1

Outline

- In this lesson, we will:
  - Describe deeper cascading conditional statements
  - Explain how the switch statement can be used
  - Give some examples of its use

2

Cascading conditional statements

- Consider the following cascading conditional statement:

```
if ( error_code == 0 ) {
    // Deal with Error Code 0
} else if ( error_code == 1 ) {
    // Deal with Error Code 1
} else if ( error_code == 12 ) {
    // Deal with Error Code 12
} else if ( error_code == 13 ) {
    // Deal with Error Code 13
} else if ( error_code == 42 ) {
    // Deal with Error Code 42
} else {
    // Deal with all other error codes
}
```

3

Vowels versus consonants

- Another for dealing with letters

```
if ( (ch == 'a') || (ch == 'e') || (ch == 'i')
                 || (ch == 'o') || (ch == 'u') ) {
    // React to a vowel
} else if ( ch == 'y' ) {
    // React to the special case of the letter 'y'
} else {
    // Deal with all consonants
}
```

4

Hexadecimal to decimal

- Here we convert a hexadecimal character to a decimal equivalent

```
if ( (hex >= '0') && (hex <= '9') ) {
    return hex - '0';
} else if ( (hex >= 'a') && (hex <= 'f') ) {
    return hex - 'a' + 10;
} else if ( (hex >= 'A') && (hex <= 'F') ) {
    return hex - 'A' + 10;
} else {
    // Deal with an invalid hexadecimal digit
}
```

5

Parsing an arithmetic expression

- This could be used parsing an arithmetic expression

```
if ( (ch >= '0') && (hex <= '9') ) {
    // Deal with a number
} else if ( hex == '(' ) {
    // Deal with an opening parenthesis
} else if ( hex == ')' ) {
    // Deal with a closing parenthesis
} else if ( (char == '+') || (hex == '-') ) {
    // Deal with possibly unary or binary + or -
} else if ( (char == '*') || (hex == '/') || (hex == '%') ) {
    // Deal with binary *, / or %
} else {
    // Deal with anything else including identifiers
    //  - And for the keeners, yes, we assumed
    //      that * was multiplication and not a pointer dereference
}
```

6

The switch statement

- In all of these cases, a single variable was being compared with either individual constants or a small range of constants
- Such sequential testing requires significant amounts of time, especially for the last test to be performed

- C++ includes a short-cut mechanism using either a mathematical formula formulated at compile time, or a look-up table or a binary search thereof, or a combination of these together with perhaps simpler conditional statements
  - This is called a *switch statement*
  - The name is likely derived from older mechanical switching devices that could make connections directly based on a value

7

Example

```
int error_code{};
// Use 'error_code'

switch ( error_code ) {
    case 0:
        // Deal with Error Code 0
        break;
    case 1:
        // Deal with Error Code 1
        break;
    case 12:
        // Deal with Error Code 12
        break;
    case 13:
        // Deal with Error Code 13
        break;
    case 42:
        // Deal with Error Code 42
        break;
    default:
        // Deal with all other error codes
}
// The break jumps to the end of the statement
```

```
if ( error_code == 0 ) {
    // Deal with Error Code 0
} else if ( error_code == 1 ) {
    // Deal with Error Code 1
} else if ( error_code == 12 ) {
    // Deal with Error Code 12
} else if ( error_code == 13 ) {
    // Deal with Error Code 13
} else if ( error_code == 42 ) {
    // Deal with Error Code 42
} else {
    // Deal with all other error codes
}
```

8

Example

```
char ch{};
// Do something with 'ch'

switch ( ch ) {
    case 'a':
    case 'e':
    case 'i':
    case 'o':
    case 'u':
        // React to a vowel
        break;
    case 'y':
        // React to the special case of the letter 'y'
        break;
    default:
        // Deal with all consonants
}
// The break jumps to the end of the statement
```

```
if ( (ch == 'a') || (ch == 'e') || (ch == 'i')
                 || (ch == 'o') || (ch == 'u') ) {
    // React to a vowel
} else if ( ch == 'y' ) {
    // React to the special case of the letter 'y'
} else {
    // Deal with all consonants
}
```

Example

```
switch ( ch ) {
    case '0':  case '1':
    case '2':  case '3':
    case '4':  case '5':
    case '6':  case '7':
    case '8':  case '9':
        return ch - '0';
    case 'a':  case 'b':
    case 'c':  case 'd':
    case 'e':  case 'f':
        return ch - 'a' + 10;
    case 'A':  case 'B':
    case 'C':  case 'D':
    case 'E':  case 'F':
        return ch - 'A' + 10;
    default:
        throw std::invalid_argument{ "Expecting a hexadecimal digit" };
}
```

```
if ( (hex >= '0') && (hex <= '9') ) {
    return hex - '0';
} else if ( (hex >= 'a') && (hex <= 'f') ) {
    return hex - 'a' + 10;
} else if ( (hex >= 'A') && (hex <= 'F') ) {
    return hex - 'A' + 10;
} else {
    // Deal with an invalid hexadecimal digit
}
```

Example

- Of course, sometimes an array is faster:

```
int hex_to_int[55] {
   0,  1,  2,  3,  4,  5,  6,  7,  8,  9, -1, -1, -1, -1, -1, -1,
  -1, 10, 11, 12, 13, 14, 15, -1, -1, -1, -1, -1, -1, -1, -1, -1,
  -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
  -1, 10, 11, 12, 13, 14, 15
};

// ...

assert( (ch >= '0') && (ch <= 'f') && (hex_to_int[ch - '0'] != -1 );
return hex_to_int[ch - '0'];
```

