Iterative Constructs

Mechanisms for deciding under what conditions an action should be repeated

Averaging
Determining Average Magnitude

- Suppose we want to calculate the average apparent brightness of a list of five star magnitude values
  - Can we do it?
    - Yes, it would be easy

- Suppose we want to calculate the average apparent brightness of a list of 8,479 stars visible from earth
  - Can we do it?
    - Yes, but it would be gruesome without the use of iteration

C++ Iterative Constructs

- Three constructs
  - while statement
  - for statement
  - do-while statement
While Syntax

Logical expression that determines whether the action is to be executed

while (Expression) Action

Action to be iteratively performed until logical expression is false

While Semantics

Expression

true

If Expression is true, Action is executed

false

Expression is evaluated at the start of each iteration of the loop

Action

If Expression is false, program execution continues with next statement
Computing an Average

```cpp
int listSize = 4;
int numberProcessed = 0;
double sum = 0;
while (numberProcessed < listSize) {
    double value;
    cin >> value;
    sum += value;
    ++numberProcessed;
}
double average = sum / numberProcessed;
cout << "Average: " << average << endl;
```

Suppose input contains: 1 5 3 1 6

Execution Trace

```cpp
int listSize = 4;
int numberProcessed = 0;
double sum = 0;
while (numberProcessed < listSize) {
    double value;
    cin >> value;
    sum += value;
    ++numberProcessed;
}
double average = sum / numberProcessed;
cout << "Average: " << average << endl;
```
int listSize = 4;
int numberProcessed = 0;
double sum = 0;
while (numberProcessed < listSize) {
    double value;
    cin >> value;
    sum += value;
    ++numberProcessed;
}
double average = sum / numberProcessed;
cout  << "Average: " << average << endl;
Suppose input contains: 1 5 3 1 6

Execution Trace

<table>
<thead>
<tr>
<th>listSize</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>numberProcessed</td>
<td>0</td>
</tr>
<tr>
<td>sum</td>
<td>0</td>
</tr>
</tbody>
</table>

```
int listSize = 4;
int numberProcessed = 0;
double sum = 0;
while (numberProcessed < listSize) {
    double value;
    cin >> value;
    sum += value;
    ++numberProcessed;
}
double average = sum / numberProcessed;
cout << "Average: " << average << endl;
```
Suppose input contains: 1 5 3 1 6

**Execution Trace**

<table>
<thead>
<tr>
<th>listSize</th>
<th>numberProcessed</th>
<th>sum</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

```c++
int listSize = 4;
int numberProcessed = 0;
double sum = 0;
while (numberProcessed < listSize) {
    double value;
    cin >> value;
    sum += value;
    ++numberProcessed;
}
double average = sum / numberProcessed;
cout << "Average: " << average << endl;
```
int listSize = 4;
int numberProcessed = 0;
double sum = 0;
while (numberProcessed < listSize) {
    double value;
    cin >> value;
    sum += value;
    ++numberProcessed;
}
double average = sum / numberProcessed;
cout << "Average: " << average << endl;
Suppose input contains: 1 5 3 1 6

Execution Trace

```
int listSize = 4;
int numberProcessed = 0;
double sum = 0;
while (numberProcessed < listSize) {
    double value;
    cin >> value;
    sum += value;
    ++numberProcessed;
}
double average = sum / numberProcessed;
cout << "Average: " << average << endl;
```
Suppose input contains: 1 5 3 1 6

Execution Trace

```
int listSize = 4;
int numberProcessed = 0;
double sum = 0;
while (numberProcessed < listSize) {
    double value;
    cin >> value;
    sum += value;
    ++numberProcessed;
}
double average = sum / numberProcessed ;
cout  << "Average: " << average << endl;
```
Suppose input contains: 1 5 3 1 6

Execution Trace

```
int listSize = 4;
int numberProcessed = 0;
double sum = 0;
while (numberProcessed < listSize) {
    double value;
    cin >> value;
    sum += value;
    ++numberProcessed;
}
double average = sum / numberProcessed;
cout << "Average: " << average << endl;
```

Suppose input contains: 1 5 3 1 6

Execution Trace

<table>
<thead>
<tr>
<th></th>
<th>maxSize</th>
<th>numberProcessed</th>
<th>sum</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>int listSize = 4;</td>
<td>int numberProcessed = 0;</td>
<td>double sum = 0;</td>
<td>while (numberProcessed &lt; listSize) { double value; cin &gt;&gt; value; sum += value; ++numberProcessed; } double average = sum / numberProcessed; cout &lt;&lt; &quot;Average: &quot; &lt;&lt; average &lt;&lt; endl;</td>
<td></td>
</tr>
</tbody>
</table>

Suppose input contains: 1 5 3 1 6

Execution Trace

<table>
<thead>
<tr>
<th></th>
<th>maxSize</th>
<th>numberProcessed</th>
<th>sum</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>int listSize = 4;</td>
<td>int numberProcessed = 0;</td>
<td>double sum = 0;</td>
<td>while (numberProcessed &lt; listSize) { double value; cin &gt;&gt; value; sum += value; ++numberProcessed; } double average = sum / numberProcessed; cout &lt;&lt; &quot;Average: &quot; &lt;&lt; average &lt;&lt; endl;</td>
<td></td>
</tr>
</tbody>
</table>
## Execution Trace

