1. Introduction

This lecture covers the following topics:

- Relational and logical operators
- Boolean expressions
- The `if` structure
- The `if .. else` structure
- The `if .. else if .. else` structure
- The `while` loop structure
- The `do..while` loop structure
- The `for` loop structure
- The `break` and `continue` statements
- Infinite loops
- Nested loops
- Solved problems
2. Relational Operators

Control statements use *relation operators* to compare two objects. There are six relational operators as follows:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>less than</td>
<td>$x &lt; y$</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
<td>$x \leq y$</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
<td>$x &gt; y$</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
<td>$x \geq y$</td>
</tr>
<tr>
<td>==</td>
<td>equal</td>
<td>$x = y$</td>
</tr>
<tr>
<td>!=</td>
<td>not equal to</td>
<td>$x \neq y$</td>
</tr>
</tbody>
</table>

Example:

```
if ( b != 0 ) c = a/b;
```

control structure using a relational operator

The result of a relational operation is either *true* or *false*.

The assignment of $c$ in the selection structure
```
if ( b != 0 ) c = a/b;
```
occurs only if $(b \neq 0)$ is *true*.

Example program section:
```
double x=1.3, y=2.7, c=0.;
if (x > y) cout << "x is greater than y.";
if (y > 0.) cout << x/y << endl;
if (x+y != 0.) c = 1/(x+y);
cout << "c = " << c << endl;
```

Output
```
0.481481
0.25
```

Note that there is no output from the second line because the relation $(x > y)$ is *false*.
3. Logical Operators

*Compound* relation expressions can be formed by *logical operators*:

<table>
<thead>
<tr>
<th>Logical Operators</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;&amp;</td>
<td>logical AND, conjunction. Both sides must be true for the result to be true</td>
<td>(x &gt; 2 \text{ &amp;&amp; } y == 3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>logical NOT, negation</td>
<td>(! (x&gt;0))</td>
</tr>
</tbody>
</table>

Example:

```plaintext
if (b != 0 && a > 0) c = a/b;
```

*control structure using a compound relational operator*

Results for the && and || operators:

| X      | Y    | X && Y | X || Y |
|--------|------|--------|--------|
| true   | true | true   | true   |
| true   | false| false  | true   |
| false  | true | false  | true   |
| false  | false| false  | false  |

```plaintext
if (b != 0 && a > 0) c = a/b;
```
4. Boolean Expressions

Expressions that evaluate to true or false are called Boolean.

We can form Boolean expressions inside control statements (previous page) or in the form of assignments as follows:

```c
int x=1, y=2, s;
bool u, z = true, t, w;
u = x > 3;
z = x <= y && y > 0;
t = y <= 0 || z;
w = !s;
s = 2 > 1;
```

Note that variables u, z, t, and w are declared as type bool and so can represent the states true and false.

Also literal constants true and false can be used in assignments and relational operations.

Results

- u = false since 1>3 is false.
- z = true since 1<=2 and 2>0 are both true.
- t = true since z is true.
- w = false since s is true, therefore its negation is false.
- s = 1 = true since 2>1 (integer representation! see next).

5. if structure

The if statement allows conditional execution; the general form is:

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

If condition is true then the block defined by the braces {...} is executed.

```c
if (x+y != 0.) {
    c = 1/(x+y);
    cout << "c = " << c << endl;
}
```

If statements is a single statement then the braces can be omitted:

```c
if (condition)
    single-statement
```

Variable c is assigned only if the condition is true. But, the output statement will be executed in any case.
6. **if .. else structure**

The `if..else` structure allows both outcomes of a selection to be defined.

The general form is:

```plaintext
if (condition) {
    statements1
    .
    .
} else {
    statements2
    .
    .
}
```

If `condition` is true then the first block is executed, otherwise (false) the second block is executed.

```plaintext
if ( x+y != 0. ) {
    c = 1/(x+y);
    cout << "c = " << c << endl;
} else {
    cout << "c is undefined! " << endl;
}
```

---

7. **if .. else if .. else structure**

More levels of selection can be added with the `else if` statement.

Add as many blocks as you need.

This is executed if none of the above conditions are true.

```plaintext
if (condition1) {
    .
    statements1
    .
} else if (condition2) {
    .
    statements2
    .
} else if (condition3) {
    .
    statements3
    .
} else {
    .
    statements4
    .
}
```
Consider the quadratic equation:

\[ f(x) = ax^2 + bx + c \]

The roots are the values of \( x \) such that \( f(x) = 0 \).

Analytical solution:

\[ x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \]

Three cases for the result \( b^2 > 4ac \):

i) \( b^2 > 4ac \) there are two roots.

ii) \( b^2 = 4ac \) there is one root.

iii) \( b^2 < 4ac \) the roots are imaginary.

