THE INCREMENT AND DECREMENT OPERATORS

++, --

• ++ is the increment operator.
  • It adds one to a variable.
  • val++; is the same as val = val + 1;

• ++ can be used before (prefix) or after (postfix) a variable:
  • ++val;
  • val++;

• -- is the decrement operator.
  • It subtracts one from a variable.
  • val--; is the same as val = val - 1;

• -- can be also used before (prefix) or after (postfix) a variable:
  • --val;
  • val--;
The Increment and Decrement Operators

**IN PROGRAM**

Program 5-1

```cpp
// This program demonstrates the ++ and -- operators.
#include <iostream>
using namespace std;

int main()
{
    int num = 4; // num starts out with 4.
    // Display the value in num.
    cout << "The variable num is " << num << endl;
    cout << "I will now increment num.\n\n";

    // Use postfix ++ to increment num.
    num++;    
    cout << "How the variable num is " << num << endl;
    cout << "I will increment num again.\n\n";

    // Use prefix ++ to increment num.
    ++num;
    cout << "How the variable num is " << num << endl;
    cout << "I will now decrement num.\n\n";

    // Use postfix -- to decrement num.
    num--;
    cout << "How the variable num is " << num << endl;
    cout << "I will decrement num again.\n\n";
}
```

**Program Output**

The variable num is 4
I will now increment num.
Now the variable num is 5
I will increment num again.
Now the variable num is 6
I will now decrement num.
Now the variable num is 5
I will decrement num again.
Now the variable num is 4
PREFIX VS. POSTFIX

• ++ and -- operators can be used in complex statements and expressions
• In prefix mode (++val, --val)
  • the operator increments or decrements, then returns the value of the variable
• In postfix mode (val++, val--)
  • the operator returns the value of the variable, then increments or decrements

```cpp
int num, val = 12;
cout << val++; // displays 12,
               // val is now 13;
cout << ++val; // sets val to 14,
               // then displays it
num = --val; // sets val to 13,
             // stores 13 in num
num = val--; // stores 13 in num,
             // sets val to 12
```

NOTES ON INCREMENT AND DECREMENT

• Can be used in expressions:
  • result = num1++ + --num2;

• Must be applied to something that has a location in memory. Cannot have:
  • result = (num1 + num2)++;

• Can be used in relational expressions:
  • if (++num > limit)
  • pre- and post-operations will cause different comparisons
**THE WHILE LOOP**

**INTRODUCTION**

- **Loop**: a control structure that causes a statement or statements to repeat
- General format of the `while` loop:
  ```
  while (expression)
  statement;
  ```
- `statement;` can also be a block of statements enclosed in `{ }`

```
while (expression)
    statement;
```
- `expression` is evaluated
  - if `true`, then `statement` is executed, and `expression` is evaluated again
  - if `false`, then the loop is finished and program statements following `statement` execute

**THE WHILE LOOP**

**IN PROGRAM**

**Program 5.3**

```cpp
1 // This program demonstrates a simple while loop.
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7     int number = 1;
8     while (number <= 5)
9     {
10        cout << "Hello\n";
11        number++;
12     }
13     cout << "That's all!\n";
14     return 0;
15 }
```

**Program Output**

Hello
Hello
Hello
Hello
Hello
That's all!
**The While Loop**

**Line 9 to 13**

Test this expression.

```c++
while (number <= 5) {
    cout << "Hello\n";
    number++;
}
```

If the expression is true, perform these statements.

After executing the body of the loop, start over.

