Control Constructs

Mechanisms for deciding when and how often an action should be taken
Boolean Algebra

- Logical expressions have the one of two values - true or false
  - A rectangle has three sides.
  - The instructor has a pleasant smile
- The branch of mathematics that deals with this type of logic is called Boolean algebra
  - Developed by the British mathematician George Boole in the 19th century
- Three key logical operators
  - And
  - Or
  - Not
Boolean Algebra

- Truth tables
  - Lists all combinations of operand values and the result of the operation for each combination
- Example

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>P and Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>
Boolean Algebra

- Truth table for or

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>P or Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>
**Boolean Algebra**

- Truth table for not

<table>
<thead>
<tr>
<th>P</th>
<th>Not P</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>
Boolean Algebra

- Can create complex logical expressions by combining simple logical expressions

- Example
  - not (P and Q)

- A truth table can be used to determine when a logical expression is true

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>P and Q</th>
<th>not (P and Q)</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>
A Boolean Type

- C++ contains a type named `bool`
- Type `bool` has two symbolic constants
  - `true`
  - `false`
- Boolean operators
  - The `and` operator is `&&`
  - The `or` operator is `||`
  - The `not` operator is `!`
- Warning
  - `&` and `|` are also operators
A Boolean Type

- Example logical expressions

```cpp
bool P = true;
bool Q = false;
bool R = true;
bool S = P && Q;
bool T = !Q || R;
bool U = !(R && !Q);
```
Relational Operators

- Equality operators
  - ==
  - !=

- Examples
  - int i = 32;
  - int k = 45;
  - bool q = i == k;
  - bool r = i != k;
Relational Operators

- Ordering operators
  - <
  - >
  - >=
  - <=

- Examples
  - int i = 5;
  - int k = 12;
  - bool p = i < 10;
  - bool q = k > i;
  - bool r = i >= k;
  - bool s = k <= 12;
Operator Precedence Revisited

- Precedence of operators (from highest to lowest)
  - Parentheses
  - Unary operators
  - Multiplicative operators
  - Additive operators
  - Relational ordering
  - Relational equality
  - Logical and
  - Logical or
  - Assignment
Operator Precedence Revisited

- Examples

\[ 5 \neq 6 \mid\mid 7 \leq 3 \]

\[ (5 \neq 6) \mid\mid (7 \leq 3) \]

\[ 5 \times 15 + 4 \equiv 13 \&\& 12 < 19 \mid\mid \text{false} \equiv 5 < 24 \]
Conditional Constructs

- Provide
  - Ability to control whether a statement list is executed
- Two constructs
  - If statement
    - If
    - If-else
    - If-else-if
  - Switch statement
The Basic If Statement

- Syntax
  \[ \text{if } (\text{Expression}) \]
  \[ \text{Action} \]

- If the \text{Expression} is true then execute \text{Action}

- \text{Action} is either a single statement or a group of statements within braces

- Example
  \[ \text{if } (\text{Value} < 0) \{ \]
  \[ \text{Value} = -\text{Value}; \]
  \[ \} \]

Diagram:
- \text{Expression}
- \text{true} \rightarrow \text{Action} \rightarrow \text{false}
cout << "Enter two integers: ";
int Value1;
int Value2;
cin >> Value1 >> Value2;
if (Value1 > Value2) {
    int RememberValue1 = Value1;
    Value1 = Value2;
    Value2 = RememberValue1;
}
cout << "The input in sorted order: "
    << Value1 << " " << Value2 << endl;
The If-Else Statement

- **Syntax**
  
  ```
  if (Expression)
    Action_1
  else
    Action_2
  ```

- **If** *Expression* is true then execute *Action_1* otherwise execute *Action_2*

```cpp
if (v == 0) {
    cout << "v is 0";
} else {
    cout << "v is not 0";
}
```
Finding the Larger of Two Values

cout << "Enter two integers: ";
int Value1;
int Value2;
cin >> Value1 >> Value2;
int Larger;
if (Value1 < Value2) {
    Larger = Value1;
}
else {
    Larger = Value2;
}
cout << "Larger of inputs is: " Larger << endl;
Selection

