APS105 Lecture 6
3.3-3.5

Dan Zingaro

February 1, 2010

Feedback from Reading Quiz

- I don't understand switch statements. Are they only used when we need to account for many cases?
  - Practically, yes; but it's not a rule ...

```
switch (i) {
  case 5: printf ("hi\n");
}
```

- What happens if there is no `default` in a `switch`, and there is no match?
  - The `switch` does nothing
  - It is not a compiler error; it may or may not be a logical error

Feedback from Reading Quiz...

- Can the `&&`, `||`, and `!` Boolean operators only be used in if-statements?
  - No. Just like + or <, they are operators that return values

```
int a;
int b = 0, c = 1;
a = b || c;
```

Logical Operators

- Logical operators operate on truth values to give new truth values
- Three logical operators: `!` (not), `&&` (and), `||` (or)

p	q	!p	p && q	p || q
non-zero	non-zero	0	1	1
non-zero	zero	0	0	1
zero	non-zero	1	0	1
zero	zero	1	0	0

Logical Operators...

- `!` has higher precedence than `&&`, which has higher precedence than `||`
- Let's try some examples. Assume `a` and `b` are `false`

```
!a && b
(!a) && b
!(a && b)
a && b || true
a && (b || true)
```

Short-circuit Evaluation

- When expressions contain `&&` or `||`, C stops evaluating as soon as it knows whether the whole expression is true or false
- e.g. consider expression `p && q`
- If `p` is `false`, C can stop there: it has no reason to check `q` because the whole expression must already be `false`
- e.g. consider expression `p || q`
- If `p` is `true`, C can stop there: it has no reason to check `q` because the whole expression must already be `true`

ConcepTest

Consider the following C statement.

```
printf ("hi") || printf ("bye");
```

What is the output of running this statement? (As return value, `printf` returns the number of characters printed.)

- A. No output
- B. `hi`
- C. `hibye`
- D. `bye`

Nested Ifs (grade.c)

- We can nest ifs inside of other ifs to handle more complicated branching structures

```
#include <stdlib.h>

int main (void) {
  int grade;
  scanf ("%d", &grade);
  if (grade < 0 || grade > 100)
    printf ("Invalid grade!\n");
  else
    if (grade < 50)
      printf ("Failing grade\n");
    else
      printf ("Passing grade\n");
  return 0;
}
```

Let's rewrite the code so that the final `else` is the "invalid grade" branch.

Dangling Else

```
if (p)
  if (q)
    <statement1>
  else
    <statement2>

if (p)
  if (q)
    <statement1>
else
  <statement2>
```

- These two programs have the same code, so one of the indentations must be misleading
- What does the program do?
- The question is, "with which `if` is `<statement2>` associated?"
- The rule is that each `else` is associated with the first available preceding `if`

ConcepTest

What is the output of the following code, assuming that `x` is an `int` variable with value 4?

```
if (x <= 2) {
if (x == 4)
printf ("one\n");
}
else
printf ("two\n");
```

- A. one
- B. two
- C. one two
- D. No output

Many Alternatives (grade2.c)

Here is one way to print a different message for each grade A, B, C and D.

```
#include <stdio.h>
int main (void) {
  char grade;
  scanf ("%c", &grade);
  if (grade == 'A')
    printf("Excellent");
  else if (grade == 'B')
    printf("Good");
  else if (grade == 'C')
    printf("Average");
  else if (grade == 'D')
    printf("Fair");
  else
    printf("Invalid grade");
  return 0;
}
```

Many Alternatives... (gradeswitch.c)

When we have several alternatives that each compare the same variable to a different value, we can use a switch instead.

```
#include <stdio.h>

int main (void) {
  char grade;
  scanf ("%c", &grade);
  switch (grade) {
    case 'A': printf("Excellent");
      break;
    case 'B': printf("Good");
      break;
    case 'C': printf("Average");
      break;
    case 'D': printf("Fair");
      break;
    default: printf("Invalid grade");
  }
  return 0;
}
```

Switch Statement

- C finds the first `case` label that matches, and starts executing from there
- The `break` means "exit the `switch`" statement; without it, all of the statements below the matching label would get executed
- A `switch` is not as flexible as an `if`
  - The case values must be constants; variables are not allowed. We cannot say `case x:`
  - The case values must all be compared to the same integer expression
  - There is no straightforward way to compare an expression to a range (like `<= 5`)

ConcepTest

For which integer values of `x` will the following code print `hi`?

```
switch (x - 3) {
  case 7: break;
  case 6:
  case 4: printf ("hi\n");
}
```

- A. 9 7
- B. 6 4
- C. 4
- D. 7