Suppose input contains: 1 5 3 1 6

<table>
<thead>
<tr>
<th>listSize</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>numberProcessed</td>
<td>value</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

```c++
int listSize = 4;
int numberProcessed = 0;
double sum = 0;

while (numberProcessed < listSize) {
    double value;
    cin >> value;
    sum += value;
    numberProcessed++;
}

double average = sum / numberProcessed;

cout << "Average: " << average << endl;
```
Suppose input contains: 1 5 3 1 6

Execution Trace

<table>
<thead>
<tr>
<th>listSize</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>numberProcessed</td>
<td>3</td>
</tr>
<tr>
<td>sum</td>
<td>9</td>
</tr>
<tr>
<td>value</td>
<td>1</td>
</tr>
</tbody>
</table>

```cpp
int listSize = 4;
int numberProcessed = 0;
double sum = 0;
while (numberProcessed < listSize) {
    double value;
    cin >> value;
    sum += value;
    ++numberProcessed;
}
double average = sum / numberProcessed;
cout << "Average: " << average << endl;
```
Suppose input contains: 15316

Execution Trace

```
int listSize = 4;
int numberProcessed = 0;
double sum = 0;
while (numberProcessed < listSize) {
    double value;
    cin >> value;
    sum += value;
    ++numberProcessed;
}
double average = sum / numberProcessed;
 cout << "Average: " << average << endl;
```
Suppose input contains: 15316

Execution Trace

```
int listSize = 4;
int numberOfProcessed = 0;
double sum = 0;
while (numberOfProcessed < listSize) {
    double value;
    cin >> value;
    sum += value;
    ++numberOfProcessed;
}
double average = sum / numberOfProcessed;
cout << "Average: " << average << endl;
```
Suppose input contains: 1 5 3 1 6

Execution Trace

int listSize = 4;
int numberProcessed = 0;
double sum = 0;
while (numberProcessed < listSize) {
    double value;
    cin >> value;
    sum += value;
    ++numberProcessed;
}
double average = sum / numberProcessed;
cout << "Average: " << average << endl;
Power of Two Table

```
const int TableSize = 20;

int i = 0;
long Entry = 1;

cout << "i" << "\t" << "2 ** i" << endl;

while (i < TableSize) {
    cout << i << "\t" << Entry << endl;
    Entry = 2 * Entry;
    ++i;
}
```

Better Way of Averaging

```
int numberProcessed = 0;
double sum = 0;
double value;
while (cin >> value) {
    sum += value;
    ++numberProcessed;
}
double average = sum / numberProcessed;
cout << "Average: " << average << endl;
```
Even Better Way of Averaging

```cpp
int numberProcessed = 0;
double sum = 0;
double value;
while ( cin >> value ) {
    sum += value;
    ++numberProcessed;
}
if ( numberProcessed > 0 ) {
    double average = sum / numberProcessed;
    cout << "Average: " << average << endl;
}
else {
    cout << "No list to average" << endl;
}
```

The For Statement

- Syntax
  
  ```cpp
  for (ForInit; ForExpression; PostExpression)
  Action
  ```

- Example
  
  ```cpp
  for (int i = 0; i < 3; ++i) {
      cout << "i is " << i << endl;
  }
  ```
ForExpr is evaluated at the start of each iteration of the loop. If ForExpr is true, Action is executed. After the Action has completed, the PostExpression is evaluated. If ForExpr is false, program execution continues with the next statement.

Execution Trace

```cpp
for (int i = 0; i < 3; ++i) {
    cout << "i is " << i << endl;
}
cout << "all done" << endl;
```
Execution Trace

for (int i = 0; i < 3; ++i) {
    cout << "i is " << i << endl;
}
cout << "all done" << endl;

i is 0
Execution Trace

for (int i = 0; i < 3; ++i) {
    cout << "i is " << i << endl;
}
cout << "all done" << endl;

i is 0

Execution Trace

for (int i = 0; i < 3; ++i) {
    cout << "i is " << i << endl;
}
cout << "all done" << endl;

i is 0
Execution Trace

```
for (int i = 0; i < 3; ++i) {
    cout << "i is " << i << endl;
}
cout << "all done" << endl;
```

```cpp
for (int i = 0; i < 3; ++i) {
    cout << "i is " << i << endl;
}
cout << "all done" << endl;
```

```
i 1

i is 0
i is 1
```
Execution Trace

for (int i = 0; i < 3; ++i) {
    cout << "i is " << i << endl;
}
cout << "all done" << endl;

i is 0
i is 1
Execution Trace

for (int i = 0; i < 3; ++i) {
    cout << "i is " << i << endl;
}
cout << "all done" << endl;

i is 0
i is 1

i is 2
Execution Trace

```cpp
for (int i = 0; i < 3; ++i) {
    cout << "i is " << i << endl;
}
cout << "all done" << endl;
```

```text
i is 0
i is 1
i is 2
```

