Operators

Outline

1. Operators
2. Arithmetic Operators
3. Operator Precedence and Associativity
4. Expressions
5. Different Forms of Assignment Operators
6. Increment and Decrement Operators
7. Swapping Values between Two Variables

1. Operators

- *Operator* – a symbol or keyword that represents an operation to be applied to some data, *yielding a value*.

  E.g. `varA = -varB + 40 * 20;`

- *Operand* – input data to an operator

- We use operators all the time in real life. You should all be familiar with Binary operators and Unary operators in basic arithmetic. You can recognize them intuitively in C:
  - *Binary operator* – an operator that accepts 2 operands

    E.g. `40 * 20`   `x - 7.3`   `a = 5`
  - *Unary operator* – an operator that accepts 1 operand

    E.g. `-10`   `+10`

1. Operators

- Basic arithmetic operators in C:
  - e.g., +   -   *   /   %
- There are other types of operators you will learn later:
  - Relational Operators, e.g., <   <=   ==   >=   >   !=
  - Logical Operators, e.g., && || !
  - Assignment Operators, e.g., =   +=   *=   &=
  - Increment and Decrement Operators, e.g., ++   - -
  - Bitwise Operators, e.g., & | !   ^
  - Comma Operator, Parentheses, Conditional Operator, Member Operator, Pointer Operators, …
  - Most Binary, some Unary and even Ternary

2. Arithmetic Operators

Operator	Description	Example
+	Addition	`8 + 5 ☾ 13`
-	Subtraction (a *binary* operator)	`8 - 5 ☾ 3`
*	Multiplication	`8 * 5 ☾ 40`

/
/
	Integer Division
	Floating-point Division
		`8 / 5 ☾ 1`
		(*Quotient*)
		`8.0 / 5.0 ☾ 1.6`

%	Modulus (yields the remainder of an integer division) *Applicable only to integers*	`8 % 5 ☾ 3` (*Remainder of 8 / 5*)

-
+
	Minus (an *unary* operator)
	Plus (an *unary* operator for *integer promotion*)
		`- (5+7) ☾ -12`
		`+ (-7) ☾ -7`

Don't type or write ×

2. Arithmetic Operators

- When used as an unary operator, '`-`' represents the *minus* operator, which turns a positive value into its negative counterpart and vice versa, i.e., additive inverse.

e.g.:

```
foo = 5;
 bar = -foo;  // Assign -5 to bar
```

Exercise: evaluate the following expressions

- `20 % 3`
- `2 % 9`
- `30 / 20 / 2`
- `10 * 2 + 4 * 3`

3. Operator Precedence & Associativity

- How should we evaluate the following expression? i.e., in what order should the operators be applied?

```
- 2 - 25 / 10 + 33 * 2
```

- Among different operators, *operator precedence* tells us which operator(s) should be applied first.
- Most of us will know immediately that * and / are applied before + and -
- Your C program will also respect the common sense arithmetic precedence

3. Operator Precedence & Associativity

- Using basic arithmetic knowledge, what is the result of the following expression?

```
- 2 - 25 / 10 + 33 % 10 * 2
```

- **Hint:** If operators have the same precedence, which one do we evaluate first, as a human?

  e.g.: `3 * 5 * 2`

- This is called *operator associativity*; among operators with the same precedence, *operator associativity* tells us whether the left-most or the right-most operator should be applied first.

3. Operator Precedence & Associativity

- Given the limited amount of operators we have learned in C so far, this table summarizes their operator precedence and operator associativity

Operators	Associativity	Precedence
+ (unary plus) - (unary minus)	Right to Left	Highest
* / %	Left to right	
+ (addition) - (subtraction)	Left to right	
= (assignment)	Right to Left	Lowest

- Operators at same level have same precedence.

  e.g.: `- a * b - c` is equivalent to `((- a) * b) – c`

- `-2 - 25 / 10 + 33 % 10 * 2` = ?

3.1. Parentheses

- Use parentheses '**(**' and '**)**' to explicitly specify the evaluation order of sub-expressions

```
(a + b) * (c + d)
```

- Multiple levels of parentheses (never use [ ] or { }!)

```
((a + b) * (a + b) - c) * (d - e)
```

- **Tips:** Use parentheses for clarity or when you are not sure about the precedence of the operators. We shall learn more about precedence when we learn other operators.

4. Expressions

- An *expression* is a combination of operators, constants, variables, and function calls
  - e.g.:

```
30
24 + a
d = b * b - 4 * a * c
sqrt(4.0) + a * sqrt(9.0)
```

- An expression
  - Can always be evaluated to a value (of some data type)
  - Can be part of another expression

5. Assignment Operators

```
variable = expression
```

- Low precedence, *right-to-left* associativity
- `expression` is evaluated first and the evaluated value is assigned to `variable`.
- ***Important:*** "`variable = expression`" is also an expression which evaluates to the value of `variable`.

  e.g.:

  ```
  var1 = var2 = 3 + 2
  ```

  is evaluated as

  ```
  var1 = (var2 = (3 + 2))
  ```

```c
int a = 0, b = 2, c;
double pi = 3.1416;

```

Equivalent to

```c
int a, b, c;
double pi;
a = 0;
b = 2;
pi = 3.1416;
```

Assignment operator can be used to initialize variables in variable declaration.

What's the value of variable **c**?

```c
int a = 0;
a = a + 2;
printf("%d", a);                 // What's the output?

```

+ has higher precedence than **=**. Thus

`a = a + 2`

is evaluated as

`a = (a + 2) Ⓟ a = (0 + 2) Ⓟ a = 2`

```
1   int a = 1, b = 2;
2   b = b * a;
3   a = 0;
4   printf("%d", b);      // What's the output?
```

Statements are executed sequentially one after another.

