# Chapter - 8
# More Control Statements

# *for* Statement

General form:

```
for (initial-statement; condition;
                iteration-statement)
    body-statement;
```

Is equivalent to:

```
initial-statement;
while (condition) {
    body-statement;
    iteration-statement;
}
```

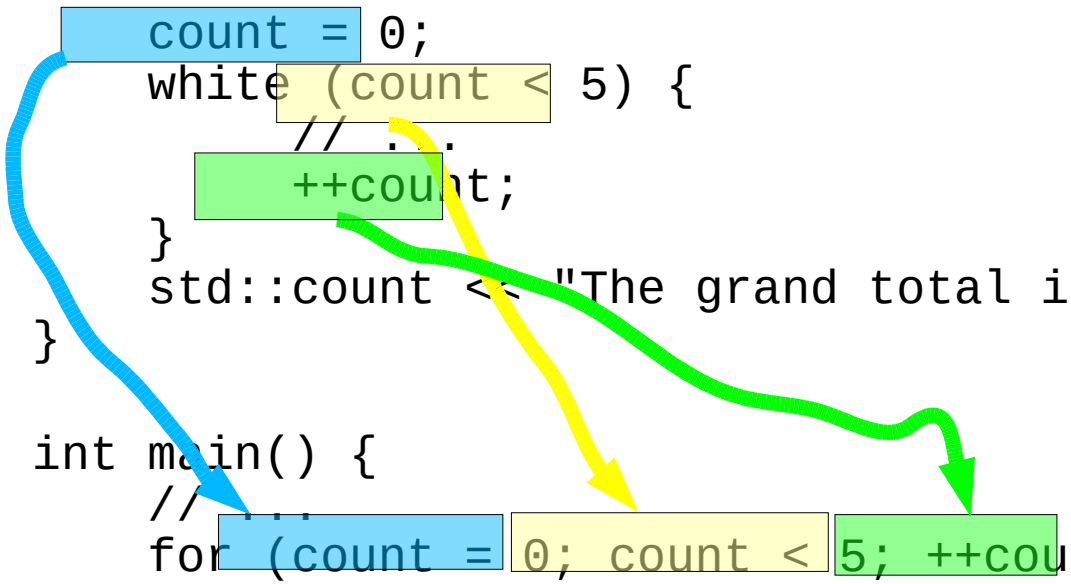
# *for* Example

```
#include <iostream>


int total;      // total of all the numbers
int current;    // current value from the user
int counter;    // for loop counter


int main() {
    total = 0;
    for (counter = 0; counter < 5; ++counter) {
        std::cout << "Number? ";


        std::cin >> current;
        total += current;
    }
    std::cout << "The grand total is " << total << '\n';
    return (0);
}
```

**Note that `counter` goes from 0 to 4. Normally you count five items as 1,2,3,4,5. You will get along much better in C++ if you change your thinking to zero-based counting and count five items as 0,1,2,3,4.**

# *for* vs. *while*

```
int main() {
    // ...
    count = 0;
    while (count < 5) {
        // ...
        ++count;
    }
    std::count << "The grand total is " << total << '\n';
}

int main() {
    // ...
    for (count = 0; count < 5; ++count) {
        // ...
    }
    std::count << "The grand total is " << total << '\n';
}
```



# Question: What Does this Program Print?

```
#include <iostream>
/*
 * Produces a Celsius to Fahrenheit conversion
 *    chart for the numbers 0 to 100.
 *
 * Restrictions:
 *      This program deals with integers only, so the
 *      calculations may not be exact.
 */


// the current Celsius temperature we are working with
int celsius;
int main() {
    for (celsius = 0; celsius <= 100; ++celsius);
        std::cout << "celsius: " << celsius <<
                " Fahrenheit: " <<
                ((celsius * 9) / 5 + 32) << '\n';
    return (0);
}
```

# Question: Why Does this Program Print the Wrong Answer?

```
#include <iostream>
int seven_count;    // number of sevens in the data
int data[5];        // the data to count 3 and 7 in
int three_count;    // the number of threes in the dat
 a
int index;          // index into the data
```