- It is often the case that depending upon the value of an expression we want to perform a particular action.
- Two major ways of accomplishing this choice:
  - If-else-If statement
    - If-else statements “glued” together
  - Switch statement
    - An advanced construct
The If-Else-If Statement

- Example

```cpp
if ((ch == 'a') || (ch == 'A'))
    cout << ch << " is a vowel" << endl;
else if ((ch == 'e') || (ch == 'E'))
    cout << ch << " is a vowel" << endl;
else if ((ch == 'i') || (ch == 'I'))
    cout << ch << " is a vowel" << endl;
else if ((ch == 'o') || (ch == 'O'))
    cout << ch << " is a vowel" << endl;
else if ((ch == 'u') || (ch == 'U'))
    cout << ch << " is a vowel" << endl;
else
    cout << ch << " is not a vowel" << endl;
```
Switch Statement

```cpp
switch (ch) {
    case 'a': case 'A':
    case 'e': case 'E':
    case 'i': case 'I':
    case 'o': case 'O':
    case 'u': case 'U':
        cout << ch << " is a vowel" << endl;
        break;
    default:
        cout << ch << " is not a vowel" << endl;
}
```
cout << "Enter simple expression: ";
int Left;
int Right;
char Operator;
cin >> Left >> Operator >> Right;
cout << Left << " " << Operator << " " << Right
<< " " << " = ";
switch (Operator) {
    case '+': cout << Left + Right << endl; break;
    case '-': cout << Left - Right << endl; break;
    case '*': cout << Left * Right << endl; break;
    case '/': cout << Left / Right << endl; break;
    default: cout << "Illegal operation" << endl;
}
Iterative Constructs

- Provide
  - Ability to control how many times a statement list is executed
- Three constructs
  - while statement
  - for statement
  - do-while statement
The While Statement

- **Syntax**
  
  \[
  \text{while (Expression)} \\
  \text{Action}
  \]

- **Semantics**
  - If *Expression* is true then execute *Action*
  - Repeat this process until *Expression* evaluates to false
  - *Action* is either a single statement or a group of statements within braces
const int TableSize = 20;

int i = 0;
long Value = 1;

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

while (i <= TableSize) {
    cout << i << "\t\t" << Value << endl;
    Value *= 2;
    ++i;
}
Character Counting

```cpp
int NumberOfNonBlanks = 0;
int NumberOfUpperCase = 0;
char c;
while (cin >> c) {
    ++NumberOfNonBlanks;
    if ((c >= 'A') && (c <= 'Z')) {
        ++NumberOfUpperCase;
    }
}
```
Counting Characters

```cpp
char c;
int NumberOfCharacters = 0;
int NumberOfLines = 0;
while (cin.get(c)) {
    ++NumberOfCharacters;
    if (c == '\n')
        ++NumberOfLines
}
cout << "Characters: " << NumberOfCharacters
    << endl;
cout << "Lines: " << NumberOfLines << endl;
```
int main() {
    cout << "Provide a list of numbers" << endl;
    int ListSize = 0;
    float ValueSum = 0;
    int Value;
    while (cin >> Value) {
        ValueSum += Value;
        ++ListSize;
    }
    if (ListSize > 0) {
        float Average = ValueSum / ListSize;
        cout << "Average: " << Average << endl;
    } else {
        cout << "No list to average" << endl;
    }
    return 0;
}
The For Statement

- **Syntax**
  \[
  \text{for} \ (\text{ForInit} \ ; \ \text{ForExpression} \ ; \ \text{PostExpression})
  \]
  
  \[\text{Action}\]

- **Semantics**
  - Execute \textit{ForInit} statement
  - While \textit{ForExpression} is true
    - Execute \textit{Action}
    - Execute \textit{PostExpression}

