SWITCH STATEMENTS

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

Marcus Birkenkrahe

March 16, 2025

README

This lecture explores the `switch` statement in C, a powerful control structure for handling multiple conditions. You'll learn how `switch` uses an integer expression to select from `case` labels—constant values triggering specific statements. The `break` keyword controls execution by exiting the `switch` block, preventing fall-through to subsequent cases unless omitted intentionally. Through examples and a practice exercise, we'll cover its syntax, behavior, and practical applications, like grading systems and day classification.

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

- What does this program do?

```
SET hamlet = 0  // Assign a number representing Hamlet's mood

  SWITCH hamlet
      CASE 0
          OUTPUT "Not to be"
          BREAK
      CASE 1
          OUTPUT "To be or not to be, that is the question"
          BREAK
      CASE 2
          OUTPUT "Alas, poor Yorick! I knew him well"
          BREAK
      CASE 3
          OUTPUT "Something's rotten in the state of Denmark"
          BREAK
      DEFAULT
          OUTPUT "Get thee to a nunnery!"
          BREAK
  END SWITCH
```

- **Bonus exercise:** Turn this pseudocode into running C code! (Canvas: *Bonus: Hamlet, Prince of Denmark.*

Another 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?

```
SET grade = 5
```

```
SWITCH grade
    CASE 1 to 4
       OUTPUT "Passing"
       BREAK
    CASE 5 or 6
       OUTPUT "FAILING"
       BREAK
    DEFAULT
       OUTPUT "Illegal grade"
       BREAK
END SWITCH
```

- 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.)

- Source code:

```
int grade = 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;
 }

Failing
```

- All output cases:

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

- 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
```

- The `default` case (when none of the case expressions apply) is **optional**, and it does not have to come last!
- Note: **You cannot write a case label for a range of values**.
- To make this happen in C, you'd have to use a loop to cycle through a range, using a loop over the values of an array:

```
int grade;
int i; // loop variable
float grades[]={5,3,0,-1,0.5}; // array of grades
size_t length_of_grades = sizeof(grades)/sizeof(grades[0]);

for (i = 0; i < length_of_grades; i++) { // LOOP through grades
  grade = (int)grades[i];
```

```
 switch (grade) {  // SWITCH grade
 case 4:
 case 3:
 case 2:
 case 1:
   printf("%d: Passing\n", grade);
   break;
 case 5:
 case 6:
   printf("%d: Failing\n", grade);
   break;
 default:
   printf("%d: Illegal grade\n", grade);
 }  // END SWITCH
} // END LOOP
```

```
5: Failing
3: Passing
0: Illegal grade
-1: Illegal grade
0: Illegal grade
```

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"

Task

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

1. Open the starter pseudocode below in the online C 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. 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.

Sample solution

```
#include <stdio.h>

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

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

    // Case for day 2: Print "Tuesday: Getting into the groove."
    // Add break statement
  case 2: // Case for day 2
    printf("Tuesday: Getting into the groove.\n"); // Print "Tuesday: Getting into the
    break; // Add break statement

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

```
    printf("Wednesday: Midweek already!\n"); // Print "Wednesday: Midweek already!"
    break; // Add break statement

    // Case for day 4: Print "Thursday: Almost there!"
    // Add break statement
  case 4: // Case for day 4
    printf("Thursday: Almost there!\n"); // Print "Thursday: Almost there!"
    break; // Add break statement

    // Case for day 5: Print "Friday: Weekend is near!"
    // Add break statement
  case 5: // Case for day 5
    printf("Friday: Weekend is near!\n"); // Print "Friday: Weekend is near!"
    break; // Add break statement

    // Case for day 6: Print "Saturday: Time to relax!"
    // Add break statement
  case 6: // Case for day 6
    printf("Saturday: Time to relax!\n"); // Print "Saturday: Time to relax!"
    break; // Add break statement

    // Case for day 7: Print "Sunday: Rest and recharge."
    // Add break statement
  case 7: // Case for day 7
    printf("Sunday: Rest and recharge.\n"); // Print "Sunday: Rest and recharge."
    break; // Add break statement

    // Default case: Print "Error: Not a valid day!"
    // Add break statement
  default: // Default case
    printf("Error: Not a valid day!\n"); // Print "Error: Not a valid day!"
    break; // Add break statement
  }
  return 0;
}
```

Summary

- **Structure and Usage**: The `switch` statement evaluates an integer expression against constant `case` labels, executing associated statements, with `break` typically used to exit and an optional `default` for unmatched cases.
- **Break's Role**: Without `break`, execution falls through to subsequent cases, ignoring their labels, which can be intentional but should be commented; with `break`, control exits after a case's statements.
- **Limitations and Flexibility**: `case` labels must be integer constants (no ranges or variables), and multiple cases can share statements (e.g., stacking or inline), as seen in grading or day-of-week examples.

References

- Davenport/Vine (2015) C Programming for the Absolute Beginner (3ed). Cengage Learning.
- Grok 3 by xAI.
- Kernighan/Ritchie (1978). The C Programming Language (1st). Prentice Hall.
- King (2008). C Programming - A modern approach (2e). W A Norton.
- Orgmode.org (n.d.). 16 Working with Source Code [website]. URL: orgmode.org