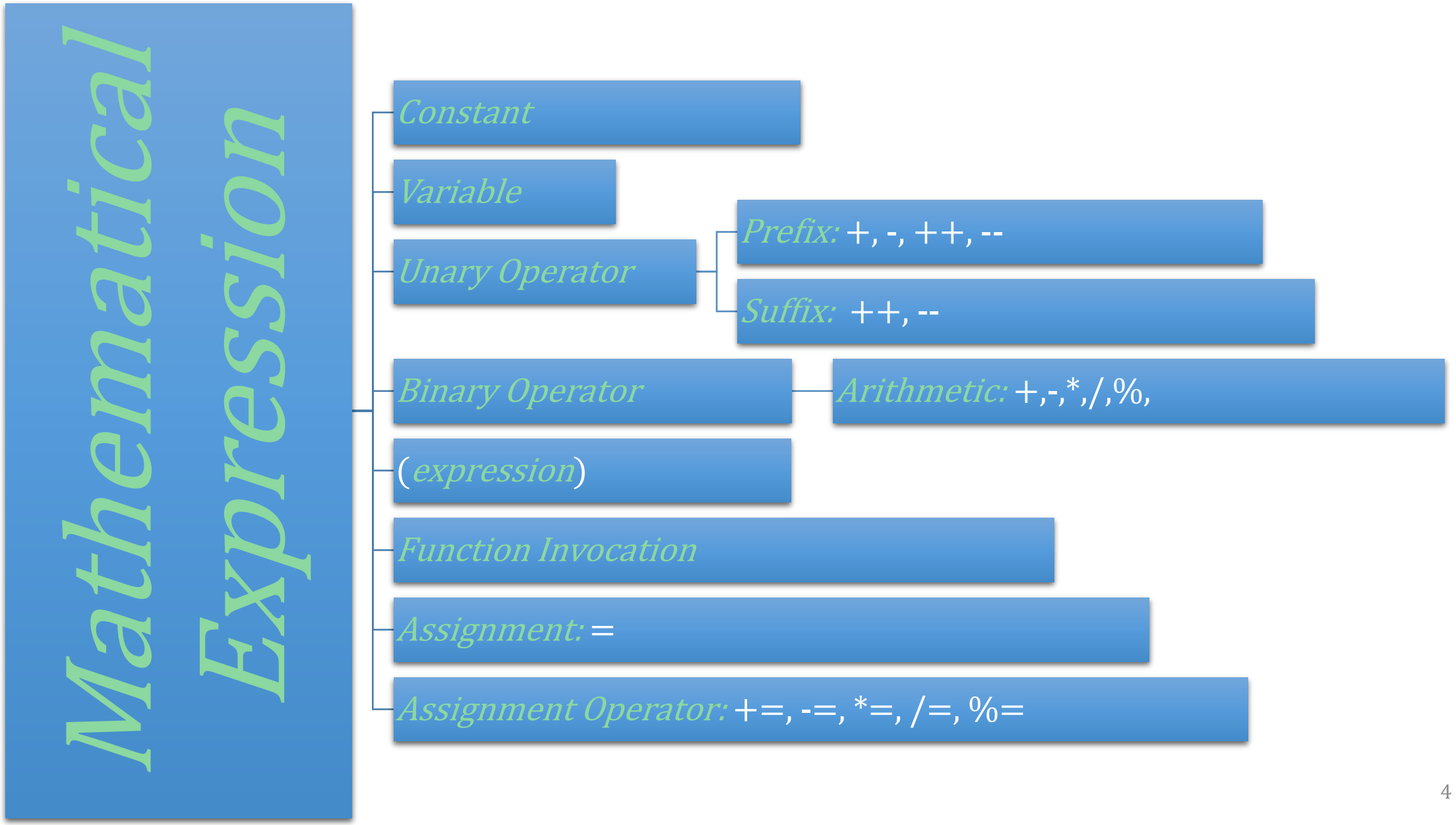
<lhs> : Left-Hand-Side – reference to memory (variable name)

(we will learn other ways to reference memory)

<rhs> : C Expression

Expression is evaluated to a value, and the memory (variable) at <lhs> is updated with that value.


Expression
Constant
Variable
Unary Operator
Prefix: +, -, ++, --, !, ~
Suffix: ++, --
Binary Operator
Arithmetic: +,-,*,/,%,
Logical: >,<,>=,<=,==, !=, &&, ||, ^^
Bit: &,|,^, <<, >>
(expression)
Function Invocation
Assignment: =
Assignment Operator: +=, -=, *=, /=, %=, |=, &=, ^=
Ternary Operator: ?

