Until now, statements were all simple statements. Here we will introduce structured statements. First of all, a compound statement is a sequence of 2 or more consecutive statements (simple, compound, and/or structured) enclosed within { and }.

I. Selection Constructs

- if
- if/else
- nested if
- switch/case

**If statement**

In pseudocode (structured English) we might express a decision as:

*If student’s grade is greater than or equal to 60*  
*Print “Passed”*

In C++ this would be coded as:

```cpp
if (grade >= 60)  
    cout << "Passed";
```

Syntax for if statement:
```
if <boolean expression> <statement1> ;
```

A boolean expression represents a condition that is either true or false. It is formed using operands (constants, variables) and operators (arithmetic operators, relational operators, logical operators).

<table>
<thead>
<tr>
<th>Arithmetic</th>
<th>Relational</th>
<th>Logical</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ -</td>
<td>&lt; less than &gt; greater than</td>
<td></td>
</tr>
<tr>
<td>* / %</td>
<td>== equal to != not equal to</td>
<td>&amp;&amp; and</td>
</tr>
<tr>
<td>arithmetic functions</td>
<td>&lt;= less than or equal to &gt;= greater than or equal to</td>
<td>! not</td>
</tr>
</tbody>
</table>

Ex. if ((x > 0) && (a == b)) ...
### Operator Precedence

<table>
<thead>
<tr>
<th>Operator</th>
<th>Associativity</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>( )</td>
<td>L → R</td>
<td>parenthesis</td>
</tr>
<tr>
<td>++ – – + !</td>
<td>R → L</td>
<td>unary operators</td>
</tr>
<tr>
<td>* ? %</td>
<td>L → R</td>
<td>multiplicative</td>
</tr>
<tr>
<td>+ –</td>
<td>L → R</td>
<td>additive</td>
</tr>
<tr>
<td>&lt;&lt; &gt;&gt;</td>
<td>L → R</td>
<td>insertion</td>
</tr>
<tr>
<td>&lt; &lt;= &gt; &gt;=</td>
<td>L → R</td>
<td>relational</td>
</tr>
<tr>
<td>== !=</td>
<td>L → R</td>
<td>equality</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>L → R</td>
<td>and</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>?:</td>
<td>R → L</td>
<td>conditional</td>
</tr>
<tr>
<td>= += -= *= /= %=</td>
<td>R → L</td>
<td>assignment</td>
</tr>
</tbody>
</table>
if/else statement

In pseudocode:
If student’s grade is greater than or equal to 60
    Print “Passed”
else
    Print “Failed”

In C++

```cpp
if (grade >= 60)
    cout << “Passed”;
else
    cout << “Failed”;
```

syntax for if/else statement:
if <boolean expression> <statement1> [else <statement2>];

The C++ conditional operator - ?:

Same as the if/else structure but more compact. This is the only ternary operator in C++. Example:

```cpp
grade>=60? cout << “Passed” : cout << “Failed”;
```

This is the same as:
cout << (grade >=60? “Passed” : “Failed”);

In this case, the conditional expression evaluates to “Passed” if the condition is true and to “Failed “ if the condition is false.
//maximum.cpp
// Finding the largest of 3 values

#include <iostream>

int main()
{
    int A, B, C, max;

    cout << "Enter 3 values, and I will tell you which is largest.\n";
    cin >> A >> B >> C; //really should have user prompts here
    if (A > B)
        max = A;
    else
        max = B;
    if (C > max)
        max = C;
    cout << "The maximum is " << max << endl;
    return 0; //successful termination
} //end main

RUN #1:

Enter 3 values, and I will tell you which is the largest.
12 7 2
The maximum is 12

RUN#2:

Enter 3 values, and I will tell you which is the largest.
6 5 9
The maximum is 9
Nested-if

//nestedIf.cpp
// modified from Hubbard Ex. 2.14
// This program converts a test score into its equivalent letter grade.

#include <iostream>
int main()
{
    int score;
    cout << "Enter the test score: ";
    cin >> score;
    if (score > 100) cout << "Error: score is out of range.";
    else if (score >= 90) cout << 'A';
    else if (score >= 80) cout << 'B';
    else if (score >= 70) cout << 'C';
    else if (score >= 60) cout << 'D';
    else if (score >= 0) cout << 'F';
    else cout << "Error: score is out of range.";

    return 0;  //successful termination
}  //end main

Why does this work? EXPLAIN.
Switch

//switch.cpp
// modified from Hubbard Ex. 2.15
// This program converts a test score into
// its equivalent letter grade, using a switch statement.

