EP578 Computing for Physicists

Topic 3

Selection & Loops

Department of
Engineering Physics

University of Gaziantep

Course web page
`www.gantep.edu.tr/~bingul/ep578`

Oct 2011

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:

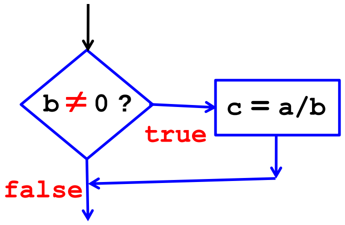
Relational Operators

Operator	Description	Example
<	less than	x < y
<=	less than or equal to	x <= y
>	greater than	x > y
>=	greater than or equal to	x >= y
==	equal to	x == y
!=	not equal to	x != y

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!=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;
```

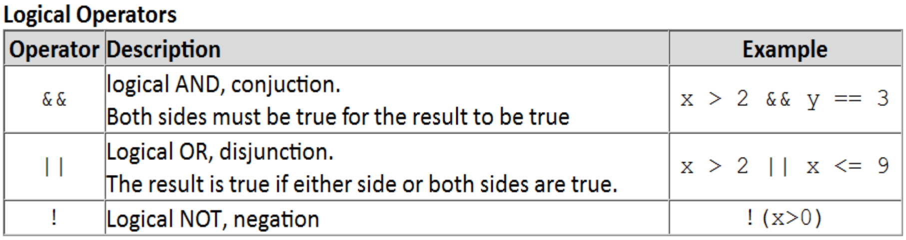
Output

```
0.481481
c = 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*:



Logical Operators

Operator	Description	Example
`&&`	logical AND, conjuction. Both sides must be true for the result to be true	`x > 2 && y == 3`
`||`	Logical OR, disjunction. The result is true if either side or both sides are true.	`x > 2 || x <= 9`
`!`	Logical NOT, negation	`!(x>0)`

Example:

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

control structure using a
compound relational operator

Results for the `&&` and `||` operators:

X	Y	X && Y (AND)	X || Y (OR)
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

```
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:

```
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:

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

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

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

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

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

```
if ( x+y != 0. )
  c = 1/(x+y);

cout << "c = " << c << endl;
```

single-statement `if` structure

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:

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

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

```
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.

```
if (condition1) {
   .
   statements1
   .
} else if (condition2) {
   .
   statements2
   .
} else if (condition3) {
   .
   statements3
   .
} else {
   .
   statements4
   .
}
```

Example: Quadratic Roots

Consider the quadratic equation:

$$f(x) = a\ x^2 + b\ x + c$$

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) $(x\text{-}4)(x+2) = 0$
when $\underline{x = 4}$, $\underline{x = -2}$

$f(x) = x^2 - 2\ x - 8$
$a = 1,\ b = -2,\ c = -8$

$x = 2/2 \pm sqrt(36)/2$
$= 1 \pm 3 = \underline{4}$ and $\underline{-2}$

ii) $(x\text{-}2)(x\text{-}2) = 0$
when $x=2$

$f(x) = x^2 - 4\ x + 4$
$a = 1,\ b = -4,\ c = -4$

$x = 4/2 \pm sqrt(0)/2 = \underline{2}$

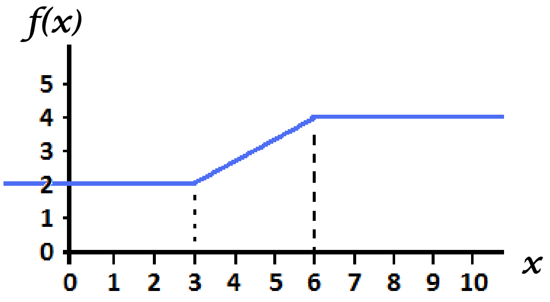
Write a computer program that inputs the coefficients a, b, c of a quadratic equation, and outputs the root(s).

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

```
#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);
    double x2 = ( -b + sqrt(Delta) ) / (2*a);
    cout << "The two roots are "<< x1<< " and " << x2<< endl;

  }
}
```

Example: Composite functions

Consider the composite function:



$$
\begin{aligned}
f(x) &= 2 && \text{for} && x < 3\\
f(x) &= 2x/3 && \text{for} && 3 \le x \le 6\\
f(x) &= 4 && \text{for} && x > 6
\end{aligned}
$$

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

$$
\begin{aligned}
f(x) &= 2 && \text{for} && x < 3\\
f(x) &= 2x/3 && \text{for} && 3 \le x < 6\\
f(x) &= 4 && \text{for} && x \ge 6
\end{aligned}
$$

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

int main() {

  double x, f;

  cout << "input x: ";
  cin >> x;

  if       ( x < 3. ) f = 2.0;
  else if ( x < 6. ) 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;
}
```

```
int classCode;
cin >> classCode;

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

```
int classCode;
cin >> classCode;


  if(classCode==1){
        cout << "Freshman\n";
  }
  else if(classCode==2){
        cout << "Sophmore\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:

(*condition*) ? ***expression1*** : ***expression2***;

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

Example:

```
max = ( x > y ) ? x : y;
```

is equivalent to

```
if ( x > y )
  max = x;
else
  max = y;
```

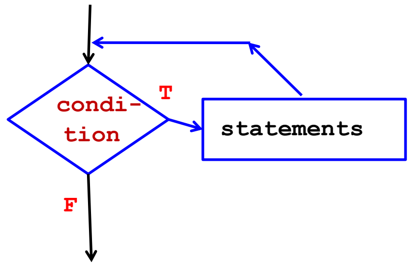
10. `while` loop structure

The `while` loop has the general form:

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

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

Note that `condition` is tested at the <u>start</u> of the loop.



