Queens College, CUNY, Department of Computer Science

Object Oriented Programming in C++

CSCI 211 / 611

Summer 2018

Instructor: Dr. Sateesh Mane

September 10, 2018

Enum and switch

- In this lecture we shall study the term "**enum**" (which is short for "enumerated type").
- We shall also study the **switch** statement.
- The switch statement is frequently used together with enum values.

1 Enum

- In C++ we can define symbolic names for integer valued constants.
- They are called “enumerated types” and denoted by the term **enum**.
- Let us consider the days of the week.

```
enum { SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY };
```

- Typically enum values are written in uppercase, although it is not required.
- The enum values obey the following rules.
  1. The values are **integer constants.**
  2. The first entry is assigned the value 0, the rest increase in steps of 1.
  3. Hence in the above example `SUNDAY=0, MONDAY=1, ..., SATURDAY=6`.
  4. However, we can assign any values we wish, including negative numbers.

     ```
     enum { SUNDAY=1, MONDAY, TUESDAY=100, WEDNESDAY, THURSDAY=-5, FRIDAY, SATURDAY };
     ```

     In this case, `SUNDAY=1, TUESDAY=100, THURSDAY=-5` and the other values increment in steps of 1, i.e. the full set is `SUNDAY=1, MONDAY=2, TUESDAY=100, WEDNESDAY=101, THURSDAY=-5, FRIDAY=-4, SATURDAY=-3`. However, such a usage of enums is rare.
- The use of an enum enables us to assign symbolic names which may be more meaningful.
- Given an array `string d[7]`, instead of writing `d[0] = "Sunday"` etc. we can write

```
d[SUNDAY] = "Sunday";
d[MONDAY] = "Monday";
...
d[SATURDAY] = "Saturday";
```

- **An enum can be declared as a data type.**
  1. To distinguish two enums we can declare them as follows.

     ```
     enum Days { SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY };
     enum Months { JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC};
     ```

  2. Both `SUNDAY=0` and `JAN=0`, but using enums sets the context of what “zero” means for their usage in the program code.
  3. We can declare variables and write code such as the following.

     ```
     Days dayOfWeek = ...                    // enum "Days" is a data type
     Months monthOfYear = ...                // enum "Months" is a data type
     if (dayOfWeek == MONDAY) { ... }
     if (monthOfYear == AUG) { ... }
     ```

- Enums are commonly employed with the **switch** statement.

2 Switch statement

2.1 Introduction and example

- We usually think of a conditional test as a Boolean statement (two-way branch).
- The **switch** statement is a multi-way branch statement.
- Another designation is that a switch statement is a **selection statement.**
- A switch statement selects one out of multiple possible outcomes (as opposed to only true/false).
- Consider the following scenario, which could happen in a computer graphics game.
    1. The user clicks the Escape or ‘Q’ or ‘q’ keys: quit the game.
    2. The user clicks the ‘R’ or ‘r’ keys: reset the game.
    3. Else continue the game animation.
- For the mouse buttons, if the left button is pressed, take some action. If the right or middle button is pressed, display a popup menu of suitable items. (In a real computer animation, we would also want to know the x and y coordinates on the screen at the location of the mouse cursor. We would also need to know if the mouse click is an up or down click. We ignore such complications below.)
- We could implement the above via the following switch statements.
- **See next page(s).**

```
enum Keyboard { ESC=(value), Key_Q='Q', Key_q = 'q', Key_R = 'R', Key_r = 'r' };
enum MouseButton { MOUSE_LEFT, MOUSE_RIGHT };

// code in calling application
int key = ...                        // input from user
int button = ...                     // input from user


void KeyboardFunction(int key)
{
  switch (key) {
  default:
    continueGame();
    break;

  case ESC:   // escape key
  case Key_q:
  case Key_Q:
    exitGame();
    break;

  case Key_r:
  case Key_R:
    resetGame();
    break;
  }
}

void MouseFunction(int button)
{
  switch (button) {
  case MOUSE_LEFT:
    mouseLeft();
    break;

  case MOUSE_RIGHT:
  default:
    displayPopup();
    break;
  }
}
```

