Topic 4
Loops

Department of Engineering Physics
University of Gaziantep

Course web page
www.gantep.edu.tr/~bingul/ep241

Sep 2013
Introduction

Loops are control structures that cause a program to repeat (iterate) a block of code.

This lecture covers the following topics:

- The `while` loop structure
- The `do..while` loop structure
- The `for` loop structure
- The `break` and `continue` statements
- Infinite loops
- Nested loops
- Solved problems
while loop structure

The **while** loop has the general form:

```c
while (condition) {
  statements
  .
  .
}
```

Here the block of statements is executed while `condition` is **true**.

Note that `condition` is tested at the **start** of the loop.
This program calculates the series sum: \(1 + 2 + 3 + 4 + 5 + \ldots + n\).

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

int main() {
    cout << "Input n: ";
    int n;
    cin >> n;
    int k=1, s=0;
    while (k<=n) {
        s = s + k;
        k++;
    }
    cout << "The series sum is "
         << s << endl;
}
```

Output

```
Input n: 8
The series sum is 36
```

Note that on the first iteration of the loop, \(k=1\) and on the final execution \(k=n\).
do..while loop structure

The do..while loop has the general form:

```java
    do {
      statements
    } while (condition);
```

Here the block of statements is executed while `condition` is true. Note that `condition` is tested at the end of the loop.
This program calculates the product: 1 * 2 * 3 * 4 * 5 * .... * n.

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

int main() {
    cout << "Input n: ";
    int n;
    cin >> n;
    int k=1, f=1;
    do{
        f = f * k;
        k++;
    }while(k<=n);
    cout << "The product is "
         << f << endl;
}
```

Output

```
Input n: 4
The product is 24
```
**for loop structure**

The **for** statement allows you to execute a block of code a specified number of times.

The general form is:

```cpp
for (initialisation; condition; increment) {
    statements
    .
    .
}
```

Example program section:

```cpp
for (int i=1; i<=5; i++) {
    cout << i << " " << i*i << endl;
}
```

**Output**

```
1  1
2  4
3  9
4 16
5 25
```
Declare counter $i$ as type `int` and initialise it to 1

Repeat while counter $i$ is less than or equal to 5

Increment counter $i$ by 1 at the end of each iteration

```cpp
for ( int i=1; i<=5; i++ ) {
    cout << i << endl;
}
```

Output

```
1
2
3
4
5
```
This program calculates the series sum: \(1 + 1/2 + 1/4 + 1/8 + 1/16 + \ldots + 1/2^n\)

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

int main() {
    cout << "Input n: ";
    int n;
    cin >> n;
    int s=0;
    for (int k=0; k<=n; k++) {
        s = s + 1.0/pow(2.0,k);
    }
    cout << "The series sum is " << s << endl;
}
```

Output

Input n: 30
The series sum is 2
Problem 1: Mean of numbers

Draw a flowchart and write a program to find and output the mean of n real numbers \((x_i, i = 1,2,\ldots,n)\) defined by

\[
\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i
\]

where \(n\) is the input from keyboard:
In a Compton Scattering experiment, X-rays of wavelength $\lambda = 10$ pm are scattered from a target. Write a program to find the wavelength in pm of the x-rays scattered through the angle $\theta$ for the range from $0^\circ$ to $180^\circ$.


$$\lambda' - \lambda = \frac{\hbar}{m_e c} (1 - \cos \theta)$$

where

- $\lambda$ is the initial wavelength,
- $\lambda'$ is the wavelength after scattering,
- $\hbar$ is the Planck constant,
- $m_e$ is the rest mass of the electron,
- $c$ is the speed of light, and
- $\theta$ is the scattering angle.
The constant $\frac{\hbar}{m_e c}$ is called the Compton Wavelength and has the value 2.426 pm.
<table>
<thead>
<tr>
<th>Angle Theta in Deg</th>
<th>Lambda2 (pm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>10.0369</td>
</tr>
<tr>
<td>20</td>
<td>10.1463</td>
</tr>
<tr>
<td>30</td>
<td>10.325</td>
</tr>
<tr>
<td>40</td>
<td>10.5676</td>
</tr>
<tr>
<td>50</td>
<td>10.8666</td>
</tr>
<tr>
<td>60</td>
<td>11.213</td>
</tr>
<tr>
<td>70</td>
<td>11.5963</td>
</tr>
<tr>
<td>80</td>
<td>12.0047</td>
</tr>
<tr>
<td>90</td>
<td>12.426</td>
</tr>
<tr>
<td>100</td>
<td>12.8473</td>
</tr>
<tr>
<td>110</td>
<td>13.2557</td>
</tr>
<tr>
<td>120</td>
<td>13.639</td>
</tr>
<tr>
<td>130</td>
<td>13.9854</td>
</tr>
<tr>
<td>140</td>
<td>14.2844</td>
</tr>
<tr>
<td>150</td>
<td>14.527</td>
</tr>
<tr>
<td>160</td>
<td>14.7057</td>
</tr>
<tr>
<td>170</td>
<td>14.8151</td>
</tr>
<tr>
<td>180</td>
<td>14.852</td>
</tr>
</tbody>
</table>
Problem 3: Proper Divisors

Write a program that reads a positive integer, $k$, and outputs its proper divisors. For example, if $k = 28$ then the proper divisors are: 1 2 4 7 14 28

Solution will be given in the lecture.
Jump Statements: `break` and `continue`

// break statement
#include <iostream>
using namespace std;
int main()
{
    double x;
    for(int i = -3; i<=3; i++)
    {
        if(i==0) break;
        x = 1.0/i;
        cout << x << endl;
    }
}

-0.3333
-0.5
-1

// continue statement
#include <iostream>
using namespace std;
int main()
{
    double x;
    for(int i = -3; i<=3; i++)
    {
        if(i==0) continue;
        x = 1.0/i;
        cout << x << endl;
    }
}

-0.3333
-0.5
-0.5
1
0.5
0.3333
Infinite loops

If the *condition* of a loop is always *true*, then the loop will iterate *infinitely*, i.e. it will loop forever!

