CS 262 Lecture 4: Operators Part 2, Control Flow

GEORGE MASON UNIVERSITY

Overview of Lecture 4

Operators

- Relational operators
- Logical operators

Control Flow – Conditional Expressions

- if statements
- else-if statements
- switch statements

Control Flow - Loops

- for loops
- while loops
- do-while loops

Control Flow – Break, Continue

Relational Operators

Relational Operators

Let you perform a comparison between two operators with **Boolean Algebra**

Recall from last class: 0 is **false**, anything else is **true**

A relational operator will always evaluate to either 0 or 1

Operator	Description	Example	Result
==	Equal to	`5 == 3`	`0` (False)
!=	Not equal to	`5 != 3`	`1` (True)
<	Less than	`5 < 2`	`0` (False)
>	Greater than	`5 > 2`	`1` (True)
<=	Less than or equal to	`5 <= 2`	`0` (False)
>=	Greater than or equal to	`5 >= 2`	`1` (True)

Logical Operators

Logical Operators

Let you test multiple expressions using Boolean logic

Just like relational operators, these will always evaluate to either 0 or 1

Operator	Description	Example	Result
`&&`	Logical And	`(1 > 5) && (8 > 3)`	`0` (False)
`||`	Logical Or	`(1 > 5) || (8 > 3)`	`1` (True)
`!`	Logical Not	`!(1 == 5)`	`1` (True)

These also work with several expressions

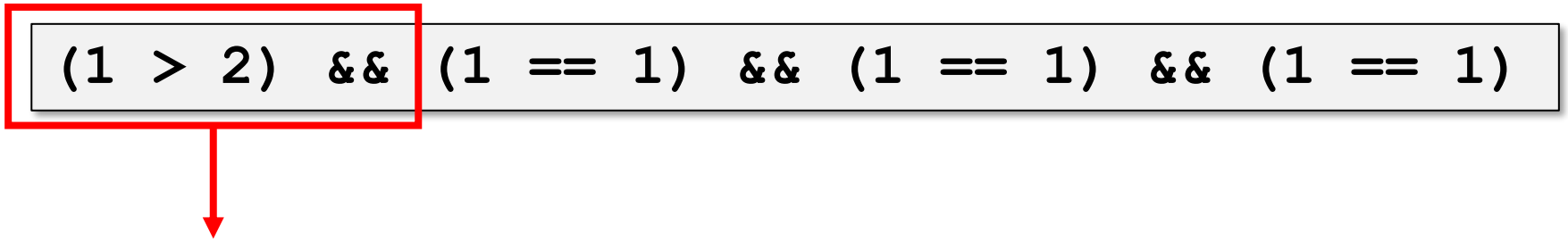
```
(1 > 4) || (2 > 4) || (3 > 4) || (4 > 4) || (5 > 4)
```

Logical Operators

Special Rules/Considerations

```
false && anything == false
true || anything == true
```

If the expression is known to be true or false **before** it finishes evaluating the entire thing, execution stops



We already know the entire expression is false after this

if Statements

if Statement

Same as you remember from other languages

```
if(expression) {
      // Execute statement1 if expression is true
      statement1;
}
else {
      // Execute statement2 if expression is false
      statement2;
}
```

Nested if Statements

Nested if Statements

```
if(expression1) {
  // expression1 is true
  if(expression2) {
    // only executed if both expression1, expression2 are true
}
  else {
    // expression1 true, expression2 false
  }
}
else {
  // expression1 false
}
```

else-if statements

else-if statements

```
if(expression1) {
  // expression1 is true
}
else if(expression2) {
  // expression1 is false, but expression2 is true
}
else {
  // both are false
}
```

Switch Statements

switch Statements

A more efficient, but more limited version of if-else-if

```
switch(expression or variable) {
  case 1: printf("This is case 1\n");
          break;
}
  case 2: printf("This is case 2\n");
          break;
}
  case 3: printf("This is case 3\n");
          break;
}
  default: printf("No cases occurred\n");
          break;
}
```

Switch Statements

switch Statements – Rules and Behavior

Only one branch will match

Case values must be int, char, short, or long

default is required, and is invoked if no cases match

Each case ends with `break;` which exits the statement when a branch matches

```
switch(expression or variable) {
  case 1: printf("This is case 1\n");
          break;
}
  case 2: printf("This is case 2\n");
          break;
}
  case 3: printf("This is case 3\n");
          break;
}
  default: printf("No cases occurred\n");
          break;
}
```

Switch Statements vs if-else

switch Statements vs if-else – When Do We Use Them?

Feature	if-else	switch
Data type	Any	Only int types
Conditions	Can use any comparison	Only checks for equality (==)
Execution flow	Sequential evaluation	Jumps directly to matching case
Readability	Good for complex conditions	Cleaner for multiple discrete values
Performance	Slower due to sequential checking	Can be faster due to jump table (367 stuff)
Fall-through Behavior	Greater than or equal to	Executes cases sequentially unless break is used

Loops

for Loops

Ideal for counting, when we know how many iterations to execute ahead of time

while Loops

Checks if a condition is true before running

do-while Loops

Always runs at least once before checking the condition

for Loop

for Loops

Perfect for when you know how many times you need to execute something

```
for(int i=0; i<5; i++) {
 printf("i is currently %d\n", i);
}
```

We can also nest for loops

```
for(int i=0; i<5; i++) {
  for(int j=0; j<i; j++) {
    printf("i = %d, j = %d\n", i, j);
  }
}
```

note: variables declared when a loop is initialized do not exist outside of the loop

while Loop

while Loops

Checks if a condition is met, then executes the loop if it is

```
char buffer[20];
int number = 0;

printf("Enter a number (0 to exit)\n");
fgets(buffer, sizeof(buffer), stdin);
sscanf(buffer, "%d", &number);

while(number !=0) {
  printf("you entered %d\nEnter another number\n", number);
  fgets(buffer, sizeof(buffer), stdin);
  sscanf(buffer, "%d", &number);
}
printf("Exiting\n");
```

do-while Loop

do-while Loop

Similar to the while loop, except this executes prior to checking the condition

Using do-while loop

```
int count = 0;
do {
  printf("count = %d\n", count);
  count++;
}
while (count < 10);
```

Using while loop

```
int count = 0;
while (count < 10) {
  printf("count = %d\n", count);
  count++;
}
```

Will these behave the same if we change the condition to `count < 0`

break, continue

break exits the current loop or switch statement

If you are in the inner-loop of a nested loop, it only exits the inner loop

continue jumps to the condition check of the current loop

Good Programming Practice

Use break and continue sparingly

Overusing these can easily create spaghetti code

Code can often be refactored with if conditions or later, function returns instead

The right loop for the right task

Use `for` loops when the iteration count is known

Use `do-while` loops only when execution must happen once (like user input)

Naming conventions

Use consistent and descriptive naming for variables

Even though `i`, `j` aren't descriptive, it is customary to use these as iterators in for loops

Loops – Practice

Let's Practice

Pay close attention – These resemble what you will see on quizzes and exams

```
// loop 1
int count = 0;
do {
  printf("count = %d\n", count);
  count++;
}
while (count < 0);
```

```
// loop 2
int count = 0;
while (count < 0) {
  printf("count = %d\n", count);
  count++;
}
```

Loop 1 prints _____ time(s)

Loop 2 prints _____ time(s)