This program calculates the series sum: 1 + 2 + 3 + 4 + 5 + ... + n.

```
#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`.

---

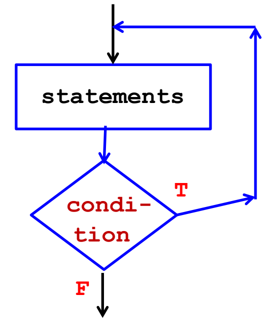
11. `do..while` loop structure

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.

```
#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
```

---

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:

```
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

```
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 + ... + $1/2^n$

```
#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;
}
```

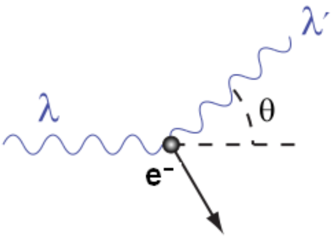
Output

```
Input n: 30
The series sum is 2
```

Example: Compton Scattering

http://en.wikipedia.org/wiki/Compton_scattering

In a Compton Scattering experiment, X-rays of wavelength λ = 10 pm are scattered from a target. Write a program to find the wavelength in pm of the x-rays scattered through the angle θ for the range from 0^o to 180^o.



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

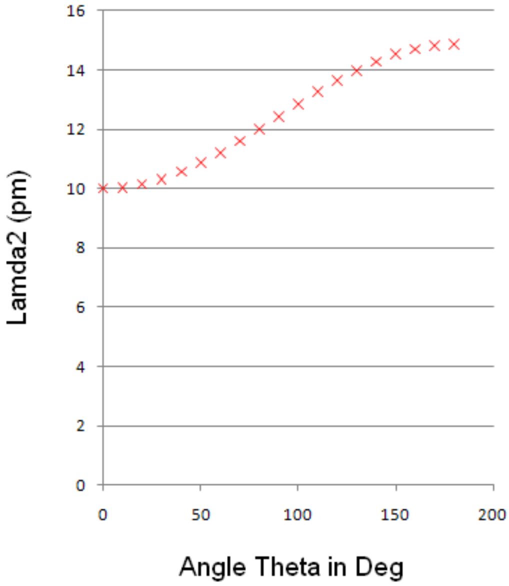
where

λ is the initial wavelength,
λ' 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
θ is the scattering angle.

```
#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;
  }
}
```

Output

```
0           10
10          10.0369
20          10.1463
30          10.325
40          10.5676
50          10.8666
60          11.213
70          11.5963
80          12.0047
90          12.426
100         12.8473
110         13.2557
120         13.639
130         13.9854
140         14.2844
150         14.527
160         14.7057
170         14.8151
180         14.852
```



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;
  }
}
```

```
-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
-1
1
0.5
0.3333
```

14. 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.

```
#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

```
while ( condition1 ) {
   statements1
   while ( condition2 ) {
      statements2
   }
   statements3
}
```

Nested `for` loops

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

`statements1` is repeated `n` times
`statements2` is repeated `n`×`m` times

i.e. there are `n`×`m` iterations of the nested loop.

Example: Nested Loop

In this example variable `i` loops over *rows* and `j` loops over *columns*.

```
#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;
}
```

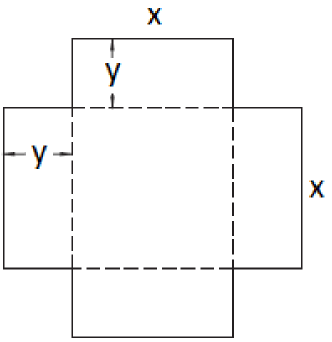
The `"\t"` (tab) *escape sequence* is injected into the output stream to improve formatting.

Output

```
1       2       3       4       5       6
2       4       6       8       10      12
3       6       9       12      15      18
4       8       12      16      20      24
5       10      15      20      25      30
6       12      18      24      30      36
7       14      21      28      35      42
8       16      24      32      40      48
```

Example: Cost Minimization

Consider a box with open top to carry V = 0.2 m^3 waste water. The cost of material used to form the box is C_m = 10 TL/m^2 and welding cost is C_w = 5 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 ==> $x_1 = 2.0$ and $x_2 = -2.0$
   * a=0, b=4, c=-2 ==> $x_1 = x_2 = 0.5$
   * a=1, b=1, c=1 ==> x_1= -0.5-0.866i and x_2 = -0.5+0.866i

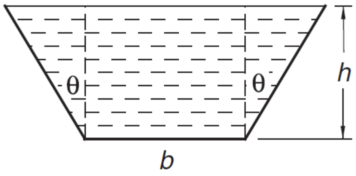
4. A leap year is a year in which one extra day (February 29) is added to the regular calendar. Most of us know that the leap years are years that are divisible by 4. For example 1992 and 1996 are leap years. But this rule does not work generally. For example centennial years are not leap years. For 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:
   `1/2 - 2/3 + 3/4 - 4/5 + 5/6 - 6/7 + ...`

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 *θ* 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.)



Hint:

Use three nested loops to search for the *h*, *b* and *θ* 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.

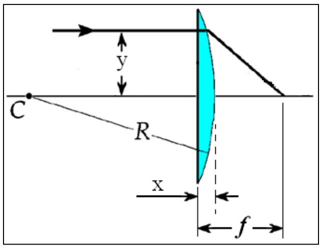


Figure shows a monochromatic light ray falling on a plano-convex lens whose radius of curvature is R = 20.0 cm, thickness is x = 1.0 cm and refractive index is n = 1.4. The distance between parallel ray and the principle axis of the lens is y. (a) Write a program to evaluate the focal length (f) of the 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