```
while ( true ) {
    cout << "infinite loop!" << endl;
}
```

```
while ( 1 ) {
    cout << "infinite loop!" << endl;
}
```

```
do {
    cout << "infinite loop!" << endl;
} while ( 7>3 );
```

```
for ( ; ; ) {
    cout << "infinite loop!" << endl;
}
```

It is sometimes useful to create infinite loops like these, but with the addition of a *condition* for breaking out of the loop.

A “break out” can be achieved with the *break* statement together with an *if* structure.....
Example use of the `break` statement in an infinite loop

This program continually inputs values and outputs their reciprocal.

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

int main() {
    while( 1 ) {
        cout << "Input x: ";
        double x;
        cin >> x;
        if ( x==0. ) break;
        cout << "The reciprocal is " << 1/x << endl;
    }
    cout << "Bye." << endl;
}
```

The program terminates when the input is zero.

**Output**

```
Input x: 34.2
The reciprocal is 0.0292398
Input x: 0.8
The reciprocal is 1.25
Input x: 3.4
The reciprocal is 0.294118
Input x: 3.0
The reciprocal is 0.333333
Input x: 0.2
The reciprocal is 5
Input x: 0
Bye.
```
Nested loops

Nested loops are *loops within loops*

Nested **while** loops

```plaintext
while ( condition1 ) {
    statements1
    while ( condition2 ) {
        statements2
    }
    statements3
}
```
Nested for loops

```java
for ( i=0; i<n; i++ ) {
    statements1
    for ( j=0; j<m; j++ ) {
        statements2
    }
}
```

*statements1* is repeated \( n \) times
*statements2* is repeated \( n \times m \) times
i.e. there are \( n \times m \) iterations of the nested loop.
Problem 4: Multiplication Table

In this example variable \( i \) loops over \textit{rows} and \( j \) loops over \textit{columns}.

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

int main() {
    for ( int i=1; i<=8; i++ ) {
        for ( int j=1; j<=6; j++ ) {
            cout << i*j << "\t";
        }
        cout << endl;
    }
    return 0;
}
```

The "\t" (tab) \textit{escape sequence} is injected into the output stream to improve formatting.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>9</td>
<td>12</td>
<td>15</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
<td>20</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>25</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>18</td>
<td>24</td>
<td>30</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td>21</td>
<td>28</td>
<td>35</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>16</td>
<td>24</td>
<td>32</td>
<td>40</td>
<td>48</td>
<td></td>
</tr>
</tbody>
</table>
Problem 5:

Write a program to list and count all integer pairs \((x, y)\) satisfying the inequality:

\[ |x| + |y| \leq 3 \]

Solution will be given in the lecture.
Homeworks

1. Using a do while loop, write a program that evaluates and outputs first 300 terms the following series:
   \[ \frac{1}{2} + \frac{2}{3} + \frac{3}{4} + \frac{4}{5} + \frac{5}{6} + \frac{6}{7} + \ldots \]

2. Using a for loop, write a program that evaluates and outputs first 300 terms the following series:
   \[ \frac{1}{2} - \frac{2}{3} + \frac{3}{4} - \frac{4}{5} + \frac{5}{6} - \frac{6}{7} + \ldots \]

3. Modify the Compton Scattering example program to output also maximum kinetic energy of the electrons corresponding to the angle \( \theta \) in the experiment.

4. A ray traveling parallel to the principal axis strike a convex mirror whose radius of curvature is \( R = 25 \text{ cm} \) as shown in Figure. Write a program to output the reflection angles (\( \theta \)) for the rays at distances \( h \) in the range \([0, 4 \text{ cm}]\) with step 0.1 cm.
5. A perfect number, \( p \), is a positive integer number whose sum of divisors excluding \( p \) itself equal to \( p \). For example, \( p = 6 \) is a perfect number since \( 6 = 1 + 2 + 3 \). Write a program to list and count all perfect numbers less than 10,000.

6. A positive integer \( n \) is called a prime number if it has only two positive divisors (1 and \( n \)). The first few primes are: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, ... Write a program to list and count all prime numbers less than 10,000.

7. Write a program that reads a positive integer number \( m \) and then prints the english name of each digit of that number in a single line. For example, if \( m = 147 \) the program should output: one four seven.

8. Write a program that lists all leap years between 1812 and 2012.

9. Write a program that finds and outputs all integer pairs \((x, y)\) satisfying the inequality: \( |2x| + |3y| < 10 \). Use two nested while loops.