FLOW OF CONTROL
Flow of Control

- Sequential flow of control
  - Statement in a program are normally executed one after another.

- Often it is desirable to alter the sequential flow of control to provide for
  - a choice of action
    - if, if-else, switch
  - or a repetition of action
    - while, for, do
Relational, Equality, and Logical Operators

- **true**: nonzero value
- **false**: zero value

### Operator precedence and associativity

<table>
<thead>
<tr>
<th>Operator</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>() ++ (postfix)</td>
<td>left to right</td>
</tr>
<tr>
<td>-- (postfix)</td>
<td>left to right</td>
</tr>
<tr>
<td>+ (unary) - (unary)</td>
<td>right to left</td>
</tr>
<tr>
<td>++ (prefix) -- (prefix) !</td>
<td>right to left</td>
</tr>
<tr>
<td>* / %</td>
<td>left to right</td>
</tr>
<tr>
<td>+ -</td>
<td>left to right</td>
</tr>
<tr>
<td>&lt; &lt;= &gt; &gt;=</td>
<td>left to right</td>
</tr>
<tr>
<td>== !=</td>
<td>left to right</td>
</tr>
<tr>
<td>&amp; &amp;</td>
<td>left to right</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>?:</td>
<td>right to left</td>
</tr>
<tr>
<td>= += -= *= /= etc.</td>
<td>right to left</td>
</tr>
<tr>
<td>, (comma operator)</td>
<td>left to right</td>
</tr>
</tbody>
</table>
Relational Operators and Expressions

\[
\begin{align*}
\text{expr} & < \text{expr} \\
\text{expr} & > \text{expr} \\
\text{expr} & \leq \text{expr} \\
\text{expr} & \geq \text{expr}
\end{align*}
\]

**Examples**
- \(a < 3\)
- \(a > b\)
- \(-1.3 \geq (2.0 \times x + 3.3)\)

**NOT Examples**
- \(a =< b\) /* out of order */
- \(a = = b\) /* space not allowed */
- \(a >> b\) /* shift expression */

- **a < b**
  - If \(a\) is less than \(b\), then the \(\text{expr}\). has the \(\text{int}\) value 1 (true).
  - If \(a\) is not less than \(b\), then the \(\text{expr}\). has the \(\text{int}\) value 0 (false).
Relational Operators and Expressions

- **Arithmetic conversion**
  - On many machines, \( a < b \) is implemented as \( a - b < 0 \).

### Declarations and Initializations

```
char    c = 'w';
int      i = 1, j = 2, k = -7;
double  x= 7e+33, y = 0.001
```

<table>
<thead>
<tr>
<th>Expression</th>
<th>Equivalent expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>'a' + 1 &lt; c</td>
<td>('a' + 1) &lt; c</td>
<td>1</td>
</tr>
<tr>
<td>- i - 5 * j &gt;= k + 1</td>
<td>((- i) - (5 * j)) &gt;= (k + 1)</td>
<td>0</td>
</tr>
<tr>
<td>3 &lt; j &lt; 5</td>
<td>(3 &lt; j) &lt; 5</td>
<td>1</td>
</tr>
<tr>
<td>x - 3.333 &lt;= x + y</td>
<td>(x - 3.333) &lt;= (x + y)</td>
<td>1</td>
</tr>
<tr>
<td>x &lt; x + y</td>
<td>x &lt; (x + y)</td>
<td>0</td>
</tr>
</tbody>
</table>

\( 3 < j \land j < 5 \iff (3 < j) \land (j < 5) \)

\( x < x + y \)
\( (x - (x + y)) < 0.0 \)

The values of \( x \) and \( x + y \) are equal, so the expr. will yield the int value 0.
Equality Operators and Expressions

\[ expr \equiv expr \]
\[ expr \neq expr \]

<Examples>
- \( c \equiv 'A' \)
- \( k \neq -2 \)
- \( x + y \equiv 3 * z - 7 \)

<NOT Examples>
- \( a = b \) /* assignment */
- \( a = = b - 1 \) /* space not allowed*/
- \( (x + y) =! 44 \) /* (x + y) = (!44) */

