1. **Nested if:** if where either the true path or false path or both contain another if

2. ```
10    int grade;
20    ...
30    if (grade >= 0 && grade <= 100) // Simple if - not nested
40        cout << "In range"; // 0 to 100, inclusive
50    else
60        cout << "Out of range"; // negative or greater than 100

70    if (grade >= 0) // Nested if
80        if (grade <= 100)
90            cout << "In range";
100        else
110            cout << "Too high";
120    else
130        cout << "Too low";
```
3. 140 char pet;
    150 double weight;
    160 int fee;  // Boarding fee
    170 ...  
    180 if (pet == 'c')  // Cat
    190     fee = 15;
    200 else         
    210     if (weight > 55.0)  // Dog
    220       fee = 50;
    230     else
    240       fee = 35;
Assume region code is 'e' or 'w'

```
false                                           true
region is 'e'

false         true
region is 'e'

false         true
region is 'w'
```

Assume region code is 'e', 'w', or invalid

```
false                                           true
region is 'e'

false               true
region is 'w'
```

```
do
western
processing

do
eastern
processing

do
invalid
processing

do
eastern
processing

do
western
processing
```
1.  char region;
    ... 
    if (region == 'e') // Nested if
    {
        // do eastern processing
    }
    else
    {
        if (region == 'w')
        {
            // do western processing
        }
        else
        {
            // do invalid processing
        }
    }

2.  if (region == 'e') // Multiway Branch
    {
        // do eastern processing
    }
    else if (region == 'w')
    {
        // do western processing
    }
    else
    {
        // do invalid processing
    }
3. **Definition of a Case Structure:** A special form of a nested if where based on one value, usually stored in a variable, the computer will execute one portion of code (one case) out of multiple, mutually exclusive portions of code (cases).

```java
28    switch (region)
29    {
30        case 'e':  // do eastern processing
31            break;
32        case 'w':  // do western processing
33            break;
34        default:  // do invalid processing
35            break;
36    }
```

**Case Structure:**

```
region ?

= 'e'  
  do eastern processing

= 'w'  
  do western processing

= other  
  do invalid processing
```
1. **switch** is used primarily as an alternative to a nested **if** or multiway branch to implement a case structure.

2. General Form:

   ```
   switch (integer expression)
   {
     case constant1:
       statement sequence 1;

     case constant2:
       statement sequence 2;

     case constant3:
       statement sequence 3;

     ...

     default:
       statement sequence n;
   }
   // Note that there is no ;
   ```

   **switch**, **case**, and **default** are keywords.

3. expression must yield integer value: can be **int**, **short**, or **char** can be negative

4. constants or constant expressions must be integer types (NO variables)

5. **Action:**

   Switch expression is evaluated

   Value of the expression is compared to the constants, looking for a match

   If there is a match
   control is transferred to corresponding statement sequence (case)

   Else
   control is transferred to the default case, if present, or to statement following switch
6. **break;**

   Transfers control to statement following switch.

   Without break, execution continues through to the end of the switch.

7. **default:**

   Optional; can have only one default case

   May be at any location; should be at bottom.

8. case expressions must be unique.

9. **if ... else** is more versatile
   a. handles ranges
      
      \[
      \text{if (age} > 17 \land \text{age} < 30) \quad \text{ // 18 to 29, inclusive}
      \]
   b. handles any type

10. **switch** is usually more efficient in
    a. code size and
    b. execution speed

    unless there are only 2 cases
Definition of a Case Structure:
A special form of a nested if where based on one value, usually stored in a variable, the computer will execute one portion of code (one case) out of multiple, mutually exclusive portions of code (cases).

I. Update the appropriate counter based on the academic year; 5 cases

```java
int year; // Academic year
int fresh = 0, soph = 0, junior = 0, senior = 0, invalid = 0;
...
if (year == 1)
    fresh++;
else
    if (year == 2)
        soph++;
    else
        if (year == 3)
            junior++;
        else
            if (year == 4)
                senior++;
            else
                invalid++;

if (year == 1) // Preferred indentation for Multiway Branches
    fresh++;
else if (year == 2)
    soph++;
else if (year == 3)
    junior++;
else if (year == 4)
    senior++;
else
    invalid++;

switch (year)
{
    case 1:
        fresh++;
        break;
    case 2:
        soph++;
        break;
    case 3:
        junior++;
        break;
    case 4:
        senior++;
        break;
    default:
        invalid++;
        break;
}
```
II. Update the appropriate counter based on the letter grade; 6 cases

