Week 3 Part 2

Kyle Dewey

Overview

- Lab announcement
- `if` / `else`
- `switch`
- Project 1
- Exam 1 Review

`turnin` **lab by Friday**

at 11:59

Conditionals

Motivation

```
int min( int x, int y ) {
   // return smaller of
   // x and y
}

int max( int x, int y ) {
   // return larger of
   // x and y
}
```

Min / Max

- Need a way to say "pick this one **if** it's larger, **else** pick the other one"
- `if` / `else` does exactly this

if

```
if ( condition )
  itIsTrue();

if ( condition ) {
  itIsTrue();
}

if ( condition ) {
  itIsTrue();
  somethingElse();
}
```

Condition

- The condition is a **Boolean expression**
- These can use relational operators

```
int x = 0;
int y = 1;

x < y
0 <= y
1 > 5
2 >= x
```

Equality

- Not equal: != (`x  != y`)
- Equality: == (NOT =)

Examples

```
if ( 1 < 2 ) {
   printf( “foo” );
}

if ( 2 == 3) {
   printf( “foo” );
}
```

Question

- What does this print?

```
int x = 0;

if ( x = 1 ) {
  printf( “foobar” );
}
```

Answer

- “foobar”
- Assignment instead of equality

```
int x = 0;

if ( x = 1 ) {
  printf( “foobar” );
}
```

if / else

```
if ( condition ) {
  itIsTrue();
} else {
  itIsFalse();
}
```

Examples

```
if ( 1 < 2 ) {
  printf( “foo” );
} else {
  printf( “bar” );
}

if ( 2 < 1 ) {
  printf( “baz” );
} else {
  printf( “moo” );
}
```

if / else if / else

- Picks the first one that's true

```
if ( condition ) {
   itIsTrue();
} else if ( otherCondition ){
   otherIsTrue();
} else {
   neitherAreTrue();
}
```

Min / Max Revisited

```
int min( int x, int y ) {
  // return smaller of
  // x and y
}

int max( int x, int y ) {
  // return larger of
  // x and y
}
```

Example #1

```
if ( 2 < 2 ) {
   printf( “foo” );
} else if ( 2 < 3 ) {
   printf( “bar” );
} else {
   printf( “baz” );
}
```

Example #2

```
if ( 2 < 3 ) {
   printf( “foo” );
} else if ( 3 < 4 ) {
   printf( “bar” );
} else {
   printf( “baz” );
}
```

Nesting

- Conditionals can be nested
- Heavy nesting usually means it should be split into more functions

Nesting Example

```
if ( 1 < 2 ) {
   if ( 5 >= 4 ) {
      printf( “fizz” );
   } else {
      printf( “buzz” );
   }
} else if ( 2 < 3 ) {
   printf( “bar” );
} else {
   printf( “baz” );
}
```

Boolean Operations

- And: & &
- Or: | |

Examples

```
if ( 1 < 2 && 2 > 3 )
if ( 1 == 2 || 2 >= 2 )
if ( 1 > 2 && 2 < 3 )
if ( ( 1 > 2 || 2 < 3 ) && 1 == 1 )
```

Precedences

- And (`&&`) has higher precedence than or (`||`)

```
if ( 1 < 2 && 2 > 3 || 5 == 5 )
if ( ( 1 < 2 && 2 > 3 ) || 5 == 5 )
```

More on Conditions

- In C, 0 (zero) is considered false, and everything else is considered true
- This refers **only** to the binary representation

Examples

```
if ( 5 )
  printf( “foo" );

if ( 0 )
  printf( “bar” );

if ( ‘0’ )
  printf( “baz” );
```

Math vs. C

- Say we have an integer variable "`x`"
- Mathematically, we can test if `0 < x < 1000`
- However, C does not behave quite like this
  - All relational operators are binary in nature

Math vs. C

```
int amount = 0;
0 < amount < 1000 // true in C (1)

( 0 < amount ) < 1000
0 < 1000
1
```