Execution Trace

```cpp
for (int i = 0; i < 3; ++i) {
    cout << "i is " << i << endl;
}
cout << "all done" << endl;
```

```text
i is 0
i is 1
i is 2
```
Execution Trace

for (int i = 0; i < 3; ++i) {
    cout << "i is " << i << endl;
}
cout << "all done" << endl;

i is 0
i is 1
i is 2
Table Revisiting

```cpp
class MyClass {
public:
    MyClass() {
        const int TableSize = 20;
        long Entry = 1;
        cout << "i" << "\t" << "2*i" << endl;
        for (int i = 0; i <= TableSize; ++i) {
            cout << i << "\t" << Entry << endl;
            Entry *= 2;
        }
    }
};
```

Table Revisiting

```cpp
class MyClass {
public:
    MyClass() {
        const int TableSize = 20;
        long Entry = 1;
        cout << "i" << "\t" << "2*i" << endl;
        for (int i = 0; i < TableSize; ++i) {
            cout << i << "\t" << Entry << endl;
            Entry *= 2;
        }
    }
};
```

```
cout << "i is" << i << endl; // illegal
```

The scope of `i` is limited to the loop!
Displaying a Diagonal

SimpleWindow W("One diagonal", 5.5, 2.25);
W.Open();
for (int j = 1; j <= 3; ++j) {
    float x = j * 0.75 + 0.25;
    float y = j * 0.75 - 0.25;
    float Side = 0.4;
    RectangleShape S(W, x, y, Blue, Side, Side);
    S.Draw();
}

Sample Display
Displaying Three Diagonals

SimpleWindow W("Three diagonals", 6.5, 2.25);
W.Open();
for (int i = 1; i <= 3; ++i) {
    for (int j = 1; j <= 3; ++j) {
        float x = i - 1 + j * 0.75 + 0.25;
        float y = j * 0.75 - 0.25;
        float Side = 0.4;
        RectangleShape S(W, x, y, Blue, Side, Side);
        S.Draw();
    }
}

The scope of $i$ includes the inner loop.
The scope of $j$ is just the inner loop.

Sample Display
int Counter1 = 0;
int Counter2 = 0;
int Counter3 = 0;
int Counter4 = 0;
int Counter5 = 0;

++Counter1;
for (int i = 1; i <= 10; ++i) {
    ++Counter2;
    for (int j = 1; j <= 20; ++j) {
        ++Counter3;
    }
    ++Counter4;
}
++Counter5;
cout << Counter1 << " " << Counter2 << " " << Counter3 << " " << Counter4 << " " << Counter5 << endl;

---

For Into While

◆ Observation
  ■ The for statement is equivalent to
    {
        ForInit;
        while (ForExpression) {
            Action;
            PostExpression;
        }
    }
Counting Characters

```cpp
int NumberOfNonBlanks = 0;
int NumberOfUpperCase = 0;
char c;
while (cin >> c) {
    ++NumberOfNonBlanks;
    if ((c >= 'A') && (c <= 'Z')) {
        ++NumberOfUpperCase;
    }
}
cout << "Nonblank characters: " << NumberOfNonBlanks << endl << "Uppercase characters: " << NumberOfUpperCase << endl;
```

Counting All Characters

```cpp
char c;
int NumberOfCharacters = 0;
int NumberOfLines = 0;
while (cin.get(c)) {
    ++NumberOfCharacters;
    if (c == '\n') {
        ++NumberOfLines;
    }
}
cout << "Characters: " << NumberOfCharacters << endl << "Lines: " << NumberOfLines << endl;
```
```cpp
#include <iostream>
#include <fstream>
using namespace std;

int main() {
    ifstream fin("mydata.txt");
    int ValuesProcessed = 0;
    float ValueSum = 0;
    float Value;
    while ( fin >> Value ) {
        ValueSum += Value;
        ++ValuesProcessed;
    }
    if (ValuesProcessed > 0) {
        ofstream fout("average.txt");
        float Average = ValueSum / ValuesProcessed;
        fout << "Average: " << Average << endl;
        return 0;
    }
    else {
        cerr << "No list to average" << endl;
        return 1;
    }
}
```

**File Processing**

**Iteration Do’s**

◆ Key Points
   - Make sure there is a statement that will eventually terminate the iteration criterion
     - The loop must stop!
   - Make sure that initialization of loop counters or iterators is properly performed
   - Have a clear purpose for the loop
     - Document the purpose of the loop
     - Document how the body of the loop advances the purpose of the loop
The Do-While Statement

- **Syntax**
  
  ```
  do Action
  while (Expression)
  ```

- **Semantics**
  - Execute `Action`
  - If `Expression` is true then execute `Action` again
  - Repeat this process until `Expression` evaluates to false

- `Action` is either a single statement or a group of statements within braces

---

Waiting for a Proper Reply

```cpp
char Reply;
do {
    cout << "Decision (y, n): ";
    if (cin >> Reply)
        Reply = tolower(Reply);
    else
        Reply = 'n';
} while ((Reply != 'y') && (Reply != 'n'));
```