Textbook

- The material covered today is in chapters 3 and 4 of the textbook
- Later today (?) we will cover material in chapter 5
- Please go through this material before the lab
How Did the First Lab Go?

• We hope you enjoyed it!

• We know there were some glitches. Sorry, we hope they did not affect you too much.
The quizzes in these slides are from http://www.cprogramming.com/.

The rest of the material is original, so feedback is welcome.
Code indentation!

- Indent your code
- Indentation is good because it helps you (and helps others too) seeing mistakes
How to indent?

• All lines should start on the same column except if the line is { or }
  – { and } should go on a line “alone”

• If a line contains { ..the next line should be indented a “tab” deeper
  – What is a tab? [photo of keyboard]

• If a line contains } it should go a tab less deep than the previous one
Indentation example

```c
#include <stdio.h>

int main(void){
    int number = 3;
    if (number > 2) {
        printf("%d > 2!", number);
    }
    return 0;
}
```
Indentation example

```c
#include <stdio.h>

int main(void)
{
    int number = 3;
    if (number > 2)
    {
        printf("%d > 2!", number);
    }
    return 0;
}
```
Enumerations

- The keyword enum allows you to associate labels to integer variables:
  ```
  enum{MON, TUE, WED, THU, FRI, SAT, SUN};
  int day = FRI; /* day is 4 */
  ```

- The first label gets the value zero, the second one, etc..

- This makes the code more readable: good!!!

- We can specify the value of the first label and the others will follow
  ```
  enum{MON=100, TUE, WED, THU, FRI, SAT, SUN};
  /* FRI is 104 */
  ```
Enumerations (cont.)

- Integer variables can be declared as enumerations:
  ```c
  enum working_day {MON, TUE, WED, THU, FRI};
  enum working_day wd = WED; /* wd is 2 */
  ```

- However variables declared in this way are treated the very same way as int!!!

- So the following does not raise any errors or warnings:
  ```c
  wd = 1270;
  ```

- You should know about enum because it is widely used, but keep your expectations low when you use it!
  More info on http://tinyurl.com/83qe6vc
typedef

- Instead of:
  ```c
  enum working_day {MON, TUE, WED, THU, FRI};
  enum working_day wd = WED;
  ```

- We can use typedef to write:
  ```c
  typedef enum {MON, TUE, WED, THU, FRI} working_day;
  working_day wd = WED;
  ```

- typedef allows us to define *something like* a new type
  - Not really a new type! (no type checking!)

- It will be very useful when we will work with structures
#DEFINE Constants

- Enumerations are essentially a way to define constants (e.g. FRI is 5)

- There is another way:
  #DEFINE SAT 5

- This is not limited to integers:
  #DEFINE SAT "Saturday"

- A simple example application:
  #DEFINE SAT "Sabato"
  /* Saturday in Italian */
Arithmetic Operations

• No comment on addition (+), subtraction (-), multiplication (*) and division (/)

• The modulus operator % returns the remainder of the integer division between two integers
e.g. 10 % 3 is 1

• Increment (++) and decrement (--) update the value of the variable by +1 or -1
integerVariable++; /* is equivalent to: */
inumberVariable = integerVariable + 1;
Assignment

• We already used the basic assignment operator '=

  – Later today we will see '===' that is the equality operator

• Other assignment operators are: +=, -=, *=, /=, %= 

• These update the value of a variable, e.g.
  integerVariable += 5; /* is equivalent to: */
  integerVariable = integerVariable + 5;
Comparisons

• You can use the following operators to compare values:
  
  a == b; /* a equal to b */
  a != b; /* a different from b */
  a > b; /* a greater than b */
  a < b; /* a less than b */
  a >= b; /* a greater than or equal to b */
  a <= b; /* a less than or equal to b */
  
• They return 1 if they are verified 0 otherwise
  
• Try it!
  
  printf("5==5: %d, 5==4: %d\n", (5==5), (5==4));
Comparison Results

• Comparisons return 1 if they are verified 0 otherwise

• Zero corresponds to \textit{false}

• Does 1 correspond to \textit{true}?
  – Yes, \textbf{but} it is not the only value corresponding to true

• In C any value different from zero is interpreted as \textit{true}
Boolean Logic Operators

- The following operators can be used to combine comparisons:
  
x && y; /* logical AND */
  
x || y; /* logical OR */
  
!x;     /* logical NOT */

- For example:
  
(x > 3) && (x < 6); /* true when x is 4 or 5 */
  
(x < 3) || (x > 6); /* always true except when x is 3, 4, 5 or 6 */
Binary Representation

- *There are only 10 types of people in the world: those who understand binary, and those who don't.*
- In base 2 only 2 digits are available: 0 and 1
  - In base 10 we have 10 digits: 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9
- Computers store data in binary format
- A binary digit is commonly referred to as a bit
- 8 bits make a byte
Binary Representation of Integers

- For integers representation is quite simple

<table>
<thead>
<tr>
<th>Binary</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>100</td>
<td>4</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- In N bits we can represent up to $2^N$ values
- In 8 bits we can represent up to decimal 255 (that makes 256 values including zero)
Binary for Floating Point Numbers