- \( a \equiv b \)
  - is either \textbf{true} or \textbf{false}
  - is implemented as \( a - b \equiv 0 \)
Equality Operators and Expressions

<table>
<thead>
<tr>
<th>Declarations and Initializations</th>
</tr>
</thead>
<tbody>
<tr>
<td>int i = 1, j=2, k=3;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Expression</th>
<th>Equivalent expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>i == j</td>
<td>j == i</td>
<td>0</td>
</tr>
<tr>
<td>i != j</td>
<td>j != i</td>
<td>1</td>
</tr>
<tr>
<td>i + j + k == - 2 * -k</td>
<td>((i + j) + k) == ((- 2) * (- k))</td>
<td>1</td>
</tr>
</tbody>
</table>

!! A common programming error

```plaintext
if (a = 1)  
...
if (a == 1)  
...```
Logical Operators and Expressions

\(| expr | (\text{unary negation})|

\(<\text{Examples}>\)

!a
!(x + 7.7)
!(a < b || c < d)

\(<\text{NOT Examples}>\)

a!
!(x + 7.7)
a !!= b

- If \( expr \) has value zero, \( ! expr \) has the \text{int} value 1 (\true).
- If \( expr \) has \text{nonzero} value, \( ! expr \) has the \text{int} value 0 (\false).

\(!!5 \iff !(!5)\) has the value 1.
## Logical Operators and Expressions

<table>
<thead>
<tr>
<th>Expression</th>
<th>Equivalent expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>! c</td>
<td>! c</td>
<td>0</td>
</tr>
<tr>
<td>! (i - j)</td>
<td>! (i - j)</td>
<td>1</td>
</tr>
<tr>
<td>! i - j</td>
<td>(! i) - j</td>
<td>-7</td>
</tr>
<tr>
<td>! ! (x + y)</td>
<td>! (! (x + y))</td>
<td>1</td>
</tr>
<tr>
<td>! x * ! ! y</td>
<td>(! x) * (!(! ! y))</td>
<td>1</td>
</tr>
</tbody>
</table>

### Declarations and Initializations

```cpp
char    c = 'A';
int      i = 7, j = 7;
double  x= 0.0, y = 2.3;
```
Logical Operators and Expressions

\[ expr \text{ } || \text{ } expr \quad \text{(logical or)} \]
\[ expr \text{ } && \text{ } expr \quad \text{(logical and)} \]

**Examples**
- \( a \text{ } && \text{ } b \)
- \( a \text{ } || \text{ } b \)
- \( !(a < b) \text{ } && \text{ } c \)
- \( 3 \text{ } && \text{ } (-2 * a + 7) \)

**NOT Examples**
- \( a \text{ } && \quad /* \text{ missing operand } */ \)
- \( a \text{ } || \text{ } b \quad /* \text{ space not allowed } */ \)
- \( a \text{ } && \text{ } b \quad /* \text{ bitwise operator } */ \)
- \( &b \quad /* \text{ the address of } b */ \)

- && has higher precedence than ||.
- Both of && and || are of lower precedence than all unary, arithmetic, equality, and relational operators.
Logical Operators and Expressions

<table>
<thead>
<tr>
<th>Expression</th>
<th>Equivalent expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>i &amp;&amp; j &amp;&amp; k</td>
<td>(i &amp;&amp; j) &amp;&amp; k</td>
<td>1</td>
</tr>
<tr>
<td>x</td>
<td></td>
<td>i &amp;&amp; j - 3</td>
</tr>
<tr>
<td>i &lt; j &amp;&amp; x &lt; y</td>
<td>(i &lt; j) &amp;&amp; (x &lt; y)</td>
<td>0</td>
</tr>
<tr>
<td>i &lt; j</td>
<td></td>
<td>x &lt; y</td>
</tr>
<tr>
<td>A' &lt;= c &amp;&amp; c &lt;= 'Z'</td>
<td>('A' &lt;= c) &amp;&amp; (c &lt;= 'Z')</td>
<td>1</td>
</tr>
<tr>
<td>c - 1 == 'A'</td>
<td></td>
<td>c + 1 == 'Z'</td>
</tr>
</tbody>
</table>

Declarations and Initializations