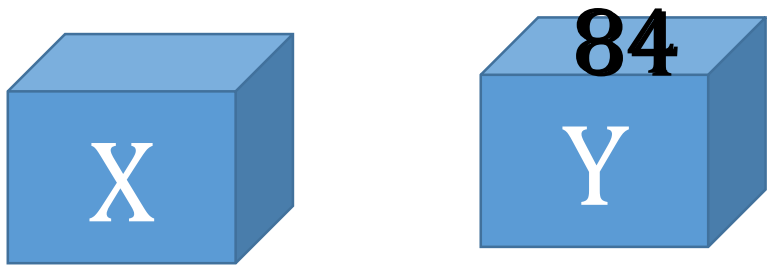

Mathematical Expression
Constant
Variable
Unary Operator
Prefix: +, -, ++, --
Suffix: ++, --
Binary Operator
Arithmetic: +,-,*,/,%,
(expression)
Function Invocation
Assignment: =
Assignment Operator: +=, -=, *=, /=, %=

Example Constant Assignments

```
int x;
float y;
char first_init;

x=13;
y=7.3;
first_init='T';
```

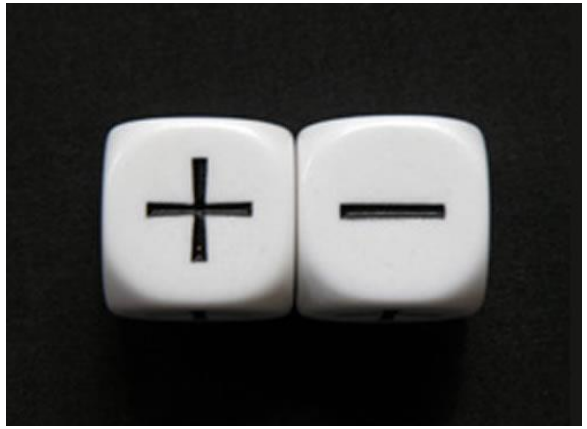
Example Variable Assignment

```
int y=84; int x;
float fx=7.3; float fy;
char first_init;


x=y;
fy=fx;
first_init=y;
```



Example Unary Operators

```
int x; int y=5;


x=-y;
x=++y;
x=y--;
y=+120;
```

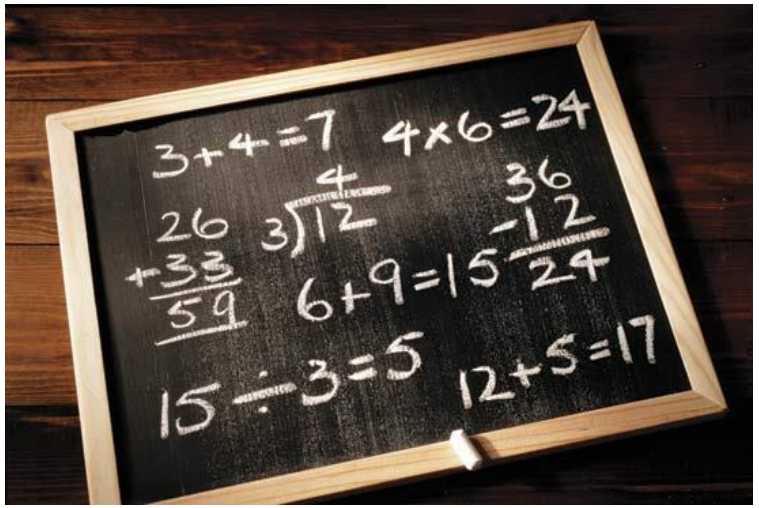


Prefix vs. Suffix Increment / Decrement

```
int x=3; int y; int z;

y=++x; /* y=4, x=4 */

z=x++; /* z=4, x=5 */
```

```
int x=3; int y; int z;

x=x+1; /* x=4 */
y=x;   /* y=4 */

z=x;   /* z=4 */
x=x+1; /* x=5 */
```

Example Mathematical Binary Operators




```
int x; int y; float fx;


x = 32 * 3;  /* x=96 */
y = x / 5; /* y = 19 */
x = x % 5; /* x = 1 */
fx = x / 3.7; /* fx = 0.2702702702703 */
```

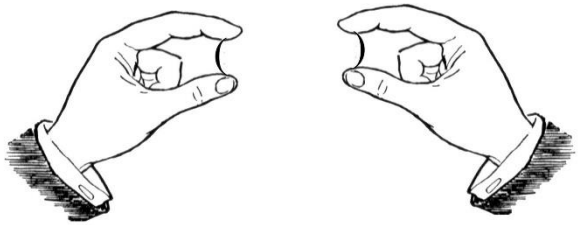
Operator Precedence

- Binary expressions are in the form <expr> <op> <expr>
- So, for instance, 3+5*7 is a valid expression
  - Should this be evaluated as 3+5=8, 8*7=56?
  - Should this be evaluated as 5*7=35, 3+35=38?

