Conditionals and Loops

Steven R. Bagley

Operator	Function	Example
!	Negate (unary)	`!(a > 3)`
>	Greater Than	`a > 3`
>=	Greater than or equal	`b >= c()`
<	Less than	`a < (3+5)`
<=	Less than or equal	`b <= c`
==	Equal	`x == 3`
!=	Not equal	`x != y`

Background colour denotes precedence groups, Highest precedence is at the top, lowest at the bottom

True or False

- Comparisons operators are true if the relation is met
  - `3<5`, `6==6`, `7!=6` are all true
  - `3>5`, `6!=6`, `7==6` are all false
- C does not have a boolean type
- Represents false as `0`, and true as any other integer

Go and show this with printf

Program Flow

- Programs usual flow from statement to statement
- The power of programming is when the programmer is in control
- Calling Functions allow us to reuse code
- But we can also do things *conditionally*

Conditional Execution

- Only execute some code *if* a particular condition is true
- What do we mean by condition?
- Anything that can be converted into a true or false value
- Any of the operators we've just seen!

Conditionals

- The condition can be based on
  - Values (not that useful on their own)
  - Value of a variable
  - Value returned by a function

The `if` statement

- C's `if` statement is used to express a decision
- If the condition is true, then execute the next statement
- If the condition is false, then don't execute the next statement

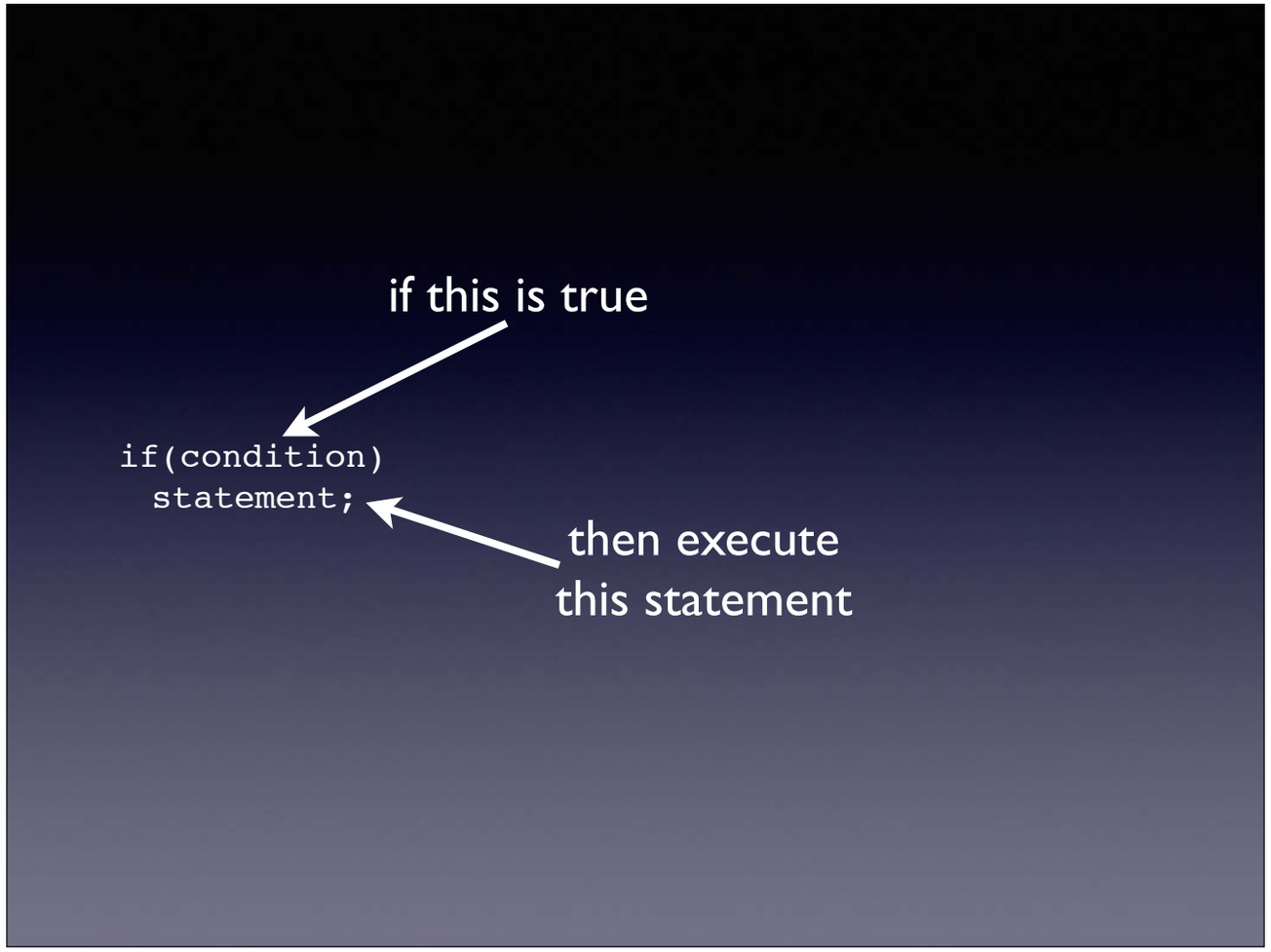
```
if(condition)
  statement;
```

the statement is executed only if condition is true

if this is true

```
if(condition)
  statement;
```

the statement is executed only if condition is true



the statement is executed only if condition is true

```
x = 3;

if(x < 5)
  printf("Hello World");
```