Math vs. C

```
int amount = 1;
0 < amount < 1000 // true in C (1)

( 0 < amount ) < 1000
1 < 1000
1
```

Math vs. C

```
int amount = 1000;
0 < amount < 1000 // true in C (1)

( 0 < amount ) < 1000
1 < 1000
1
```

Boolean Operators

- And (&&) and or (||) work in the same way

```
1 && 2 // returns 1
1 || 2 // returns 1
0 && 1 // returns 0
0 || 0 // returns 0
```

Short-Circuiting

- C stops looking at an expression once its truth value can be determined
- This is called **short-circuiting**

```
1 == 1 || 2 == 3 || 3 == 5
2 == 2 && 3 < 2 && 5 > 7
```

Question

- What does `x` equal after this code runs?

```
int x = 1;
if ( x > 1 && x = 0 ) {
   x = 2;
}
```

Answer

- `x = 1`

```
int x = 1;
if ( x > 1 && x = 0 ) {
   x = 2;
}
```

Question

- What does `x` equal after this code runs?

```
int x = 1;
if ( x > 0 && x = 0 ) {
   x = 2;
}
```

Answer

- x = 0

```
int x = 1;
if ( x > 0 && x = 0 ) {
   x = 2;
}
```

switch

`switch`

- Can be seen as a specialized version of `if`
- Can look nicer than `if` for select cases

switch

```
switch ( integral expression ) {
   case constant integral expression:
      statements;
      break;
   case constant integral expression:
      statements;
      break;
   default:
      statements;
}
```

Integral Expression

- Something that returns an integer

```
switch ( 5.5 ) // non-integer
```

Constant Integral

- Something that returns an integer constant

```
int x = 1;
int y = 10;

switch( y ) {
   case x: // non-constant
     ...
```

Semantics

- Find first matching case
- Execute statements until we hit a `break`
- If no cases match, execute the default

Example #1

```
int x = 5;

switch ( x ) {
   case 4:
      printf( “four” );
      break;
   case 5:
      printf( “five” );
      break;
   default:
      printf( “Do not know” );
}
```

Example #2

```
int x = 4;

switch ( x ) {
   case 4:
      printf( “four” );
      break;
   case 5:
      printf( “five” );
      break;
   default:
      printf( “Do not know” );
}
```

Example #3

```
int x = 4;

switch ( x ) {
   case 4:
      printf( “four” );
   case 5:
      printf( “five” );
      break;
   default:
      printf( “Do not know” );
}
```

Example #4

```
int x = 4;

switch ( x ) {
   case 4:
   case 5:
      printf( “five” );
      break;
   default:
      printf( “Do not know” );
}
```

Example #5

```
int x = 4;

switch ( x ) {
   case 4:
   case 5:
     printf( “five” );
   default:
     printf( “Do not know” );
}
```

Example #6

```
int x = 6;

switch ( x ) {
   case 4:
   case 5:
     printf( “five” );
     break;
   default:
     printf( “Do not know” );
}
```

Useful Example

```
int vowel = 0;

switch ( getchar() ) {
   case ‘a’:
   case ‘e’:
   case ‘i’:
   case ‘o’:
   case ‘u’:
      vowel = 1;
}
```

Useful Example

```
int input = getchar();
int vowel = 0;

if ( input == ‘a’ || input == ‘e’ ||
     input == ‘i’ || input == ‘o’ ||
     input == ‘u’ )
  vowel = 1;
```

Project 1

Exam Structure

- 75 Minutes
- 20% of grade
- Short answer style
- Questions of variable length

Don’t worry about...

- Specific ASCII character codes (`'a'` = `97`, etc.)
- Specific `scanf` / `printf` placeholders

Do worry about...

- Hint hint: I ask a lot of questions in the slides
- Understanding what code does
- Being able to write code to do certain things, perhaps with functions
- Different kinds of errors