---

**The While Loop**

```
number <= 5
```

True

Print "Hello"

Add 1 to `number`

False
**THE WHILE LOOP**

- *expression* is evaluated *before* the loop executes. The following loop will never execute:

```cpp
int number = 6;
while (number <= 5) {
    cout << "Hello\n";
    number++;
}
```

**WATCH OUT FOR INFINITE LOOPS**

- The loop must contain code to make *expression* become *false*
- Otherwise, the loop will have no way of stopping
- Such a loop is called an *infinite loop*, because it will repeat an infinite number of times

```cpp
int number = 1;
while (number <= 5) {
    cout << "Hello\n";
}
```
THE \textbf{WHILE LOOP} \hfill \textbf{INPUT VALIDATION}

\begin{itemize}
\item Input validation is the process of inspecting data that is given to the program as input and determining whether it is valid.
\item The while loop can be used to create input routines that reject invalid data, and repeat until valid data is entered.
\item Here's the general approach, in pseudocode:
\end{itemize}

\begin{quote}
\textit{Read an item of input.}
\textit{While the input is invalid}
\hspace{1em}\textit{Display an error message.}
\hspace{1em}\textit{Read the input again.}
\textit{End While}
\end{quote}

\textbf{THE \textbf{WHILE LOOP}} \hfill \textbf{EXAMPLE}

\begin{verbatim}
cout << "Enter a number less than 10: ";
cin >> number;
while (number >= 10)
{
    cout << "Invalid Entry!" << "Enter a number less than 10: ";
    cin >> number;
}
\end{verbatim}
THE **WHILE LOOP**

**IN PROGRAM**

```cpp
// Get the number of players per team.
cout << "How many players do you wish per team? ";
cin >> teamPlayers;
// Validate the input.
while (teamPlayers < MIN_PLAYERS || teamPlayers > MAX_PLAYERS)
{
    // Explain the error.
    cout << "You should have at least " << MIN_PLAYERS
        << " but no more than " << MAX_PLAYERS << " per team.\n";
    // Get the input again.
    cout << "How many players do you wish per team? ";
cin >> teamPlayers;
}
// Get the number of players available.
cout << "How many players are available? ";
cin >> players;
// Validate the input.
while (players <= 0)
{
    // Get the input again.
    cout << "Please enter 0 or greater. ";
cin >> players;
}
```

**COUNTERS**

- **Counter**: a variable that is incremented or decremented each time a loop repeats
- Can be used to control execution of the loop (also known as the **loop control variable**)
- Must be initialized before entering loop

**Program 5-6**

```cpp
// This program displays a list of numbers and
// their squares.
#include <iostream>
using namespace std;

int main()
{
    const int MIN_NUMBER = 1;  // Starting number to square
    const int MAX_NUMBER = 10; // Maximum number to square
    int num = MIN_NUMBER;      // Counter
    cout << "Number Number Squared\n";
    cout << "------------------------\n";
```
THE DO-WHILE LOOP

- **do-while**: a posttest loop – execute the loop, then test the expression

- **General Format**: 
  ```
  do 
  statement; // or block in { } 
  while (expression);
  ```

- Note that a semicolon is required after `(expression)`
THE DO-WHILE LOOP

AN EXAMPLE

```cpp
int x = 1;
do {
    cout << x << endl;
} while(x < 0);
```

• Although the test expression is false, this loop will execute one time because do-while is a posttest loop.

THE DO-WHILE LOOP

IN PROGRAM

Program 5.7

```cpp
1 // This program averages 3 test scores. It repeats as
2 // many times as the user wishes.
3 #include <iostream>
4 using namespace std;
5
6 int main()
7 {
8     int score1, score2, score3; // Three scores
9     double average;          // Average score
10     char again;             // To hold Y or N input

11 do
12 { // Get three scores.
13     cout << "Enter 3 scores and I will average them: \"; 
14     cin >> score1 >> score2 >> score3;
15     // Calculate and display the average.
16     average = (score1 + score2 + score3) / 3.0;
17     cout << "The average is: \" << average << \".\n\";
18     // Do you want to average another set?
19     cout << "Do you want to average another set? (Y/N) \";
20     cin >> again;
21     while (again == 'Y' || again == 'y');
22 }
23 return 0;
24 }
```
THE DO-WHILE LOOP
IN PROGRAM

Program Output with Example Input Shown in Bold
Enter 3 scores and I will average them: 80 90 70 [Enter]
The average is 80.
Do you want to average another set? (Y/N) y [Enter]
Enter 3 scores and I will average them: 60 75 88 [Enter]
The average is 74.3333.
Do you want to average another set? (Y/N) n [Enter]