Examples we can use these results to validate our program

i) \((x-4)(x+2) = 0\) when \( x = 4, x = -2 \)

\[ f(x) = x^2 - 2x - 8 \]

\[ a = 1, b = -2, c = -8 \]

\[ x = 2/2 \pm \sqrt{36}/2 \]

\[ = 1 \pm 3 = 4 \text{ and } -2 \]

ii) \((x-2)(x-2) = 0\) when \( x = 2 \)

\[ f(x) = x^2 - 4x + 4 \]

\[ a = 1, b = -4, c = -4 \]

\[ x = 4/2 \pm \sqrt{0}/2 \]

\[ = 2 \]

Write a computer program that inputs the coefficients \( a, b, c \) of a quadratic equation, and outputs the root(s).

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

int main() {
    double a, b, c;
    cin >> a >> b >> c;

    double Delta = b*b - 4*a*c;

    if ( Delta < 0. ) {
        cout << "The roots are imaginary!" << endl;
    } else if ( Delta == 0. ) {
        double x1 = -b / (2*a);
        cout << "The root is " << x1 << endl;
    } else {
        double x1 = ( -b - sqrt(Delta) ) / (2*a);
        cout << "The two roots are " << x1 << " and " << x2 << endl;
    }
}
```
Example: Composite functions

Consider the composite function:

\[ f(x) = \begin{cases} 
2 & \text{for } x < 3 \\
\frac{2x}{3} & \text{for } 3 \leq x < 6 \\
4 & \text{for } x \geq 6
\end{cases} \]

Write a program that inputs a value for \( x \) and outputs the corresponding value of \( f(x) \).

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

int main() {
    double x, f;
    cout << "input x: " ;
    cin >> x;
    if (x < 3.0) f = 2.0;
    else if (x < 6.0) f = 2.0/3.0*x;
    else f = 4.0;

    cout << "f(\( x \)) = " << f << endl;

    return 0;
}
```

Example outputs:

- input x: 0
  f(0) = 2
- input x: 1
  f(1) = 2
- input x: 2
  f(2) = 2
- input x: 3
  f(3) = 2
- input x: 4
  f(4) = 2.66667
- input x: 5
  f(5) = 3.33333
- input x: 6
  f(6) = 4
- input x: 7
  f(7) = 4
8. switch Statement

This is an alternative for the if .. else if .. else structure. General form:

```
switch (expression)
{
    case constant1:
        group of statements 1;
        break;
    case constant2:
        group of statements 2;
        break;
    .
    .
    .
    default:
        default group of statements;
}
```

```cpp
int classCode;
cin >> classCode;
switch(classCode)
{
    case 1:
        cout << "Freshman\n";
        break;
    case 2:
        cout << "Sophomore\n";
        break;
    case 3:
        cout << "Junior\n";
        break;
    case 4:
        cout << "Graduate\n";
        break;
    default:
        cout << "bad code\n";
}
```

```cpp
int classCode;
cin >> classCode;
if(classCode==1)
    cout << "Freshman\n";
else if(classCode==2)
    cout << "Sophomore\n";
else if(classCode==3)
    cout << "Junior\n";
else if(classCode==4)
    cout << "Graduate\n";
else
    cout << "bad code\n";
```
9. ? Operator
The ? operator (conditional expression operator) provides a concise form of the if .. else structure.

The general form is:

\[( \text{condition} ) \ ? \ \text{expression1} : \ \text{expression2} ; \]

The value produced by this operation is either expression1 or expression2 depending on condition being true or false.

Example:

\[ \text{max} = (x > y) \ ? \ x : y; \]

is equivalent to

\[
\begin{align*}
\text{if} \ (x > y) \\
\text{max} &= x; \\
\text{else} \\
\text{max} &= y;
\end{align*}
\]

10. while loop structure
The while loop has the general form:

\[
\begin{align*}
\text{while} \ (\text{condition}) \ {\{} \\
\text{statements} \\
\cdots \\
\{ \\
\end{align*}
\]

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 \(k=n\).

11. do..while loop structure

The `do..while` loop has the general form:

```
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 ";
    cout << f << endl;
}
```

**Output**

```
Input n: 4
The product is 24
```

---

12. **for** loop structure

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

The general form is:

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

This program calculates the series sum:
1 + 1/2 + 1/4 + 1/8 + 1/16 + ..... + 1/2^n

#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;
}
Example: Compton Scattering


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 $180^\circ$.

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

where

- $\lambda$ is the initial wavelength,
- $\lambda'$ is the wavelength after scattering,
- $h$ 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.

