Control Conditionals

Use the for loop.
Selective Execution

Boolean expression

True

Statement

False
v1, if alone

if (boolean expression)  
    statement;
Two alternatives

Boolean expression

True

Statement 1

False

Statement 2
v2, optional else

if (boolean expression)
    statement_1;
else
    statement_2;
Blocks

- A sequence of statements treated like a single statement
- A block of statements can go wherever any single statement can go
  - Not restricted to selection
- Syntax: set off by brackets {}
if (boolean1) {
    statement1;
    statement2;
} else {
    statementA;
    statementB;
}
if (boolean_expr1)
  statement_1;
else if (boolean_expr2)
  statement_2;
else if (boolean_expr3)
  statement_3;
else
  statement_last;

- Evaluate Booleans in order
  - If false, go on to the next
- First Boolean that evaluates to true has its statement (or block) run
  - Skip the rest of the if
- If no true Boolean, run the else
Nested ifs

if (boolean_1) {
    if (boolean_1_1) {
        statement;
    } else if (boolean_1_2) {
        statement;
    } else {
        statement;
    }
} // of boolean_1 if

- Formatting is helpful
- It is not required
  - Makes it clear where blocks begin and end
- Use comments if it is ugly
Dangling else problem

if (boolean_1)
  if (boolean_1_1)
    statement_1_1;
else
  statement_2;

Wrong indentation. else goes with the most recent if in the code.

```java
if (boolean_1)
  if (boolean_1_1)
    statement_1_1;
else
  statement_2;
```
What is the output?

// Example 2.9

#include <iostream>

int main() {
    int x = -4;
    if (x > 0)
        if (x < 10)
            std::cout << "Orange" << std::endl;
        else
            std::cout << "Yellow" << std::endl;
    else
        std::cout << "Yellow" << std::endl;
}

- Orange
- Yellow
- (No Output)
- I don't know
Repetition

While Loops

For Loops

Do-While Loops
Three Loops

- while
  - top-tested loop (pre-test)
- for
  - counting loop
  - forever-sentinel
- do
  - bottom-tested loop (post-test)
The while loop

while (expression)
    statement;

Test to start and after every repetition

Execute each repetition
while (condition)

condition

statement

True

False
Similar to the if statement

- Check the Boolean condition
- If true, execute the statement / block
- Repeat until the Boolean is false
Forever loops and never loops

- Because the conditional can be always true or always false you can get a loop that runs forever or never runs

```cpp
int count = 0;
while (count == 0) // forever
    cout << "Hi Mom";
while (count = 1) // insidious error!
    count = 0
```
count = 1 always returns 1 (true)

Possible solution, reverse: 1 == count is OK, 1 = count is illegal

```cpp
int count = 0
while (count != 0) // never
    cout << "Hi Mom";
while (1 = count) // won’t compile
    count = 0
```
How to count using while

- First, outside the loop, initialize the counter
- Test for the counter’s value in the condition
- Do the body of the loop
- Something in the body should change the value of the counter
More counting

```cpp
int counter = 0;  // init counter
while (counter < 10) {  // test counter
    cout << "hi mom";
    cout << "Counter is: " << counter << endl;
    counter++;  // change counter
}
```
do-while (condition)
Bottom-tested loop: do

- Bottom-tested (post-test)
- One trip through the loop is guaranteed
  - i.e. statement is executed *at least* once

```plaintext
do
    statement;
while (expression);
```
// Example 3.2
#include <iostream>
using std::cin;
using std::cout;
using std::endl;

int main() {
    // single statement while
    int cnt = 0;
    while (cnt < 5) ++cnt;
    cout << cnt << endl;

    // while block, count down
    cnt = 5;
    while (cnt > 0) {
        cout << cnt << ", ";
        --cnt;
    } // of while
    cout << endl << cnt << endl;

    /* basic while structure
     - set up start condition, outside loop
     - measure condition in Boolean
For loop

- while loop is pretty general.
- Anything that can be done using repetition can be done with a while loop.
- Because counting is so common, there is a specialized construct called a for loop.
- for loop makes it easy to setup a counting loop.
Three parts

- Three parts to a for loop (just like the while)
  - Set the initial value for the counter
  - Set the condition for the counter
  - Set how the counter changes each time through the loop
For loop

for (initialize; condition; change)
statement;

semicolons required!
for (count = 1; count <= 5; count++) statement

count = 1

- True: statement
- False: count++
- It is generally considered poor programming practice to alter the counter or limit variables within the body of the `for` loop.
- The components of the `for` statement can be arbitrary statements.
  - e.g. The loop condition can be a function call.
Which of the following is the least common loop syntax?

- While Loop
- Do-While Loop
- For Loop
- I don't know
Top-tested equivalence

for (x = init; x <= limit; x++)
    statement;

x = init;
while (x <= limit) {
    statement;
    x++;
}
C++ allows you to declare variables inside the `for` loop

- If declared *inside* the `for` loop, it is only available *inside* the loop
- The *scope* of the variable is the statement / block of the loop
```cpp
int i = 100;
for (int i = 10; i > 0; i--)
    cout << i;
```

This `i` is local scope to the loop.
Prints 10 to 1

This `i` is global scope.
Prints 100
Three fields are optional

```c++
int val = 10;
for (;;) {
    if (val <= 0)
        break;
    cout << "Infinite break val: " << val << endl;
    val -= 3;
}
```

- No init
- No condition
- No change per iteration
The comma operator, usually found inside one of the for loop fields, is used to perform a sequence of operations in that field.