(Line 1) a is set to 1 and b is set to 2.

(Line 2) b becomes 2.

(Line 3) a becomes 0 but changing a ***does not affect other variables***.

```
int b, c, d;
d = c = b = 0;      // Assign 0 to variables b, c, and
d
// d = c = b = 0 is evaluated as d = (c = (b = 0))
```

5.1. Assignment Operators – Short Form

- `i = i + 2;` can be written as `i += 2;`

- The semantics of

  ```
  variable = variable op (expression);
  ```

  is equivalent to

  ```
  variable op= expression;
  ```

- Some short form assignment operators:

  ```
  +=    -=    *=    /=    %=
  ```

- Note that `i *= j + 2;` is equivalent to `i = i * (j + 2);` and not to `i = i * j + 2;`

6. Increment / Decrement Operator

- To increase the value of a variable, **`i`**, by one, we can write the following statement:

```
i = i + 1;
```

- We can also write a statement with an **increment operator** to achieve the same result:

  `i++;` or **`++i;`**

  (see Appendix for their difference)

- Similarly, we can write **`i--`** or **`--i`** to decrease the value of **`i`** by one.

7. Swapping the value between two variables

```
int a = 0, b = 1, tmp;
// How to exchange/swap the value of a and b?
```

```
  a = b;          // Method A ?
  b = a;
```

```
  tmp = b;          // Method B ?
  b = a;
  a = tmp;
```

```
  tmp = b;          // Method C ?
  a = tmp;
  b = a;
```

Answer: Method B

Dry run the code segments and you'll know why!

Summary

- Arithmetic operators (**+**, **-**, *****, **/**, **%**)
- Operator **precedence** and **associativity**
- Different forms of assignment operators (**=**, **+=**, **-=**, ***=**, …)
- Increment (++) and decrement (--) operators
- Swapping the value between two variables

Appendix (Optional Topics)

- Difference between `++i` (prefix) and `i++` (postfix) Increment operators

- Practical uses of Integer Division (`/`) and Modulus (`%`) operators

More on Increment Operator

- The increment operator (**++**) can be placed in either **prefix** or **postfix** position, with different results.

- **`++i`** (prefix increment to **`i`**)
  - Increase the value of **`i`** by 1, FIRST before everything in this line.
  - The value of the expression "**`++i`**" is the NEW value of **`i`**.

- **`i++`** (postfix increment to **`i`**)
  - The value of the expression "**`i++`**" is the OLD value of **`i`**.
  - Increase the value of **`i`** by 1, LAST after everything in this line.

More on Increment Operator

Statement that involves ++ operator	Equivalent statements

k = ++i * 2;
// prefix increment of i
	i = i + 1; // side effect first
	k = i * 2; // NEW value of i*2

k = i++ * 2;
// postfix increment of i
	k = i * 2; // OLD value of i*2
	i = i + 1; // side effect last

printf("%d\n", ++k);
// prefix increment of k
	k = k + 1; // side effect first
	printf("%d\n", k); // NEW k

printf("%d\n", k++);
// postfix increment of k
	printf("%d\n", k); // OLD k
	k = k + 1; // side effect last

More on Increment Operator

- Example

```c
int i, k;
i = 0;
k = ++i;
printf("%d\n", i);
printf("%d\n", k);
printf("%d\n", ++k);
```

```
1
1
2
```

```c
int i, k;
i = 0;
k = i++;
printf("%d\n", i);
printf("%d\n", k);
printf("%d\n", k++);
```

```
1
0
0
```

Some uses of Integer Division and Modulus Operators

Suppose **n** is an integer

- `(n % 10)` yields the right most digit of **n**

  e.g.: `1234 % 10 Ⓟ 4`

- (n / 100 % 10) yields the 3rd digit from the right of n

  e.g.: `1234 / 100 % 10 Ⓟ 12 % 10 Ⓟ 2`

- Determining if **n** is odd or even

  if **n** is odd, **`(n % 2)`** shall be 1 or -1 (i.e., not zero)

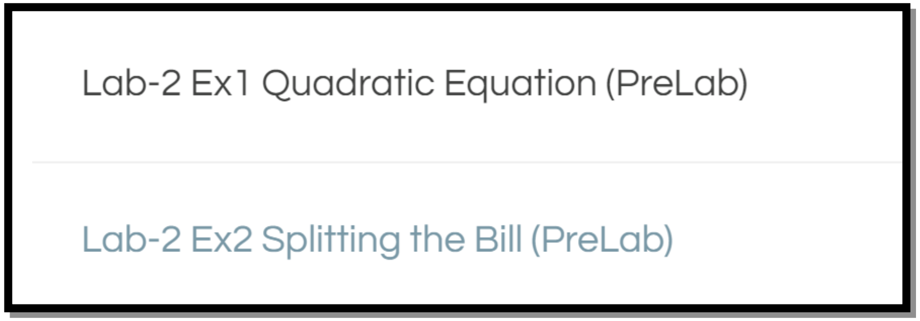
  if **n** is even, **`(n % 2)`** shall be 0

Reading Assignment

- C: How to Program, 8th ed, Deitel and Deitel
- Chapter 2 Introduction to C Programming
  - Section 2.5
- Chapter 3 Structured Program Development in C
  - Sections 3.11, 3.12

Reminder: PreLabs are Ready!

- Every Mon afternoon we will release the **PreLabs**
  - Meant to help you prepare for the lab
  - Due **Wed 9:30am** – Please try it after the lecture and submit before Wed!
  - Don't worry – it's super easy (takes < 30 min) and it's very easy marks to get! Don't forget!



Lab-2 Ex1 Quadratic Equation (PreLab)

Lab-2 Ex2 Splitting the Bill (PreLab)

PreLabs are marked "(PreLab)" on repl.it