the statement is executed only if condition is true

if this is true

```
x = 3;

if(x < 5)
  printf("Hello World");
```

the statement is executed only if condition is true

if this is true

```
x = 3;

if(x < 5)
  printf("Hello World");
```

then execute
this statement

the statement is executed only if condition is true

```
x = 5;

if(x < 5)
  printf("Hello World");
```

the statement is executed only if condition is true

if this is true

```
x = 5;

if(x < 5)
  printf("Hello World");
```

the statement is executed only if condition is true

if this is true

```
x = 5;

if(x < 5)
  printf("Hello World");
```

then execute
this statement

the statement is executed only if condition is true

```
x = 5;

if(x = 1)
  printf("Hello World");
```

we're not testing equality here -- we are setting x to equal 1, the result of that expression is 1 which is interpreted as true
you want...

if this is true

```
x = 5;

if(x = 1)
  printf("Hello World");
```

we're not testing equality here -- we are setting x to equal 1, the result of that expression is 1 which is interpreted as true
you want...

if this is true

```
x = 5;

if(x = 1)
  printf("Hello World");
```

then execute
this statement

we're not testing equality here -- we are setting x to equal 1, the result of that expression is 1 which is interpreted as true
you want...

if this is true

```
x = 5;

if(x == 1)
  printf("Hello World");
```

then execute
this statement

here x does not equal 1 so it doesn't print

Keyboard

- Can use `if` to test if a key is pressed on a keyboard
- Use `getchar()` function to read key
  `int getchar()`
- Returns the character code of the key pressed
- Or -1 if end of file (hence, returns an `int`)

Press ENTER

- Slight problem…
- UNIX only returns keyboard input when a carriage return is pressed
- So we have to also press RETURN before we see our keypress

Character literals

- `getchar()` returns the ASCII character code
- Fortunately, C provides us with a little trick so we don't have to remember them
- Put the character in single quotes, e.g. `'A'`
- Compiler interprets it as a character literal
- And places the right value into the program

In this case, the number 65 (for 'A')
DEMO!

Blocks

- Executing only one statement is limiting
- We can execute a block of statements by putting that block in `{ … }`
- Like we do for functions
- All the statements in the block then executed if the condition is true
- Often sensible to include `{ }` anyway

```
if(condition)
{
  statement;
  statement;
  statement;
  statement;
}
```

the statements are executed only if condition is true
Modify program to demo

if this is true

```
if(condition)
{
  statement;
  statement;
  statement;
  statement;
}
```

the statements are executed only if condition is true
Modify program to demo

if this is true

```
if(condition)
{
  statement;
  statement;
  statement;
  statement;
}
```

then execute all these statements

the statements are executed only if condition is true
Modify program to demo

if true do this, else do that

- What happens if we want to do one thing if the condition is true
- But something else if it is false
- Could use a second if, but not always possible
- C provides an `else` clause

```
if(condition)
{
  statement;
  statement;
  statement;
  statement;
}
else
{
  statement;
  statement;
  statement;
  statement;
}
```

Demo

if this is true

```
if(condition)
{
  statement;
  statement;
  statement;
  statement;
}
else
{
  statement;
  statement;
  statement;
  statement;
}
```

Demo

if this is true

```
if(condition)
{
  statement;
  statement;
  statement;
  statement;
}
else
{
  statement;
  statement;
  statement;
  statement;
}
```

then execute all these statements

Demo

if this is true

```
if(condition)
{
  statement;
  statement;
  statement;
  statement;
}
else
{
  statement;
  statement;
  statement;
  statement;
}
```

then execute all these statements

else execute all these statements

Demo

if else if else if

- Can chain `if…else` statements
- If this then, else if something else then, else…
- E.g. for handling menu choices
- Can also combine in the `if` part too
- But be careful…

```
int c = getchar();

if(c == ‘a’)
  printf(“You pressed a\n”);
else if(c == ‘b’)
  printf(“You pressed b\n”);
else if(c == ‘c’)
  printf(“You pressed d\n”);
else
  printf(“You pressed something else\n”);
```

Demo
Have to store the value from getchar() or it would mean something else…

```
int c = getchar();
int x = 14;

if(x > 10)
  if(c == 'a')
     printf("Hello World\n");
```

Makes sense if x is greater than 10 and c is a then print Hello world

```
int c = getchar();
int x = 5;

if(x > 10)
  if(c == ‘a’)
     printf(“Hello World\n”);
  else
     printf(“Goodbye Universe\n”);
```

What happens if I press b?