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

int main()
{
    double lambda1, lambda2, theta;
    // compton wavelength in pm
    const double cw = 2.426;

    lambda1 = 10.0; // pm

    for(int deg=0; deg<=180; deg +=10)
    {
        theta = deg * M_PI/180.0;
        lambda2 = lambda1 + cw*(1.0-cos(theta));
        cout << deg << "t" << lambda2 << endl;
    }
}
```

Sayfa 25

Sayfa 26
### 13. Jump Statements

**// 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;
    }
}
```

**// 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;
    }
}
```
14. Infinite loops

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

```cpp
while ( true ) {
    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.

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

A “break out” can be achieved with the break statement together with an if structure.....

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

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

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.
15. Nested loops

Nested loops are *loops within loops*

Nested *while* loops

```java
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.
Example: Nested Loop

In this example variable $i$ loops over rows and $j$ loops over 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) escape sequence is injected into the output stream to improve formatting.

Output

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

Example: Cost Minimization

Consider a box with open top to carry $V = 0.2 \text{ m}^3$ waste water. The cost of material used to form the box is $C_m = 10 \text{ TL/m}^2$ and welding cost is $C_w = 5 \text{ TL/m}$. Design the box so that its total cost is minimum. Verify the result analytically.

Solution will be given in lecture.
Homeworks

Solve the following problems. You have to prepare a pdf document and sent it to me until next lecture.
E-mail: bingul[at]gantep.edu.tr (replace [at] with @)

1. Write a program to input an integer number and output whether it is even or not. Use the % operator.

2. Write a program that reads a grade A, B, C, D, or F and then prints "excellent", "good", "fair", "poor", or "failure". Use the switch statement.

3. Write a program to input coefficients of a quadratic equation of the form: \( ax^2 + bx + c = 0 \) and output the roots of the equation for all possible the cases: real roots, complex roots and \( a = 0 \). Examples:
   * a=1, b=0, c=-4 \( \Rightarrow \) \( x_1 = 2.0 \) and \( x_2 = -2.0 \)
   * a=0, b=4, c=-2 \( \Rightarrow \) \( x_1 = x_2 = 0.5 \)
   * a=1, b=1, c=1 \( \Rightarrow \) \( x_1 = -0.5\pm0.866i \) and \( x_2 = -0.5+0.866i \)

4. A leap year is a year in which one extra day (February 29) is added to the regular calendar. Most of us know that the leap years are years that are divisible by 4. For example 1992 and 1996 are leap years. But this rule does not work generally. For example centennial years are not leap years. For example 1800 and 1900 are not leap years.
   A year is called the leap year if
   * it is divisible by 4 and but not divisible by 100
   * or it is divisible by 400
   Write a program that reads a year and outputs whether it is leap year or not.
5. Using a for loop, write a program that evaluates and outputs first 300 terms the following series:

\[
\frac{1}{2} - \frac{2}{3} + \frac{3}{4} - \frac{4}{5} + \frac{5}{6} - \frac{6}{7} + \ldots
\]

6. Write a program that reads a positive integer, \(k\), and outputs its proper divisors. Use a do while loop.

For example, for \(k = 28\), the proper divisors are: 1 2 4 7 14 28

7. Write a program that finds and outputs all integer pairs \((x, y)\) satisfying the inequality: \(|2x| + |3y| < 10\).

Use two nested for loops.

8. The figure shows the cross section of a channel carrying water.

Determine \(h\), \(b\) and \(\theta\) that minimize the length of the wetted perimeter while maintaining a cross-sectional area of 6 m\(^2\).

(Minimizing the wetted perimeter results in least resistance to the flow.)

\[\text{Hint:} \]

Use three nested loops to search for the \(h\), \(b\) and \(\theta\) minimizing the circumference.
9. In Optics, in an ideal optical system, all rays of light from a point in the object plane would converge to the same point (called focal point) in the image plane, forming a clear image. The influences which cause different rays to converge to different points are called aberrations. Spherical aberrations occur because the focal points of rays far from the principal axis of a spherical lens (or mirror) are different from the focal points of rays of the same wavelength passing near the axis.

Figure shows a monochromatic light ray falling on a plano-convex lens whose radius of curvature is $R = 20.0$ cm, thickness is $x = 1.0$ cm and refractive index is $n = 1.4$. The distance between parallel ray and the principle axis of the lens is $y$. (a) Write a program to evaluate the focal length ($f$) of the lens as a function of the position $y$. You should evaluate and output the value of $f$ in a loop for the variable $y$ whose range is between 0 and 12 cm with step 0.1 cm. The result that you will obtain can explain the spherical aberration in a lens. (b) Using a graphic program, plot the values of $f$ as a function of $y$.

See also: http://www1.gantep.edu.tr/~bingul/ep118/docs/ep118-lec09-aberrations.pdf