THE FOR LOOP

• Useful for counter-controlled loop

• General Format:

  for(initialization; test; update)
  statement; // or block in { }

• No semicolon after the update expression or after the )
THE **FOR LOOP**

**MECHANICS**

```plaintext
for(initialization; test; update)
    statement; // or block in { }
```

1. Perform *initialization*  
2. Evaluate *test* expression  
   1. If true, execute *statement*  
   2. If false, terminate loop execution  
3. Execute *update*, then re-evaluate *test* expression
THE FOR LOOP

EXAMPLE

int count;

for (count = 1; count <= 5; count++)
    cout << "Hello" << endl;

THE FOR LOOP

FLOWCHART

Assign 1 to count

count <= 5

True

cout statement

Increment count

False
THE FOR LOOP

IN PROGRAM

Program 5.9

```cpp
// This program displays the numbers 1 through 10 and their squares.
#include <iostream>
using namespace std;

int main()
{
    const int MIN_NUMBER = 1, // Starting value
              MAX_NUMBER = 10; // Ending value
    int num;

    cout << "Number Number Squared\n";
    cout << "-----------------------------\n";
    for (num = MIN_NUMBER; num <= MAX_NUMBER; num++)
        cout << num << "\t" << (num * num) << endl;
    return 0;
}
```

THE FOR LOOP

PROGRAM OUTPUT

Program Output

<table>
<thead>
<tr>
<th>Number</th>
<th>Number Squared</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>25</td>
</tr>
<tr>
<td>6</td>
<td>36</td>
</tr>
<tr>
<td>7</td>
<td>49</td>
</tr>
<tr>
<td>8</td>
<td>64</td>
</tr>
<tr>
<td>9</td>
<td>81</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
</tr>
</tbody>
</table>
**THE FOR LOOP**

**STEPS**

**Step 1:** Perform the initialization expression.

**Step 2:** Evaluate the test expression. If it is true, go to Step 3. Otherwise, terminate the loop.

```
for (num = MIN_NUMBER; num <= MAX_NUMBER; num++)
    cout << num << "\t" << (num * num) << endl;
```

**Step 3:** Execute the body of the loop.

**Step 4:** Perform the update expression, then go back to Step 2.

---

**THE FOR LOOP**

**FLOWCHART**

- **Assign MIN_NUMBER to num**
- **num <= MAX_NUMBER**
- **True**
  - Display num and num * num
  - Increment num
- **False**
WHEN TO USE THE FOR LOOP

• In any situation that clearly requires
• an initialization
• a false condition to stop the loop
• an update to occur at the end of each iteration

THE FOR LOOP IS A PRETEST LOOP

• The for loop tests its test expression before each iteration, so it is a pretest loop.
• The following loop will never iterate:

```cpp
for (count = 11; count <= 10; count++)
    cout << "Hello" << endl;
```
FOR LOOP - MODIFICATIONS

- You can have multiple statements in the \textit{initialization} expression. Separate the statements with a comma:

```c++
int x, y;
for (x=1, y=1; x <= 5; x++)
{
    cout << x << " plus " << y
         << " equals " << (x+y)
         << endl;
}
```

- You can also have multiple statements in the \textit{test} expression. Separate the statements with a comma:

```c++
int x, y;
for (x=1, y=1; x <= 5; x++, y++)
{
    cout << x << " plus " << y
         << " equals " << (x+y)
         << endl;
}
```
FOR LOOP - MODIFICATIONS

- You can omit the *initialization* expression if it has already been done:

```c
int sum = 0, num = 1;
for (; num <= 10; num++)
    sum += num;
```

- You can declare variables in the *initialization* expression:

```c
int sum = 0;
for (int num = 0; num <= 10; num++)
    sum += num;
```