```c
char    c = 'B';
int      i = 3, j = 3, k = 3;
double  x = 0.0, y = 2.3;
```

Short-circuit Evaluation

- In evaluating the expr.s that are the operands of `&&` and `||`, the evaluation process stops as soon as the outcome true or false is known.

  - `expr1 && expr2`, if `expr1` has value zero
  - `expr1 || expr2`, if `expr1` has nonzero value
Compound Statement

- Compound statement
  - a series of declarations and statements surrounded by braces
  - block
  - for grouping statements into an executable unit
  - is itself a statement, thus it can be placed wherever a statement is placed.
  ```
  {  
    a = 1;
    {     /* nested */
      b = 2;
      c = 3;
    }
  }
  ```
Expression and Empty Statement

- **Expression statement**
  - an expression followed by ;

- **Empty statement**
  - written as a single semicolon
  - useful where a statement is needed syntactically

```c
a = b;       /* assignment statement */
a + b + c;   /* legal, but no useful work gets done */
;            /* empty statement */
printf("%d\n", a);    /* a function call */
```
if and if-else Statements

```c
if (expr)
    statement

- If `expr` is nonzero, then `statement` is executed; otherwise, `statement` is skipped and control passes to the next statement.
```

```c
if (j < k) {
    min = j;
    printf("j is smaller than k\n");
}
```
if and if-else Statements

if (expr)
    statement1
else
    statement2

if (c >= 'a' && c <= 'z')
    ++lc_cnt;
else {
    ++other_cnt;
    printf("%c is not a lowercase letter\n", c);
}

if (i != j) {
    i += 1;
    j += 2;
};
else
    i -= j;       /* syntax error */
if and if-else Statements

```c
if (a ==1)
    if ( b == 2)       /* if statement is itself a statement */
        printf("***\n");
```

- **dangling else problem**

```c
if (a ==1)
    if ( b == 2)
        printf("***\n");
else
    printf("###\n");
```

An else attaches to the nearest if.
if and if-else Statements

```c
if (c == ' ')
    ++blank_cnt;
else if (c >= '0' && c <= '9')
    ++digit_cnt;
else if (c >= 'a' && c <= 'z' || c >= 'A' && c <= 'Z')
    ++letter_cnt;
else if (c == '\n')
    ++nl_cnt;
else
    ++other_cnt;
```

```c
if (c == ' ')
    ++blank_cnt;
else
    if (c >= '0' && c <= '9')
        ++digit_cnt;
    else
        if (c >= 'a' && c <= 'z' || c >= 'A' && c <= 'Z')
            ++letter_cnt;
        else
            ....
```
while Statement

**while** \((expr)\)

- statement
- next statement

- First \(expr\) is evaluated. If it is nonzero, then \(statement\) is executed and control is passed back to \(expr\). This repetition continues until \(expr\) is zero.
- Its body gets executed zero or more times.

```c
while ((c = getchar()) == ' ')  
    ;  /*empty statement*/
```

This code causes blank characters in the input stream to be skipped.
for Statement

\[ \text{for} \ (expr1; \ expr2; \ expr3) \]
\[ \text{statement} \]
\[ \text{next statement} \]

- First, \( expr1 \) (initialization) is evaluated.
- \( expr2 \) is evaluated. If it is nonzero, then \( \text{statement} \) is executed, \( expr3 \) is evaluated, and control is passed back to \( expr2 \).
  - \( expr2 \) is a logical expression controlling the iteration.
  - This process continues until \( expr2 \) is zero.
for Statement

for (i = 1; i <= n; ++i)
    factorial *= i;

sum = 0;
for (i = 1; i <= 10; ++i)
    sum += i;

for ( ; i <= 10; ++i)
    sum += i;

for ( ; ; )
    sum += i++;

Infinite Loop !!
Comma Operator

expr1, expr2

- `expr1` is evaluated, and then `expr2`.