```cpp
char grade; // Letter grade
int acount = 0, bcount = 0, ccount = 0, dcount = 0, fcount = 0;
...
if (grade == 'A' || grade == 'a')
  acount++;
else if (grade == 'B' || grade == 'b')
  bcount++;
else if (grade == 'C' || grade == 'c')
  ccount++;
else if (grade == 'D' || grade == 'd')
  dcount++;
else if (grade == 'F' || grade == 'f')
  fcount++;
else
  cout << "Invalid letter grade
";

switch (grade)
{
case 'A':
case 'a':
  acount++;
  break;
case 'B':
case 'b':
  bcount++;
  break;
case 'C':
case 'c':
  ccount++;
  break;
case 'D':
case 'd':
  dcount++;
  break;
case 'F':
case 'f':
  fcount++;
  break;
default:
  cout << "Invalid letter grade.
";
  break;
}
```
III. Set the letter grade based on a numeric score; 5 cases

100 char letter_grade;
101 int score;
102 ...
103 if (score >= 90)  
104 letter_grade = 'A';
105 else if (score >= 80)  
106 letter_grade = 'B';
107 else if (score >= 70)  
108 letter_grade = 'C';
109 else if (score >= 60)  
110 letter_grade = 'D';
111 else  
112 letter_grade = 'F';

113 switch (score / 10) // Uses integer division
114 {
115  case 10:
116  case 9:
117   letter_grade = 'A';
118   break;
119  case 8:
120   letter_grade = 'B';
121   break;
122  case 7:
123   letter_grade = 'C';
124   break;
125  case 6:
126   letter_grade = 'D';
127   break;
128  default:  
129    letter_grade = 'F';
130    break;
131 }

IV. Classify lowercase letter; 2 cases: vowel or consonant

152 char letter;
153 ...
154 if (letter == 'a' || letter == 'e' || letter == 'i' || letter == 'o' || letter == 'u' || letter == 'y')
155    cout << "vowel\n";
156 else  
157    cout << "consonant\n";

158 switch (letter)
159 {
160    case 'a':
161    case 'e':
162    case 'i':
163    case 'o':
164    case 'u':
165    case 'y':
166      cout << "vowel\n";
167      break;
168    default:  
169      cout << "consonant\n";
170      break;
171   }
V. Perform the appropriate operation; 5 cases

```c
float operand1, operand2, result;
char operator;
...
if (operator == '+')
  result = operand1 + operand2;
else if (operator == '-')
  result = operand1 - operand2;
else if (operator == '*')
  result = operand1 * operand2;
else if (operator == '/')
  if (operand2 == 0)
    cout << "Cannot divide by zero\n";
  else
    result = operand1 / operand2;
else
  cout << "Invalid operator\n";

switch (operator)
{
  case '+':
    result = operand1 + operand2;
    break;
  case '-':
    result = operand1 - operand2;
    break;
  case '*':
    result = operand1 * operand2;
    break;
  case '/':
    if (operand2 == 0)
      cout << "Cannot divide by zero\n";
    else
      result = operand1 / operand2;
    break;
  default:
    cout << "Invalid operator\n";
    break;
}
```
1. **Review:**

   **Input-Bound Loops:**
   exit at the end of the data

2. **Counter-Controlled Loop:**
   number of iterations is known

   Use a counter variable to count the iterations:
   a. initialize counter before the loop
   b. test counter at top of the loop
   c. increment counter at bottom of the loop

3. **Output:**

   1
   2
   3 first 3 lines printed within loop
   6 printed after loop

   ```
   int i, sum = 0;
   i = 1; // Initialize counter
   while (i <= 3) // Test counter
   {
       cout << i << endl; // Increment counter
       sum += i;
   }
   cout << endl << sum;
   ```

   ```
   int i, sum = 0;
   for (i = 1; i <= 3; i++)
   {
       cout << i << endl; // Note indentation
       sum += i;
   }
   cout << endl << sum;
   ```
4. General Form:
   for (initialization expression; test expression; update expression)
   simple or compound statement;

   initialization expression: evaluated once at start of for

   test expression: evaluated before each possible iteration
                   if true
                   execute loop
                   else
                   exit loop

   update expression: evaluated at end of each iteration and
                    control goes back to test expression
5. The comments show the output from each loop.

a. `for (i = 1; i <= 5; i++)`  // 1 2 3 4 5
   `cout << setw(2) << i;`

b. `for (i = 1; i <= 6; i++)`  // 1 2 3 4 5 6
   `cout << setw(2) << i;`

c. `for (i = 5; i >= 1; i--)`  // 5 4 3 2 1
   `cout << setw(2) << i;`

d. `for (i = 0; i < 5; i++)`  // 0 1 2 3 4
   `cout << setw(2) << i;`

e. `for (i = 0; i < 10; i += 2)`  // 0 2 4 6 8
   `cout << setw(2) << i;`

f. `for (int i = 1; i <= 4; i++)`  // 1 2 3 4
   `cout << setw(2) << i;`