Does the else associate with if(x>10) or if(c == ‘a’)???

```
int c = getchar();  'B'
int x = 5;

if(x > 10)
  if(c == 'a')
     printf("Hello World\n");
  else
     printf("Goodbye Universe\n");
```

What happens if I press b?

Does the else associate with if(x>10) or if(c == 'a')???

Association

- Previous code is ambiguous since `else` is optional
- C removes the ambiguity by saying `else` associates with the closest previous `else`-less `if`
- Use braces to form a block if you want the opposite

```
int c = getchar();
int x = 14;

if(x > 10)
  if(c == 'a')
     printf("Hello World\n");
  else
     printf("Goodbye Universe\n");
```

```
int c = getchar();
int x = 14;

if(x > 10)
{
  if(c == 'a')
     printf("Hello World\n");
  else
     printf("Goodbye Universe\n");
}
```

These two are equivalent...

```
int c = getchar();
int x = 14;

if(x > 10)
{
  if(c == 'a')
     printf("Hello World\n");
}
else
  printf("Goodbye Universe\n");
```

This is the opposite form...

Combining Conditionals

- Sometimes we might want to combine conditionals
- If both these things are true…
- If either of these things are true…
- Already seen one (messy) way to do this
- Can also lead to duplicated code

duplicated code is bad

```
int c = getchar();
int x = 14;

if(x > 10)
{
  if(c == ‘a’)
     printf(“Hello World\n”);
}
```

printf Hello World only printed if both x >10 and c == ‘a’

```
int c = getchar();
int x = 14;

if(x > 10)
   printf("Hello World\n");

if(c == 'a')
   printf("Hello World\n");
```

Or is harder -- if both true, then HEllo World would be printed twice...
Need to make a note if we execute this

```
int c = getchar();
int x = 14;

if(x > 10)
   printf("Hello World\n");

if(c == 'a')
   printf("Hello World\n");
```

This will not work

Or is harder -- if both true, then HEllo World would be printed twice...
Need to make a note if we execute this

```
int c = getchar();
int x = 14;

if(x > 10)
{
   printf("Hello World\n");
}
else if(c == 'a')
      printf("Hello World\n");
```

For 'Or', we can use an else...

```
int c = getchar();
int x = 14;

if(x > 10)
{
   printf("Hello World\n");
}
else if(c == 'a')
      printf("Hello World\n");
```

This will work

For 'Or', we can use an else...

Combining Conditionals

- Easier way…
- C lets us use boolean algebra to combine conditionals
- AND and OR and NOT
- More operators

Operator	Function	Meaning
!	Negate (unary)	NOT
&&	AND	Both must be true
||	OR	Either, can be true (or both)

In precedence order...

Precedence is lower than the other conditionals

Lazy Evaluation

- C will evaluate the left-hand side
- And then (if necessary) the right-hand side
- This is called lazy-evaluation
- Remember your truth-tables from G51CSA
- It means that a function may not get called
- Or a variable accessed

This is incredibly useful!!!

A	B	Result
FALSE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

```
A && B
```

If A is false, then the result is always false, so no need to evaluate right-hand side

A	B	Result
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

```
A || B
```

If A is true, then the result is always true, so no need to evaluate right-hand side

```
int c = getchar();
int x = 14;

if(x > 10 && c == ‘a’)
{
     printf(“Hello World\n”);
}
```

printf Hello World only printed if both x >10 and c == ‘a’
if x is less than ten then c is never tested
Can cause interesting side-effects if you call a function…

Program Flow

- Functions allow us to divert into another block of code (possibly several times)
- `if` statements let us decide whether a block of code gets executed or not
- Repeat a block of code several times
- The *loop*

Loops

- Execute a series of statements repeatedly
- Until some condition is met
- C provides three different type of loops
- `while`, `do {...} while`, `for`

Why loop?

- Do things a set number of times
- Step up over a set of data values
- Calculate a particular value
- Wait until some input is received
- Most interactive programs spend a lot of times in loops

(e.g. using Newton-Raphson to calculate a square root, or the HCF algorithm dave showed you)

Loops — The nasty way

- Mimic what the CPU is actually going to do
- Use the `goto` instruction
- Tells C to go to a specific point in the program
- Do not use this!
- Ever…

goto Demo...

`while` loop

- The `while` loop is probably the simplest
- Has two parts
  - A block of statements to execute
  - A conditional
- While the conditional evaluates to true
- Execute the statements

```
while(condition)
{
  statement;
  statement;
  statement;
  statement;
}
```

Demo
Does this look familiar?

while this is true

```
while(condition)
{
  statement;
  statement;
  statement;
  statement;
}
```

Demo
Does this look familiar?

while this is true

```
while(condition)
{
  statement;
  statement;
  statement;
  statement;
}
```

then execute these statements repeatedly

Demo
Does this look familiar?

How `while` works

- First, the condition is evaluated
- If false, the block of statements is skipped
- If true, all the statements are executed
- Then condition tested again
- Statements can alter the value of variables
- So change whether the condition is true

Go demo make a program that calculates a range of temperatures