Control Structures in C

- **Control structures** control the flow of execution in a program or function.
- There are three kinds of execution flow:
  - **Sequence**:
    - the execution of the program is sequential.
  - **Selection**:
    - A control structure which chooses alternative to execute.
  - **Repetition**:
    - A control structure which repeats a group of statements.
- We will focus on the **selection** control structure now.
Conditions

• A program may choose among alternative statements by testing the value of key variables.
  – e.g., if( your_grade > 60 )
    cout << “you are passed!”;

• **Condition** is an expression that is either false (represented by 0) or true (represented by 1).
  – e.g., “your_grade > 60” is a condition.

• Conditions may contain **relational** or **equality operators**, and have the following forms.
  – variable relational-operator variable (or constant)
  – variable equality-operator variable (or constant)
## Operators Used in Conditions

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;</code></td>
<td>Less than</td>
<td>Relational</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>Greater than</td>
<td>Relational</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>Less than or equal to</td>
<td>Relational</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>Greater than or equal to</td>
<td>Relational</td>
</tr>
<tr>
<td><code>==</code></td>
<td>Equal to</td>
<td>Equality</td>
</tr>
<tr>
<td><code>!=</code></td>
<td>Not equal to</td>
<td>Equality</td>
</tr>
</tbody>
</table>
Examples of Conditions

<table>
<thead>
<tr>
<th>Operator</th>
<th>Condition</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$&lt;=$</td>
<td>$x \leq 0$</td>
<td>$x$ less than or equal to 0</td>
</tr>
<tr>
<td>$&lt;$</td>
<td>$\text{Power} &lt; \text{MAX_POW}$</td>
<td>$\text{Power}$ less than $\text{MAX_POW}$</td>
</tr>
<tr>
<td>$==$</td>
<td>$\text{mom_or_dad} = 'M'$</td>
<td>$\text{mom_or_dad}$ equal to ‘M’</td>
</tr>
</tbody>
</table>
| $!=$     | $\text{num} 
eq \text{SETINEL}$ | $\text{num}$ not equal to $\text{SETINEL}$ |
Logical Operators

• There are three kinds of **logical operators**.
  – `&&`: and
  – `||`: or
  – `!`: not

• **Logical expression** is an expression which uses one or more logical operators, e.g.,
  – `(temperature > 90.0 && humidity > 0.90)`
  – `!(n <= 0 || n >= 100)`. 
# The Truth Table of Logical Operators

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>nonzero</td>
<td>nonzero</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>nonzero</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>nonzero</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Op 1</th>
<th>! Op 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>nonzero</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Operator Precedence

• An operator’s **precedence** determines its order of evaluation.

• **Unary operator** is an operator that has only one operand.
  - !, +(plus sign), −(minus sign), and &(address of)
  - They are evaluated second only after function calls.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>function calls</td>
<td>highest</td>
</tr>
<tr>
<td>! + − &amp;</td>
<td></td>
</tr>
<tr>
<td>* / %</td>
<td></td>
</tr>
<tr>
<td>+ −</td>
<td></td>
</tr>
<tr>
<td>&lt; &lt;= &gt; = &gt;</td>
<td></td>
</tr>
<tr>
<td>== !=</td>
<td></td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>=</td>
<td>lowest</td>
</tr>
</tbody>
</table>
Evaluation for \( \neg \text{flag} \lor (y + z \geq x - z) \)

\text{The result of this expression is true}
Comparing Characters

- We can also compare characters in C using the **relational** and **equality operators**.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘9’ &gt;= ‘0’</td>
<td>1 (true)</td>
</tr>
<tr>
<td>‘a’ &lt; ‘e’</td>
<td>1 (true)</td>
</tr>
<tr>
<td>‘Z’ == ‘z’</td>
<td>0 (false)</td>
</tr>
<tr>
<td>‘a’ &lt;= ‘A’</td>
<td>system dependent</td>
</tr>
</tbody>
</table>
DeMorgan’s Theorem

• **DeMorgan’s theorem** gives us a way of transforming a logical expression into its complement.
  - The complement of $\text{expr}_1 \land \text{expr}_2$ is $\text{comp}_1 \lor \text{comp}_2$, where $\text{comp}_1$ and $\text{comp}_2$ are the complement of $\text{expr}_1$ and $\text{expr}_2$, respectively.
  - The complement of $\text{expr}_1 \lor \text{expr}_2$ is $\text{comp}_1 \land \text{comp}_2$.