```plaintext
a = 0, b = 1

for (sum = 0, i =1; i <= 10; ++i)
    sum += i;

//
// for (sum = 0, i =1; i <= 10; sum += i, ++i)
//
// for (sum = 0, i =1; i <= 10; ++i, sum += i)
```
do Statement

do
  statement
while (expr);
next statement

- First statement is executed and expr is evaluated. If the value of expr is nonzero, then control is passed back to statement. When expr is zero, control passes to next statement.
do Statement

\[
\text{do } \{ \\
\text{ printf("Input a positive integer: ");} \\
\text{ scanf("%d", &n);} \\
\text{ if (error = (n <= 0))} \\
\text{ printf("\nERROR: Do it again!\n\n");} \\
\} \text{ while (error);}
\]

For expressions of type float or double, an equality test can be beyond the accuracy of the machine.

\[
\text{double sum = 0.0, x;} \\
\text{ for (x = 0.0; x != 9.9; x += 0.1)} \\
\text{ sum += i;}
\]

Infinite Loop !!!!
\[ \Rightarrow \text{Use a relational expression!} \]
break and continue Statements

break;

- causes an exit from the intermost enclosing loop or switch statement

```c
while (1) {
    scanf("%lf", &x);
    if (x < 0.0)
        break; /* exit loop if x is negative */
    printf("%f\n", sqrt(x));
}
/* break jumps to here */
```
break and continue Statements

continue;

- causes the current iteration of a loop to stop and causes the next iteration of the loop to begin immediately

```c
for (i=0; i<TOTAL; ++i) {
    c = getchar();
    if (c >= '0' && c <= '9')
        continue;
    ....        /* process other characters */
    /*continue transfers control to here to begin next iteration*/
}
```
switch Statement

**switch statement**
- a multiway conditional statement generalizing the *if-else* statement

```c
switch (c) { /* c should be of integral type */
    case 'a':
        ++a_cnt;
        break;
    case 'b':
        case 'B':
            ++b_cnt;
            break;
    default:
        ++other_cnt;
}
```

1. Evaluate the **switch** expression.
2. Go to the **case** label having a constant value that matches the value of the expression in (1), or, if a match is not found, go to the **default** label, or, if there is no **default** label, terminate the **switch**.
3. Terminate the **switch** when a **break** statement is encountered, or terminate the **switch** by “falling off the end”.
Conditional Operator

expr1 ? expr2 : expr3

- **expr1** is evaluated.
  - If it is nonzero(true), then **expr2** is evaluated, and that is the value of the conditional expression as a whole.
  - If **expr1** is zero(false), then **expr3** is evaluated, and that is the value of the conditional expression as a whole.

```
if (y < z)
  x = y;
else
  x = z;
```

\[ \Leftrightarrow \]

```
x = (y < z) ? y : z;
```
Conditional Operator

\( expr1 \ ? \ expr2 \ : \ expr3 \)

- Its type is determined by both \( expr2 \) and \( expr3 \)
  - is determined by both \( expr2 \) and \( expr3 \)
  - Different types \( \Rightarrow \) Usual Conversion Rules
  - does not depend on which of \( expr2 \) or \( expr3 \) is evaluated.

<table>
<thead>
<tr>
<th>Declarations and Initializations</th>
</tr>
</thead>
<tbody>
<tr>
<td>char    a = 'a', b = 'b';</td>
</tr>
<tr>
<td>int     i = 1, j = 2;</td>
</tr>
<tr>
<td>double  x = 7.07;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Expression</th>
<th>Equivalent expression</th>
<th>Value</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>i==j ? a - 1 : b +1</td>
<td>(i==j) ? (a - 1) : (b +1)</td>
<td>99</td>
<td>int</td>
</tr>
<tr>
<td>j%3 == 0 ? i + 4 : x</td>
<td>((j%3) == 0) ? (i + 4) : x</td>
<td>7.07</td>
<td>double</td>
</tr>
<tr>
<td>j%3 ? i + 4 : x</td>
<td>(j%3) ? (i + 4) : x</td>
<td>5.0</td>
<td>double</td>
</tr>
</tbody>
</table>