- **Example**
  
  \[
  \text{for} \ (\text{int} \ i = 0; \ i < 20; \ ++i) \{ \\
  \quad \text{cout} \ll "i is " \ll i \ll \text{endl}; \\
  \}
  \]
Iteration Using the For Statement

- ForInit
- ForExpression
  - true
  - Action
    - PostExpression
  - false
const int TableSize = 20;

long Value = 1;

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

for (int i = 0; i <= TableSize; ++i) {
    cout << i << " \t\t" << Value << endl;
    Value *= 2;
}

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

Three diagonals
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

```c
{ 
    ForInit; 
    while (ForExpression) { 
        Action; 
        PostExpression; 
    } 
}
```
Iteration

Key Points

- Make sure there is a statement that will eventually nullify the iteration criterion (i.e., the loop must stop)
- Make sure that initialization of any loop counters or iterators is properly performed
- Have a clear purpose for the loop
  - Document the purpose of the loop and how the body of the loop advances the purpose of the loop
Riddle

- Four hobos traveling across the country in need of money
- Farmer offers 200 hours of work that could be done over the next couple of weeks
- The laziest hobo convinces the other three hobos to draw straws
- Each straw would be marked with an amount
  - The amount would represent both the number of days and the numbers of hours per day that the hobo would work
  - Example
    - If the straw was marked three then the hobo who drew it would work for three hours per day for three days
- What are the best markings of the straws for a clever, lazy hobo?
Observations

- Need to find sets of whole numbers $a$, $b$, $c$, and $d$ such that
  - $a^2 + b^2 + c^2 + d^2 = 200$
- Maximal legal number is 14 as $15^2$ equals 225 which is greater than 200
- Minimal legal number is 1
- No advantage to listing the combinations more than once
  - Implication
    - Generate the solutions systematically
  - We will make sure that $a \leq b \leq c \leq d$
Method

- Generate all possibilities for a where for each a possibility
  - Generate all possibilities of b where for each b possibility
    - Generate all possibilities for c where for each c possibility
  - Generate all possibilities for d where for each d possibility
    - Determine whether the current combination is a solution
for (int a = 1; a <= 14; ++a) {
    for (int b = a; b <= 14; ++b) {
        for (int c = b; c <= 14; ++c) {
            for (int d = c; (d <= 14); ++d) {
                if (a*a + b*b + c*c + d*d == 200) {
                    cout << a << " " << b << " " << c << " " << d << endl;
                }
            }
        }
    }
}
Simple Visualization

- What statements can we make about the following data set?
  4.90 2.41 0.82 0.77 2.60 5.10 7.52 9.45 9.65
  7.81 5.04 2.51 0.95 0.80 2.62
- Statistical analysis indicates that observations come from interval 0 … 10 with an average value of 4.97 and a standard deviation of 2.95
- Another approach is to detect whether the sequence of observations represents a pattern
  ■ Are the numbers arranged for example in Fibonacci order?
- If no patterns are recognized, try data visualization
  ■ Plot the data set values in a two-dimensional manner
    – y-axis correspond to data set values
    – x-axis correspond to positions in the data set sequence
```cpp
#include <iostream>  // Program 4.12
#include <string>
#include "rect.h"
using namespace std;

int ApiMain() {
    const float Unit = 0.25;
    cout << "Enter size of data set: ";
    int n;
    cin >> n;
    SimpleWindow W("Data set display", n+2, 10);
    W.Open();
    for (float x = 1; x <= n; ++x) {
        cout << "Enter data value (n): ";
        float y;
        cin >> y;
        RectangleShape Point(W, x, y, Blue, Unit, Unit); Point.Draw();
    }
    return 0;
}
```
Sample Run

- Data values do have structure
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
char Reply;
do {
    cout << "Decision (y, n): ";
    if (cin >> Reply)
        Reply = tolower(Reply);
    else
        Reply = 'n';
} while ((Reply != 'y') && (Reply != 'n'));}