- Rules in C: Operator Precedence
  - Always do multiplication/division/modulo first
  - Then do addition/subtraction

Parenthesis

- Evaluate sub-expression in parenthesis first
  - e.g. (3+5)*7 is evaluated 3+5=8, 8*7=56
- Parenthesis can be nested
  - e.g. ((3+5)*(2+2)) is evaluated 3+5=8, 2+2=4, 8*4=32
- If you're not sure, use parenthesis
  - Extra parenthesis don't change the answer, 3+(5*7) = 36
  - Missing parenthesis may result in the "wrong" answer, 3+5*7 = 36

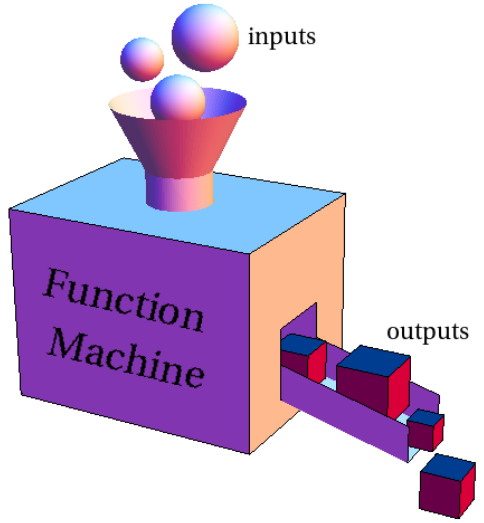


Function Invocation

- Expression of the form <fn_name>(<expr1>,<exp2>, …)
  - <fn_name> : Name of a function previously declared
  - <expr1>,<exp2>, … : One expression for each argument

- Argument expressions are evaluated (left to right)
  - Expression values are copied to function argument "variables"
- Function is invoked
- The value of the function invocation is the value returned by the function

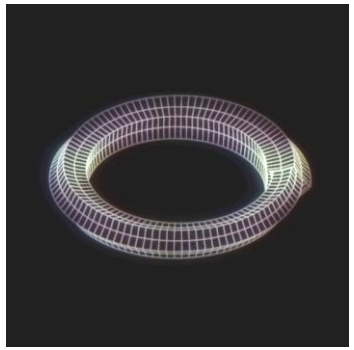
Example Function Invocations




```
int add(int x, int y) { return x+y; }
int sub(int x, int y) { return x-y; }

int main() {
        int a=5; int b=7; int c;
        c=sub(add(a,b),sub(b,a));
}
```

```
c=sub(add(5,7),sub(7,5))
        add(x=5,y=7)=>12
        sub(x=7,y=5)=>2
c=sub(12,2)
        sub(x=12,y=2)=>10
c=10
```

Assignment Expressions



- The value of an assignment is the value of the Left Hand Side

```
int x; int y; int z;
x=y=z=0;

if ( (x=x+1)>0 ) then printf("incremented x is positive");
```

Assignment Operators

- Assignment of the form: <LHS><op>=<RHS>;
  - <LHS> : Memory Reference (variable) as in assignment
  - <op> : Binary operator such as +, -, /, *, %, …
  - <RHS> : Expression
- Shorthand for <LHS> = <LHS> <op> <RHS>;

```
int x=6;
x +=2; /*x=8*/
x /=3; /*x=2*/
x*=5; /*x=10*/
```

```
int x=6;
x = x+2; /*x=8*/
x = x / 3; /*x=2*/
x = x*5; /*x=10*/
```

Resources

- Programming in C, Chapter 3
- WikiPedia: Operators in C and C++ (https://en.wikipedia.org/wiki/Operators_in_C_and_C%2B%2B)
- GNU C Tutorial, Expressions and Operators (http://www.crasseux.com/books/ctutorial/Expressions-and-operators.html#Expressions%20and%20operators)