SWITCH STATEMENTS

CSC100 Introduction to programming in C/C++ (Spring 2025)

Marcus Birkenkrahe

March 15, 2025

Contents

README

- In this section of the course, we go beyond simple statements and turn to program flow and evaluation of logical conditions
- This section follows chapter 3 in Davenport/Vine (2015) and chapters 4 and 5 in King (2008)
- Practice workbooks, input files and PDF solution files in GitHub

Overview

- The `switch` statement is fairly complex: it combines conditional expressions, constant expressions, default and break statements.

```
switch ( expression ) {
  case constant expression : statements
  ...
  case constant expression : statements
  default : statements
}
```

- Controlling `expression`: an integer expression in parentheses, like `(5)`. Characters are treated as integers in C and cannot be tested, so `('a')` is not allowed.

- `Case` labels: each case begins with a constant expression, like `Case 5: -` this is like any other expression except that it cannot contain variables or function calls.
- **Statements:** any number of statements. No braces required around the statements. The last statement is usually `break` to close the case.

Simple example

- In the example program below, the `grade` is set in the variable declaration. Depending on the value, a `case` is triggered and the corresponding statements are executed.
- What is the output of the code below for `grade =` 5,3,0,-1,0.5?

VALUE	OUTPUT
5	Failing
3	Passing
0	Illegal grade
-1	Illegal grade
0.5	Illegal grade

- The code:

```
int grade = 0.5;
switch (grade) {
 case 4:
 case 3:
 case 2:
 case 1:
   printf("Passing");
   break;
 case 5:
 case 6:
   printf("Failing");
   break;
 default:
   printf("Illegal grade");
   break;
 }
```

```
Illegal grade
```

- What does this program do? Which problem/solution is implemented?

  Answer: The program reflects "passing" grades 4,3,2,1, and "failing" grade 5,6. Any other grade value is not allowed. (This happens to be the German grade scale, which is A = 1 to D = 4, and F = 5 or 6.)

- You can also put several case labels on the same line as shown below - the code is otherwise identical to the previous one:

```
int grade = 3;

switch (grade) {
 case 4: case 3: case 2: case 1:
   printf("Passing");
   break;
 case 5: case 6:
   printf("Failing");
   break;
 default:
   printf("Illegal grade");
   break;
 }
```

```
Passing
```

- Note: You cannot write a case label for a range of values.

- The default case (when none of the case expressions apply) is optional, and it does not have to come last.

The role of the `break` statement

- The `switch` statement is a *controlled jump*. The `case` label is a marker indicating a position within the switch.

- Let's run the previous program again, without the `break` statements. What do you think the output will be?

```
int grade = 5;

switch (grade) {
  // cases 4,3,2,1 all lead to a passing grade
 case 4:
 case 3:
 case 2:
 case 1:
   printf("Passing");
 case 5:
 case 6:
   printf("Failing");
 default:
   printf("Illegal grade");
 }

FailingIllegal grade
```

- What happens without the `break` statements?

  > Answer: When the last statement in a case has been executed, control falls through to the first statement in the following case; its case label is ignored. Without `break` (or some other jump statement, like `return` or `goto`, control flows from one case to the next.

- Deliberate falling through (omission of `break`) should be indicated with an explicit comment.

Practice Exercise: "Day of the Week Classifier"

Objective

Write a C program using a `switch` statement to classify an integer input (1-7) as a specific day of the week and print a corresponding message. This reinforces understanding of `switch`, `case`, `break`, and `default`.

Instructions for Students

1. Copy the starter pseudocode below into a `main` program in your source code editor.

2. Fill in the missing parts in C:
    - Declare and initialize the `day` variable with a value (e.g., `int day = 3;`).
    - Replace each comment with the appropriate `case` statement, `printf`, and `break`.
    - Add the `default` case.
3. Test your program with at least three values:
    - One weekday (e.g., 3)
    - One weekend day (e.g., 6)
    - One invalid value (e.g., 8)
4. (Bonus) Remove one `break` statement, predict the output, and run it to confirm.

Starter Pseudocode: onecompiler.com/c/43bxaes2k

```
#include <stdio.h>

int main() {
  // Declare an integer variable 'day' and set it to a test value (1-7)
  // e.g., int day = 3;

  // Write a switch statement to evaluate 'day'
  switch (day) {
    // Case for day 1: Print "Monday: Start of the workweek!"
    // Add break statement

    // Case for day 2: Print "Tuesday: Getting into the groove."
    // Add break statement

    // Case for day 3: Print "Wednesday: Midweek already!"
    // Add break statement

    // Case for day 4: Print "Thursday: Almost there!"
    // Add break statement

    // Case for day 5: Print "Friday: Weekend is near!"
```

```
    // Add break statement

    // Case for day 6: Print "Saturday: Time to relax!"
    // Add break statement

    // Case for day 7: Print "Sunday: Rest and recharge."
    // Add break statement

    // Default case: Print "Error: Not a valid day!"
    // Add break statement
  }

  return 0;
}
```

Expected Outputs

- `day = 3`: "Wednesday: Midweek already!"
- `day = 6`: "Saturday: Time to relax!"
- `day = 8`: "Error: Not a valid day!"
- Bonus (e.g., remove `break` after `case 5`):
  - If `day = 5`, output becomes "Friday: Weekend is near!Saturday: Time to relax!" due to fall-through.

Timing

- **Total**: 15-20 minutes
  - **3-5 minutes**: Copy and understand the pseudocode.
  - **5-8 minutes**: Fill in the C code for `day` and `switch` cases.
  - **3-5 minutes**: Test with three values.
  - **2-4 minutes** (optional): Bonus question on fall-through.