2.2 Switch statement: syntax

- We must learn the syntax of the switch statement.
- The argument of the switch statement is an integer or an expression whose result can be converted to an integer value.
- The formal definition is the argument must be of **integral type.**
- This is different from an "if" statement, where the argument evaluates to a Boolean value.
- Next we consider the "`case`" statements.
  1. Each `case` statement enumerates a value the argument can take.
  2. Therefore the value of each case must also be of integral type.
  3. The values in the case statements need not be consecutive, the order is arbitrary.
  4. Two or more `case` statements cannot have the same value: they must be distinct.
  5. The value of a `case` statement must be an **integer constant.**
  6. An expression such as "`case 1+2*3`" is allowed.
  7. A `char` value "`case 'a'`" is allowed (`char` is of integral type).
  8. A floating point constant "`case 0.5`" is not allowed (`double` is not of integral type).
  9. The statement "`case i`" is not allowed (i is an integer variable, not constant).
- If the value of the input argument matches the value of a `case` statement, the block of code immediately following that `case` statement is executed.
  1. However, the situation is not so simple.
  2. We shall discuss the matter in more detail in Sec. 2.3 below.
- If the value of the input argument does not match the values of any `case` statement, the block of code following the "`default`" statement is executed.
  1. The use of "`default`" is optional.
  2. If there is no "`default`" statement, and if the value of the input argument does not match the values of any `case` statement, then none of the code in the switch is executed.
  3. **The "default" statement can appear anywhere.**
  4. The "default" statement does not have to be first or last.

2.3 Switch statement: flow of execution

- The execution of the code in a switch statement obeys some peculiar rules.
- If the input argument finds a match with a `case` statement, the code following that `case` is executed until a `break` statement is encountered.
- The program execution exits the switch statement when a `break` statement is encountered.
- This is sometimes called a "drop through" feature.
- For the example in Sec. 2.1, the following actions occur if the keyboard buttons are pressed.

   1. If `key == ESC`, execution drops through `Key_q` and `Key_Q` and "`exitGame()`" is called. The switch statement is exited at the `break` statement after `exitGame()`.
   2. If `key == Key_q`, execution also drops through `Key_Q` and "`exitGame()`" is called. The switch statement is exited at the `break` statement after `exitGame()`.
   3. If `key == Key_r`, execution drops through `Key_R` and "`resetGame()`" is called. The switch statement is exited at the `break` statement after `resetGame()`.
   4. For other keys, the `default` scenario is executed. The switch statement is exited at the `break` statement after `continueGame()`.

- For the example in Sec. 2.1, the following actions occur if a mouse button is clicked.

   1. If `button == MOUSE_LEFT`, the function "`mouseLeft()`" is called. The switch statement is exited at the `break` statement after `mouseLeft()`.
   2. If `button == MOUSE_RIGHT`, execution drops through and "`displayPopup()`" is called. The switch statement is exited at the `break` statement after `displayPopup()`.
   3. The function "`displayPopup()`" is also called in the `default` scenario.
   4. **The `default` scenario can share a drop through with `case` statements.**

2.4 Switch statement: nesting

- It is perfectly possible for switch statements to be nested.
- The code block following a case statement can contain any valid C++ code.
- In particular the code block can contain a nested switch statement.

3 Switch and if

- A switch statement can always be implemented using multiple "if" tests.

```
if      (condition #1) { code block #1 }
else if (condition #2) { code block #2 }
...
else                   { code block "else" }
```

- The use of multiple "if" statements offers greater generality.
- The use of a switch statement permits optimizations and more readable code in special circumstances.

3.1 If:

- **Each conditional test "condition #1" and "condition #2" etc. must be executed and its true/false outcome determined.**
- Tests employing non-integral variables such as "`x == 0.5`" are allowed.
- Each condition is a full C++ statement (which returns a Boolean value). For example "condition #1" could be the output of a function call, and "condition #2" could be the output of a different function call, etc.

```
if      (func1(...) != 2)   { code block #1 }
else if (func2(...) > 0.5)  { code block #2 }
...
else                        { code block "else" }
```

3.2 Switch:

- The value of the input argument is **computed only once.**
- The values of the cases are restricted to be constants, and of integral type only.
- **The tests are only for equality,** i.e. `input == (value of case)`. Tests for negations such as "`input != 2`" or inequalities such as "`input > 0.5`" are not permitted.
- The values of the cases are known at compile time, whereas the results of the conditional tests in each "if" statement must be computed at run time.
- Because of these restrictions, the compiler is able to perform optimizations which are not possible for a set of multiple "if" statements.
- The compiler sees all the outcomes (all the cases) "at once" as opposed to testing a set of conditional "if" statements sequentially.
- The cases in a switch statement typically employ enums.
- Enums usually have descriptive names which make the code easier to read and understand.