#include <iostream>
int main()
{
    int score;
    cout << "Enter the test score: ";
    cin >> score;
    switch (score/10) {
        case 10: case 9: cout << 'A' << endl; break;
        case 8: cout << 'B' << endl; break;
        case 7: cout << 'C' << endl; break;
        case 6: cout << 'D' << endl; break;
        case 5: case 4: case 3: case 2: case 1: case 0:
            cout << 'F' << endl; break;
        default: cout << "Error: score is out of range.\n";
    }
    return 0;    //successful termination
}    //end main

How does this work? The computer "drops down" through the "empty" conditions. EXPLAIN.
In the following example, the function `rand()` is defined in the header file `<stdlib>`. `rand()` generates pseudo-random unsigned integers in the range of 0 to `RAND_MAX`. `RAND_MAX` is a constant which is also defined in `<stdlib>`.
II. Iteration (Repetition) Constructs
In order to control iteration, we use one of three structured control statements.
   • for
   • while
   • do/while

A Counting Program: We would like to write a program that will print the integers between 1 and 20, say, like the following output:

Here’s one way:

```cpp
//counting2.cpp
// A Counting Program
// Using a FOR/DO Loop

#include <iostream>
using namespace std;
int main()
{
    int count ;

    for (count=1; count<=20; count++)
        cout << count << endl;
    return 0;   //successful termination
} //end main
```
Here’s another way:

```cpp
//counting.cpp
// A Counting Program
// Using a WHILE/DO Loop

#include <iostream>
using namespace std;

int main()
{
    int count;

    count = 1;
    while (count <= 20)
    {
        cout << count << endl;
        count = count + 1;
    }
    return 0;  //successful termination
} //end main
```

And yet another way:

```cpp
//counting3.cpp
// A Counting Program
// Using a DO/WHILE Loop

#include <iostream>
using namespace std;

int main()
{
    int count;

    count = 1;
    do {
        cout << count << endl;
        count = count + 1;
    } while (count <= 20);

    return 0;  //successful termination
} //end main
```
A Summing Program

We wish to write a program that will calculate the sum of the integers between 1 and 20.

The program:

```cpp
//sum1.cpp
// A Summing Program
// Using a WHILE/DO Loop

#include <iostream>
using namespace std;

int main()
{
    int sum, count;
    sum = 0;
    count = 1;
    while (count <= 20){
        sum = sum + count;
        count = count + 1;
    }
    cout << "The sum of the integers from 1 through 20 is ";
    cout << sum << endl;
    return 0;   //successful termination
} //end main
```

produces this output:

```
The sum of the integers from 1 through 20 is 210

Press any key to continue...
```

and so does this program:

```cpp
//sum2.cpp
// A Summing Program
// Using a FOR Loop

#include <iostream>
using namespace std;

int main()
{
    int sum, count;
    sum = 0;

    for (count=1; count<=20; count++)
    {
        sum = sum + count;
        cout << "The sum of the integers from 1 through 20 is ";
        cout << sum << endl;
    }
    return 0;   //successful termination
} //end main
```
Just an Average Program ;-)
When to use the while construct? When to use for?
Improving our Output

Setw is a stream, insertion manipulator that sets the field width of the output. It requires the iomanip.h header file, i.e.: 
\#include <iomanip>

The following program illustrates how it works.

```cpp
//columns.cpp
// Some Calculations
// Prettier output using stream manipulators

#include <iostream>
#include <iomanip>

int main()
{
    int number, square, cube;

    cout << setw(6) << "NUMBER" << setw(8) << "SQUARE";
    cout << setw(8) << "CUBE" << endl << endl;
    for (number=1; number <=20; number++){
        square = number * number;
        cube = square * number;
        cout << setw(6) << number << setw(8) << square;
        cout << setw(8) << cube << endl;
    } //end for
    return 0; //successful termination of program
} //end main
```