- Comma guarantees execution order
  - Left-to-right
Comma Example

```cpp
for (int i = 10, j = 20; i * j < 500; i += 5, j += 5)
    cout << "Values are i:" << i << ", j" << j << endl;
```

Two local vars in the `for` loop
- Both `int`
- only one declare type allowed
- Both initialized
  - `i` first, then `j`

Loop starts with `i=10` and `j=20`

- `i` and `j` both increment by 5 each iteration
- Loop ends when `i * j > 500`
// Example 3.3
#include <iostream>
using std::cin;
using std::cout;
using std::endl;

int main() {
    // basic for loop
    for (int i = 0; i < 5; ++i) cout << i << "", "
    cout << endl;

    // equivalent while loop
    int i = 0;
    while (i < 0) {
        cout << i << "", "
        ++i;
    }
    cout << endl;

    // for with block
    int cnt = 0;
    for (int i = 5; i >= 0; --i) {
        cout << i << "", "
        ++cnt;
    }
    cout << endl;

    // won't compile, only one type in init
    for (int i = 10, int j = 20; i * j < 500; i += 5, j += 5)
        cout << "Values are i:" << i << " j:" << j << endl;

    cout << "Final i:" << i << " j:" << j << endl;

    // comma separated fields. Only one type in the init field
    i = 20;
    j = 10;
    for (i = 10, j = 20; i * j < 500; i += 5, j += 5)
        cout << "i:" << i << " j:" << j << endl;

    cout << "Final i:" << i << " j:" << j << endl;

    // no expression, cond ends with EOF
    for (int i; cin >> i;)
        cout << "Input Value was:" << i << " SAW THE EOF CHARACTER" << endl;
}

Non-local exit (break and continue)

- The structure of iteration helps us, as readers, understand clearly when iteration continues and when it ends.

- Non-local exits can be important, but beware that they can make the code very difficult to read.
Break and Continue

- **break**
  - Exit the nearest enclosing loop struct *(for, while, etc.)*
  - If nested, exit to the enclosing control

- **continue**
  - Stop the present iteration of the loop
  - Start the next
Breaks are for loops / switches

- The break statement is for loops and the (upcoming) switch statement
  - Don’t break out of an if block!
- Can goto which requires a label (don't ever use this!)

... 
goto jmp
... 
jmp:
// Example 3.4
#include <iostream>
using std::cin;
using std::cout;
using std::endl;
using std::fixed;
#include <iomanip>
using std::setprecision;

int main() {
    // sum up the numbers, end at 0
    int sum = 0;
    int num;
    cout << "Enter a number. Ends when 0 entered" << endl;
    cout << "Number:"
    cin >> num;
    for (;;) { // infinite for
        sum += num;
        if (num == 0) break;
        cout << "Number:"
        cin >> num;
    } // of for
} // of main
// Example 3.5
#include <iostream>
#include <iomanip>
using std::cin;
using std::cout;
using std::endl;
using std::noskipws;
using std::setw;

int main() {
    char C;  // Input character
    int char_count = 0, // Number of characters in input stream
        line_count = 0, // Number of newlines
        blank_count = 0, // Number of blanks
        digit_count = 0, // Number of digits
        letter_count = 0, // Number of letters
        other_count = 0; // Number of other characters

    cin >> noskipws;

    while (cin >> C) {
        ++char_count;  // Increment character count
    }
}
Switch Statement

- A less general substitute for the multibranch if. It is used for selecting among discrete values (int-ish), i.e. not continuous values.

```java
switch (int_expression) {
    case val1: statement_list;
    case val2: statement_list;
    ...
    default: statement_list;
}
```
switch (int_expression) {
    case val1: statement_list;
    case val2: statement_list;
    default: statement_list;
}
Behavior

1. The `int_expression` is evaluated

2. If the value is in a `case`, execution begins at that `statement_list`
   1. Continues through subsequent `statement_lists` until `break` or `return`

3. If no `case` is true do the default
   1. `default` is optional, do nothing if nothing is true and no default
// Example 3.6
#include <iostream>
using std::cin;
using std::cout;
using std::endl;
#include <iomanip>
using std::noskipws;
using std::setw;

int main() {
    char C; // Input character
    int char_count = 0, // Number of characters in input stream
        line_count = 0, // Number of newlines
        white_count = 0, // Number of whitespace characters
        digit_count = 0, // Number of digits
        other_count = 0; // Number of other characters

    cin >> noskipws;

    while (cin >> C) {
        ++char_count; // Increment overall character count
        switch (C) { // Classify each character
            case '\n': // Newline
                ++line_count;
                break;
            case ' ': // White space (newline, blank or tab)
            case '	':
                ++white_count;
                break;
            case '0': // Decimal digit
            case '1':
            case '2':
            case '3':
            case '4':
            case '5':
            case '6':
            case '7':
            case '8':
            case '9':
                ++digit_count;
                break;
            default: // Other character
                ++other_count;
                break;
        }  
    }  

    cout << endl;
    cout << "Newlines:   " << setw(5) << line_count << endl;
    cout << "Whitespace: " << setw(5) << white_count << endl;
    cout << "Digits:     " << setw(5) << digit_count << endl;
    cout << "Other:      " << setw(5) << other_count << endl;
    cout << "Total:      " << setw(5) << char_count << endl;
    cout << endl;
}
The problem with break

- You get “fall-through” behavior if you do not put a `break` at the end of every case group.
- Easily forgotten!
  - It’s a feature, not a bug
  - Unless you forget…
Ternary operator

- An if statement does not return a value
  - Sometimes we want exactly that
- Enter the ternary operator
  - Book calls it the conditional operator
conditional ? expr1 : expr2;

- If the boolean returns true, return the result of expr1, else return result of expr2

- Similar to the following if but with a return.
  
  if (cond) expr1; else expr2;

- but with a return of the appropriate expr.
cout << "give me a file name";
cin >> name;
std::ostream &sout = name.empty() ?
    std::cout :
    ofstream(name);