

Computer Science & Engineering 150A

Problem Solving Using Computers

Mini Lecture – Short Circuiting

Christopher M. Bourke

Spring 2009

- C logical AND and OR operators: `&&` and `||`
- Logical `false` `&& A` is false *regardless* of what `A` is
- Logical `true` `|| B` is true *regardless* of what `B` is
- C *short-circuits* these operators

- Short-circuiting: if it is not necessary to evaluate the right-hand-side of a logical expression, then the right-hand-side is *not* evaluated
- Each side of the operator is a separate, independent *expression*:
[ExpressionA] && [ExpressionB]
[ExpressionA] || [ExpressionB]
- The usual *order of precedence* is applicable only *inside* each expression.

Short Circuiting

Example

CSCE150A

Introduction

Short
CircuitingMultiple
Operators

```
1 int a = 10;  
2 int b = 0;  
3 int c = 1;
```

- Consider: `c || a * b`

Short Circuiting

Example

CSCE150A

Introduction

Short
CircuitingMultiple
Operators

```
1 int a = 10;  
2 int b = 0;  
3 int c = 1;
```

- Consider: `c || a * b`
- `c` is true, thus the entire expression is true

Short Circuiting

Example

CSCE150A

Introduction

Short
CircuitingMultiple
Operators

```
1 int a = 10;  
2 int b = 0;  
3 int c = 1;
```

- Consider: `c || a * b`
- `c` is true, thus the entire expression is true
- `a * b` does not need to be evaluated

Short Circuiting

Example

CSCE150A

Introduction

Short
CircuitingMultiple
Operators

```
1 int a = 10;  
2 int b = 0;  
3 int c = 1;
```

- Consider: `c || a * b`
- `c` is true, thus the entire expression is true
- `a * b` does not need to be evaluated
- Thus, it is not, `a * b` is short-circuited

Short Circuiting

Example

CSCE150A

Introduction

Short
CircuitingMultiple
Operators

```
1 int a = 10;  
2 int b = 0;  
3 int c = 1;
```

- Consider: `b && a + c`

Short Circuiting

Example

CSCE150A

Introduction

Short
CircuitingMultiple
Operators

```
1 int a = 10;  
2 int b = 0;  
3 int c = 1;
```

- Consider: `b && a + c`
- `b` is false, thus the entire expression is false

Short Circuiting

Example

CSCE150A

Introduction

Short
CircuitingMultiple
Operators

```
1 int a = 10;  
2 int b = 0;  
3 int c = 1;
```

- Consider: `b && a + c`
- `b` is false, thus the entire expression is false
- `a + c` does not need to be evaluated

Short Circuiting

Example

CSCE150A

Introduction

Short
CircuitingMultiple
Operators

```
1 int a = 10;  
2 int b = 0;  
3 int c = 1;
```

- Consider: `b && a + c`
- `b` is false, thus the entire expression is false
- `a + c` does not need to be evaluated
- Thus, it is not, `a + c` is short-circuited

- Sometimes we have multiple operators:

`A && B || C`

`A || B && C`

- Ideally: *use parentheses* to make them unambiguous:

`A && (B || C)` or `(A && B) || C`

`A || (B && C)` or `(A || B) && C`

- Without parentheses how are these evaluated?

- Recall that `&&` has a higher order of precedence than `||`
- In `A && B || C`:
 - 1 First, `A && B` is evaluated
 - 2 This requires that first `A` be evaluated
 - 3 Then (*if necessary*) evaluate `B`
 - 4 Call the result `R`
 - 5 Then (*if necessary*) `R || C` gets evaluated.

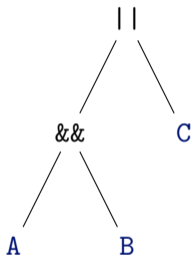


Figure: Evaluation Tree of the Expression `A && B || C`

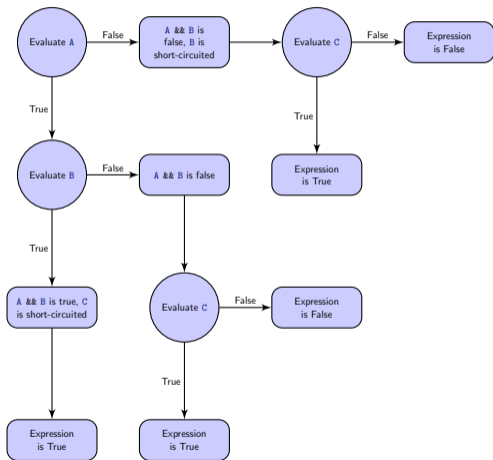


Figure: Evaluation of the Expression $A \ \&\& \ B \ || \ C$

- In $A \ || \ B \ \&\& \ C$:
 - 1 The $\&\&$ has higher precedence.
 - 2 But, the evaluation tree is different

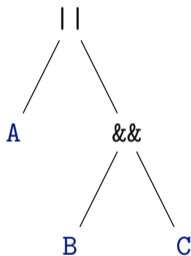


Figure: Evaluation Tree of the Expression `A || B && C`

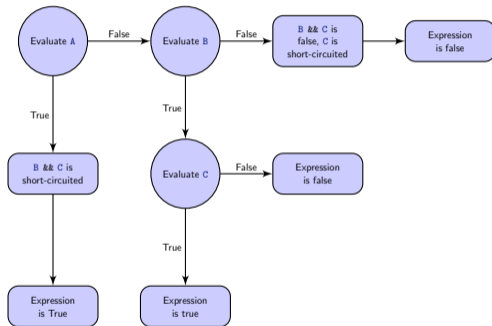


Figure: Evaluation of the Expression $A \ || \ B \ \&\& \ C$

Truth Table 1

CSCE150A

Introduction

Short
CircuitingMultiple
Operators

A	B	C	A && B C	A && (B C)
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

Table: C Truth Table

A	B	C	$A \ \ B \ \&\& \ C$	$A \ \ (B \ \&\& \ C)$
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

Table: C Truth Table

What do you take away from this?

- Short-circuiting can get very complicated even with simple expressions
- Evaluating requires building an evaluation tree, *then* doing an *in-order* traversal
- Honestly: this is an advanced concept that requires graph theory, graph traversal algorithms, etc.
- If future questions appear, they will be *simple*
- Follow the rule: **Always use parentheses**