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>SQUARE</th>
<th>CUBE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>27</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>64</td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td>125</td>
</tr>
<tr>
<td>6</td>
<td>36</td>
<td>216</td>
</tr>
<tr>
<td>7</td>
<td>49</td>
<td>343</td>
</tr>
<tr>
<td>8</td>
<td>64</td>
<td>512</td>
</tr>
<tr>
<td>9</td>
<td>81</td>
<td>729</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>1000</td>
</tr>
<tr>
<td>11</td>
<td>121</td>
<td>1331</td>
</tr>
<tr>
<td>12</td>
<td>144</td>
<td>1728</td>
</tr>
<tr>
<td>13</td>
<td>169</td>
<td>2197</td>
</tr>
<tr>
<td>14</td>
<td>196</td>
<td>2744</td>
</tr>
<tr>
<td>15</td>
<td>225</td>
<td>3375</td>
</tr>
<tr>
<td>16</td>
<td>256</td>
<td>4096</td>
</tr>
<tr>
<td>17</td>
<td>289</td>
<td>4913</td>
</tr>
<tr>
<td>18</td>
<td>324</td>
<td>5832</td>
</tr>
<tr>
<td>19</td>
<td>361</td>
<td>6859</td>
</tr>
<tr>
<td>20</td>
<td>400</td>
<td>8000</td>
</tr>
</tbody>
</table>
Branching -
Break and Continue Statements

Conditional vs. unconditional branching, goto
These statements should be used only when absolutely necessary.

**Break** - to break out of a loop. Transfers control to the first statement after the end of the loop.

Example. What is output?

```cpp
#include <iostream>
using namespace std;

int main()
{
    int x;
    for (x=1; x<=10; x++)
    {
        if (x==5)
            break;    //break loop only if x is 5
        cout << x << " ";
    }    //end for
    cout << "\nBroke out of loop at x = " << x << endl;
    return 0;    //successful termination of program
}    //end main
```

Press any key to continue
**CONTROLLING EXECUTION - SELECTION & ITERATION**

**Continue** - skips the remainder of the body of the loop, and continues the loop. Transfers control to “loop again.”

Example. What is output?

```cpp
// Deitel2-26.cpp
// This program is modified from the Deitel book, Figure 2.27, p.105.
// Using the Continue statement in a for structure

#include <iostream>
using namespace std;

int main() {
    int x;
    for (x=1; x<=10; x++) {
        if (x==5)
            continue;     // skip remaining code in loop
            // only if x is 5
        cout << x << " ";
    } //end for
    cout << "\nUsed continue to skip printing the value 5 " << endl;
    cout << endl << endl;
    return 0; //successful termination of program
} //end main
```

```
1 2 3 4 6 7 8 9 10
Used continue to skip printing the value 5

Press any key to continue
```
Random Numbers
Using switch, break, for, mod

//TestRandNum.cpp
// This program gets 1000 random numbers and tests them
#include <iostream>
#include <stdlib>

int main(){
    int num0, num1, num2, num3, num4, num5, num6, num7, num8, num9;
    num0=num1=num2=num3=num4=num5=num6=num7=num8=num9=0;
    for (int i=0; i<1000; i++) {
        int r = rand()%10;  //remainder gives a one digit RN
        switch (r) {
            case 0: num0++; break;  //Note: This program will be more
            case 1: num1++; break;  //efficient when we learn arrays.
            case 2: num2++; break;
            case 3: num3++; break;
            case 4: num4++; break;
            case 5: num5++; break;
            case 6: num6++; break;
            case 7: num7++; break;
            case 8: num8++; break;
            case 9: num9++; break;
            default: cout << "Error: unexpected random number" << r <<endl;
        } //end switch
    } //end for
    cout << "number of 0 : " << num0 << endl;
    cout << "number of 1 : " << num1 << endl;
    cout << "number of 2 : " << num2 << endl;
    cout << "number of 3 : " << num3 << endl;
    cout << "number of 4 : " << num4 << endl;
    cout << "number of 5 : " << num5 << endl;
    cout << "number of 6 : " << num6 << endl;
    cout << "number of 7 : " << num7 << endl;
    cout << "number of 8 : " << num8 << endl;
    cout << "number of 9 : " << num9 << endl;
    return 0;   //successful termination
} //end main