Example

```
char ch{};
// Do something...

switch ( ch ) {
    case '0':  case '1':
    case '2':  case '3':
    case '4':  case '5':
    case '6':  case '7':
    case '8':  case '9':
        // Deal with a number
    case '(':
        // Deal with an opening parenthesis
    case ')':
        // Deal with a closing parenthesis
    case '+':  case '-':
        // Deal with possibly  unary or binary + or -
    case '*':  case '/':  case '%':
        // Deal with binary *, / or %
    default:
        // Deal with anything else, including identifiers
}
```

```
if ( (ch >= '0') && (hex <= '9') ) {
    // Deal with a number
} else if ( hex == '(' ) {
    // Deal with an opening parenthesis
} else if ( hex == ')' ) {
    // Deal with a closing parenthesis
} else if ( (char == '+') || (hex == '-') ) {
    // Deal with possibly unary or binary + or -
} else if ( (char == '*') || (hex == '/') || (hex == '%')
) {
    // Deal with binary *, / or %
} else {
    // Deal with anything else, including identifiers
}
```

Allowable types

- The argument must be an integer, character or Boolean type
  - The type cannot be `float`, `double`, a pointer or any class
  - It can also be an enumerated type

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

```
switch ( wkdy ) {
    case MONDAY:
        // Deal with a Monday
        break;
    case TUESDAY:
        // Deal with a Tuesday
        break;
    case WEDNESDAY:
        // Deal with a Wednesday
        break;
    case THURSDAY:
        // Deal with a Thursday
        break;
    case FRIDAY:
        // Deal with a Friday
        break;
    default:
        // Weekend!
}
```

13

Allowable types

- The identifier must be an integer, character or Boolean type

```
enum Months {
    JANUARY,
    FEBRUARY,
    MARCH,
    APRIL,
    MAY,
    JUNE,
    JULY,
    AUGUST,
    SEPTEMBER,
    OCTOBER,
    NOVEMBER,
    DECEMBER
};
```

```
switch ( current.month() ) {
    case JANUARY:  case MARCH:  case MAY:
    case JULY:     case AUGUST: case OCTOBER:
    case DECEMBER:
        length = 31;
        break;
    case APRIL:     case JUNE:
    case SEPTEMBER: case NOVEMBER:
        length = 30;
        break;
    case FEBRUARY:
        if ( (current.year()%4 != 0) || (
             (current.year()%100 == 0) && (current.year()%400 != 0)
        ) ) {
            length = 28;
        } else {
            length = 29;
        }
    case default:
        assert( false );
}
```

14

Allowable types

- Of course, sometimes an array is easier:

```
enum Months {
    JANUARY,
    FEBRUARY,
    MARCH,
    APRIL,
    MAY,
    JUNE,
    JULY,
    AUGUST,
    SEPTEMBER,
    OCTOBER,
    NOVEMBER,
    DECEMBER
};
```

```
// Declare an array
unsigned int month_lengths[12]{
    31,  0, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31
};


if ( current.month() == FEBRARY ) {
    length = ( (current.year()%4   != 0) || (
         (current.year()%100 == 0) && (current.year()%400 != 0)
    ) ) ? 28 : 29;
} else {
    length = month_lengths[current.month()];
}

assert( length != 0 );
```

15

Most significant source of error

- Novice programmers often forget to include the break statement at the end of each case if they intend to for the code executed to be mutually exclusive

```
switch ( wkdy ) {
    case MONDAY:
        std::cout << "Today is the Moon's day" << std::endl;
    case TUESDAY:
        std::cout << "Today is Tiw's day" << std::endl;
    case WEDNESDAY:
        std::cout << "Today is Woden's day" << std::endl;
    case THURSDAY:
        std::cout << "Today is Thor's day" << std::endl;
    case FRIDAY:
        std::cout << "Today is Freya's day" << std::endl;
    case SATURDAY:
        std::cout << "Today is Saturn's day" << std::endl;
    case SUNDAY:
        std::cout << "Today is the Sun's day" << std::endl;
    default:
        assert( false ); // Should not happen
}
```

16

Most underused feature

- One of the most underused features is to use the fall through to perform two actions for some cases, but only one for others
  - Suppose an escalating response is required

```
switch ( error_code ) {
    case   5: case  16: case 154:
        // Critical errors
        //  - send a report on the error
    case   1: case  73: case  97:
    case  54: case 253: case 255:
        // Serious errors
        //  - alert the user
    default:
        // All errors
        //  - log the error
}
```


Summary

- Following this lesson, you now
  - Understand how and when a switch statement can be used
  - Are aware of the potential benefits

References

[1] https://en.wikipedia.org/wiki/Switch_statement

Colophon

These slides were prepared using the Georgia typeface. Mathematical equations use Times New Roman, and source code is presented using `Consolas`.

The photographs of lilacs in bloom appearing on the title slide and accenting the top of each other slide were taken at the Royal Botanical Gardens on May 27, 2018 by Douglas Wilhelm Harder. Please see

https://www.rbg.ca/

for more information.



Disclaimer

These slides are provided for the ECE 150 *Fundamentals of Programming* course taught at the University of Waterloo. The material in it reflects the authors' best judgment in light of the information available to them at the time of preparation. Any reliance on these course slides by any party for any other purpose are the responsibility of such parties. The authors accept no responsibility for damages, if any, suffered by any party as a result of decisions made or actions based on these course slides for any other purpose than that for which it was intended.