- For floating point numbers the representation is less trivial
- There are many ways to represent them, a common one is defined by the IEEE-754 standard
  - One bit is used for sign
  - N bits are used for the mantissa 'm'
  - M bits are used for the exponent 'x'
  - A number is then represented as sign times 1.m times 2^x

- For more information see for example http://en.wikipedia.org/wiki/IEEE_floating_point
The `sizeof` operator

- The `sizeof` operator returns the number of memory bytes used by a variable, today..
  - computers typically have giga bytes of RAM
  - phones typically have mega bytes of RAM
  - micro-controllers can have just few kilo bytes!
- The size of different types depend on the implementation, except `char`, which is always of size 1
  - On computers *typically* `int` and `float` are size 4, `double` 8
Binary Operators

- Sometimes (esp. when working with micro-controllers) it can be useful to operate directly on the bits that are used to represent integer numbers

- C provides the following binary operators
  
x & y;  /* binary AND */
x | y;   /* binary OR */
~x;     /* binary NOT */
x ^ y;   /* XOR */
x << y;  /* left shift */
x >> y;  /* right shift */
Binary Operators (cont.)

- It will be a lot easier to learn about binary operators when you will work with microcontrollers, so that you can see their practical applications.

- Same for "flagging bits" on the book.
Operator Precedence

• You know that multiplication takes precedence over addition, e.g. $3 \times 4 + 2 = (3 \times 4) + 2$

• What about C operators beyond arithmetic ones?

• Operator precedence is defined by the language (you will find a table in the book)

• In general it is a good idea to explicitly use parenthesis when you combine multiple operators, to make the code easier to read
## Operator Precedence (cont.)

<table>
<thead>
<tr>
<th>Description</th>
<th>Represented By</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Parenthesis</td>
<td>() []</td>
</tr>
<tr>
<td>1 Structure Access</td>
<td>. -&gt;</td>
</tr>
<tr>
<td>2 Unary</td>
<td>! - ++ -- - * &amp;</td>
</tr>
<tr>
<td>3 Multiply, Divide, Modulus</td>
<td>* / %</td>
</tr>
<tr>
<td>4 Add, Subtract</td>
<td>+ -</td>
</tr>
<tr>
<td>5 Shift Right, Left</td>
<td>&gt;&gt; &lt;&lt;</td>
</tr>
<tr>
<td>6 Greater, Less Than, etc</td>
<td>&gt; &lt; =</td>
</tr>
<tr>
<td>7 Equal, Not Equal</td>
<td>== !=</td>
</tr>
<tr>
<td>8 Bitwise AND</td>
<td>&amp;</td>
</tr>
<tr>
<td>9 Bitwise Exclusive OR</td>
<td>^</td>
</tr>
<tr>
<td>10 Bitwise OR</td>
<td></td>
</tr>
<tr>
<td>11 Logical AND</td>
<td>&amp;&amp;</td>
</tr>
<tr>
<td>12 Logical OR</td>
<td></td>
</tr>
<tr>
<td>13 Conditional Expression</td>
<td>? :</td>
</tr>
<tr>
<td>14 Assignment</td>
<td>= += -= etc</td>
</tr>
<tr>
<td>15 Comma</td>
<td>,</td>
</tr>
</tbody>
</table>

Image from [http://www2.its.strath.ac.uk/courses/c/section3_16.html](http://www2.its.strath.ac.uk/courses/c/section3_16.html)
Quiz to check your understanding

Which of the following is the correct operator to compare two variables?

- A. :=
- B. =
- C. equal
- D. ==
Quiz to check your understanding

- Which of the following is true?
  - A. 1
  - B. 66
  - C. .1
  - D. -1
  - E. All of the above
Quiz to check your understanding

• Which of the following is the boolean operator for logical-and?
  – A. &
  – B. &&
  – C. |
  – D. |&
Quiz to check your understanding

• Evaluate !(1 && !(0 || 1)).
  – A. True
  – B. False
  – C. Cannot be evaluated
The if Statement

• The if statement allows you to execute statements only when certain conditions are verified

• This the anatomy of an if statement:
  ```python
  if (<condition>)
  {
    <statements executed if condition is true>
  }
  ```
The if Statement (cont.)

• Here are some simple examples:

```c
if (temperature <= 0.0)
{
    printf("it's freezing!\n");
}

/* assume q is a char variable with value inputted from the user */
if (input == 'q')
{
    printf("are you sure you want to quit?\n");
}
```
If-else

• We can add an else statement to any if statement

• General definition:

```java
if (<condition>)
{
    <statements executed if condition is true>
}
else
{
    <statements executed if condition is false>
}
```
If-else example

• Adding to the earlier examples:
  
  ```c
  if (temperature <= 0.0)
  {
    printf("it's freezing!\n");
  }
  else
  {
    printf("it's NOT freezing\n");
  }
  ```
Writing code that is clear for **humans** (yourself and others) is **critical** for good programming!

- Indent your code
- The `enum` and `DEFINE` statements allow you to associate values with text labels (enum is only for integers)
- C has operators to calculate and compare values
- The `if` and `else` statements can be used to implement simple control logic