The scope of the variable `num` is the `for` loop.
**KEEPING A RUNNING TOTAL**

- **running total**: accumulated sum of numbers from each repetition of loop
- **accumulator**: variable that holds running total

```c
int sum=0, num=1; // sum is the
while (num <= 10) // accumulator
{   sum += num;
    num++;
}
cout << "Sum of numbers 1 - 10 is"
<< sum << endl;
```

**KEEPING A RUNNING TOTAL**

**FLOWCHART**

```
1. Set accumulator to 0
2. Is there a number to read?
   - Yes (True)
     1. Read the number
     2. Add the number to the accumulator
   - No (False)

Diagram:
```
```
**Keeping a Running Total**

**Program 5-12**

```cpp
1 // This program takes daily sales figures over a period of time
2 // and calculates their total.
3 #include <iostream>
4 #include <iomanip>
5 using namespace std;
6
7 int main()
8 {
9    int days;       // Number of days
10    double total = 0.0; // Accumulator, initialized with 0
11    // Get the number of days.
12    cout << "For how many days do you have sales figures? ";
13    cin >> days;
14    // Get the sales for each day and accumulate a total.
15    for (int count = 1; count <= days; count++)
16    {
17        double sales;
18        cout << "Enter the sales for day " << count << " ";
19        cin >> sales;
20        total += sales; // Accumulate the running total.
21    }
22
23    // Display the total sales.
24    cout << fixed << showpoint << setprecision(2);
25    cout << "The total sales are $" << total << endl;
26    return 0;
27 }
```

**Keeping a Running Total**

**Output**

```plaintext
// Display the total sales.
cout << fixed << showpoint << setprecision(2);
cout << "The total sales are $" << total << endl;
return 0;
}
```

Program Output with Example Input Shown in Bold

For how many days do you have sales figures? 5 [Enter]
Enter the sales for day 1: 489.32 [Enter]
Enter the sales for day 2: 421.65 [Enter]
Enter the sales for day 3: 497.89 [Enter]
Enter the sales for day 4: 532.37 [Enter]
Enter the sales for day 5: 506.92 [Enter]
The total sales are $2448.15

Lecture 05 - Loops
SENTINELS

- **sentinel**: value in a list of values that indicates end of data
- Special value that cannot be confused with a valid value, e.g., -999 for a test score
- Used to terminate input when user may not know how many values will be entered
SENTINELS

OUTPUT

Program Output with Example Input Shown in Bold

Enter the number of points your team has earned so far in the season, then enter -1 when finished.

Enter the points for game 1: 7 [Enter]
Enter the points for game 2: 9 [Enter]
Enter the points for game 3: 4 [Enter]
Enter the points for game 4: 6 [Enter]
Enter the points for game 5: 8 [Enter]
Enter the points for game 6: -1 [Enter]

The total points are 34

DECIDING WHICH LOOP TO USE

• The while loop is a conditional pretest loop
  • Iterates as long as a certain condition exits
  • Validating input
  • Reading lists of data terminated by a sentinel

• The do-while loop is a conditional posttest loop
  • Always iterates at least once
  • Repeating a menu

• The for loop is a pretest loop
  • Built-in expressions for initializing, testing, and updating
  • Situations where the exact number of iterations is known
NESTED LOOPS

- A nested loop is a loop inside the body of another loop
- Inner (inside), outer (outside) loops:

```cpp
for (row=1; row<=3; row++)  //outer
    for (col=1; col<=3; col++)  //inner
        cout << row * col << endl;
```

NESTED LOOPS IN PROGRAM

```cpp
// Determine each student's average score.
for (int student = 1; student <= numStudents; student++)
{
    total = 0;  // Initialize the accumulator.
    for (int test = 1; test <= numTests; test++)
    {
        double score;
        cout << "Enter score " << test << " for ":
        cin >> score;
        total += score;
    }
    average = total / numTests;
    cout << "The average score for student " << student << " is " << average << ".\n\n";
}
```
NESTED LOOPS

NOTES

• Inner loop goes through all repetitions for each repetition of outer loop

• Inner loop repetitions complete sooner than outer loop

• Total number of repetitions for inner loop is product of number of repetitions of the two loops.