• e.g., $\text{age} > 25 \land (\text{status} == \text{'S'} \lor \text{status} == \text{'D'})$
  
is equal to
  $!(\text{age} \leq 25 \lor (\text{status} != \text{'S'}) \land \text{status} != \text{'D'})$
The **if** Statement

• The **if** statement is the primary selection control structure.
• Syntax: `if (condition) statement; else statement;`
• An example of two alternatives:
  ```cpp
  if ( rest_heart_rate > 56 )
    cout <<"Keep up your exercise program!\n";
  else
    cout <<"Your heart is in excellent health!\n";
  ```
• An example of one alternative:
  ```cpp
  if ( x != 0.0 )
    product = product * x;
  ```
Nested if Statements

- Nested if statement is an if statement with another if statement as its true task or false task.

- e.g.,
  ```c
  if (road_status == ‘S’)
      if (temp > 0) {
          cout <<“Wet roads ahead!\n”;
      }else{
          cout<<“Icy roads ahead!\n”;  
      }
  else
      printf(“Drive carefully!\n”);
  ```
An Example for the Flowchart of Nested if Statements

Main if statement

road_status is 'S'

Drive carefully message

Another if statement

temp > 0

true

false

Icy roads message

Wet roads message
Multiple-Alternative Decisions

- If there are many alternatives, it is better to use the syntax of *multiple-alternative decision*.

- Syntax:
  
  ```java
  if (condition_1)
     statement_1
  else if (condition_2)
     statement_2
  ...
  else if (condition_n)
     statement_n
  else
     statement_e
  ```
An Example of Multiple-Alternative Decisions

double comp_tax(double salary)
{
    double tax;

    if (salary < 0.0)
        tax = -1.0;

    else if (salary < 15000.00) /* first range */
        tax = 0.15 * salary;
    else if (salary < 30000.00) /* second range */
        tax = (salary - 15000.00) * 0.18 + 2250.00;
    else if (salary < 50000.00) /* third range */
        tax = (salary - 30000.00) * 0.22 + 5400.00;
    else if (salary < 80000.00) /* fourth range */
        tax = (salary - 50000.00) * 0.27 + 11000.00;
    else if (salary <= 150000.00) /* fifth range */
        tax = (salary - 80000.00) * 0.33 + 21600.00;
    else
        tax = -1.0;

    return (tax);
}
The `switch` Statement

- The `switch` statement is used to select one of several alternatives when the selection is based on the value of a single variable or an expression.

```java
switch (controlling expression) {
    case label_1:
        statement_1
        break;
    case label_2:
        statement_2
        break;
    ...
    case label_n:
        statement_n
        break;
    default:
        statement_d;
}
```

If the result of this controlling expression matches `label_1`, execute `statement_1` and then break this switch block.

If the result matches none of all labels, execute the default statement `statement_d`. 
An Example of a `switch` Statement with Type `char` Case Labels

```c
switch (class) {
    case 'B':
    case 'b':
        printf("Battleship\n");
        break;
    case 'C':
    case 'c':
        printf("Cruiser\n");
        break;
    case 'D':
    case 'd':
        printf("Destroyer\n");
        break;
    case 'F':
    case 'f':
        printf("Frigate\n");
        break;
    default:
        printf("Unknown ship class %c\n", class);
}
```
Homework #3 (1/2)

• Write a program that prompts the user to **input the boiling point** in degree Celsius.

• The program should **output the substance** corresponding to the boiling point listed in the table.

• The program should output the message **“substance unknown”** when it does not match any substance.

<table>
<thead>
<tr>
<th>Substance</th>
<th>Boiling point</th>
</tr>
</thead>
<tbody>
<tr>
<td>Water</td>
<td>100°C</td>
</tr>
<tr>
<td>Mercury</td>
<td>357°C</td>
</tr>
<tr>
<td>Copper</td>
<td>1187°C</td>
</tr>
<tr>
<td>Silver</td>
<td>2193°C</td>
</tr>
<tr>
<td>Gold</td>
<td>2660°C</td>
</tr>
</tbody>
</table>
Homework #3 (2/2)

• Examples of the scenario of your program.

Please input: 357
The substance is Mercury.

Please input: 3333
Substance unknown.

• You can determine the substance **within a range of boiling points** to get bonus (e.g., +5 degrees).

Please input: 359
The substance is Mercury.

• You can apply any technique