# Program (cont.)

```
int main()
{
    seven_count = 0;  three_count = 0;
    std::cout << "Enter 5 numbers\n";
    std::cin >> data[1] >> data[2] >> data[3] >>
                data[4] >> data[5];


    for (index = 1; index <= 5; ++index) {
        if (data[index] == 3)
            ++three_count;


        if (data[index] == 7)
            ++seven_count;
    }


    std::cout << "Threes " << three_count <<
            " Sevens " << seven_count << '\n';
    return (0);
}
```

# *switch* Statement

General form:

```
switch ( expression ) {
    case constant1:
        statement
        . . . .
        break;
    case constant2:
        statement
        . . . .
        // Fall through
    default:
        statement
        . . . .
        break;
    case constant3:
        statement
        . . . .
        break;
}
```

# From the *calc* program

```
if (operator == '+') {
    result += value;
} else if (operator == '-') {
    result -= value;
} else if (operator == '*') {
    result *= value;
} else if (operator == '/') {
    if (value == 0) {
        std::cout << "Error:Divide by zero\n";
        std::cout << "   operation ignored\n";
    } else
        result /= value;
} else {
    std::cout << "Unknown operator " <<
                 operator << '\n';
}
```

# As a **switch** Statement

```
#include <iostream>
int   result;      // the result of the calculations
char  oper_char;   // operator the user specified
int   value;       // value specified after the operato
  r
main() {
    result = 0;    // initialize the result


    // loop forever (or until break reached)
    while (1) {
        std::cout << "Result: " << result << '\n';
        std::cout << "Enter operator and number: ";
        std::cin >> oper_char >> value;


        if ((oper_char == 'q') || (oper_char == 'Q'))
            break;
```

# As a switch (cont.)

```
  switch (oper_char) {
    case '+':
        result += value;
        break;
    case '-':
        result -= value;
        break;
    case '*':
        result *= value;
        break;
    case '/':
        if (value == 0) {
            std::cout << "Error:Divide by zero\n";
            std::cout << "   operation ignored\n";
        } else
            result /= value;
        break;
    default:
        std::cout << "Unknown op. " <<oper_char << '\n';
        break;
  }
}
```

# Ending `breaks`

A **break** is not required at the end of a case switch:

```
// a not so good example of programming
switch (control) {
        case 0:
                std::cout << "Reset\n";
        case 1:
                std::cout << "Initializing\n";
                break;
        case 2:
                std::cout << "Working\n";
}
```

Did the programmer intend to fall through for `case 0` or did he forget the break statement?

# A Better Switch

```
// a better example of programming
switch (control) {
        case 0:
                std::cout << "Reset\n";
                // Fall through
        case 1:
                std::cout << "Initializing\n";
                break;
        case 2:
                std::cout << "Working\n";
}
```

# Let's Add a New Case at the End

```
// We have a little problem
switch (control) {
   case 0:
       std::cout << "Reset\n";
      // Fall through
   case 1:
        std::cout << "Initializing\n";
        break;
   case 2:
        std::cout << "Working\n";
   case 3:
        std::cout << "Closing down\n";
}
```

We have a problem.

# Our Problem is Fixed.

```
// Almost there
switch (control) {
        case 0:
                std::cout << "Reset\n";
                // Fall through
        case 1:
                std::cout << "Initializing\n";
                break;
        case 2:
                std::cout << "Working\n";
                break;
        case 3:
                std::cout << "Closing down\n";
                break;
}
```

But what happens if control is '5'. The switch does nothing. Did the programmer intend for this to happen or is it just an accident.

# Final switch

```
// The final version
switch (control) {
    case 0:
        std::cout << "Reset\n";
        // Fall through
    case 1:
        std::cout << "Initializing\n";
        break;
    case 2:
        std::cout << "Working\n";
        break;
    case 3:
       std::cout << "Closing down\n";
       break;
    default:
        std::cout << "Internal error, control value " <<
                 control << " impossible\n";
        break;
}
```

A “default” is required even if it is only:

```
default:
        // Do nothing
        break;
```

# **switch**, **break**, and **continue**