Data Type
Syntax Rules Recap

- **keywords**
  - `break` `double` `if` `sizeof` `void`
  - `case` `else` `int` `static` `.....`

- **Identifiers**
  - `not#me` `123th`
  - `scanf` `printf`
  - `_id` `so_am_i` `gedd007`

- **Constant**
  - `122.72` `‘a’` `‘+’`

- **String Constants**
  - “a string of text” “a”

- **Operators**

```
( []
+ - * / %
&& || !   /* logic */
^ ~ & |   /* bitwise */
sizeof
?:      /* (n > 0) ? f : n */
* & ->
>> <<   /* shift */
< > <= >= == !=        /* relational operators */
++ -- /* increment and decrement */
= += -= *= /= %= >>= <<= &=
^= |n
```
## Operators

### Operator precedence (order from top to down)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[]</td>
</tr>
<tr>
<td>!</td>
<td>~</td>
</tr>
<tr>
<td>*</td>
<td>/</td>
</tr>
<tr>
<td>+</td>
<td>- (binary)</td>
</tr>
<tr>
<td>*</td>
<td>/</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>&gt;&gt;</td>
</tr>
<tr>
<td>&lt;</td>
<td>&lt;=</td>
</tr>
<tr>
<td>==</td>
<td>!=</td>
</tr>
<tr>
<td>&amp;</td>
<td>left to right</td>
</tr>
<tr>
<td>^</td>
<td>left to right</td>
</tr>
<tr>
<td></td>
<td></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>=</td>
<td>+=</td>
</tr>
<tr>
<td>,</td>
<td>left to right</td>
</tr>
</tbody>
</table>
#include <stdio.h>

int main(void)
{
    int a, b, c; /* declaration */
    float x, y = 3.3, z = -7.7; /* declaration with initializations */

    printf("Input two integers: "); /* function call */
    scanf("%d%d", &b, &c); /* function call */
    a = b + c; /* assignment */
    x = y + z; /* assignment */
    ....
}
Fundamental Data Types

- all variables must be declared before they are used
- other types (array, pointer, structure, union) are derived from the fundamental data types
Data Types and Sizes

• sizes are machine dependant
  – short and int are at least 16 bits
  – long is at least 32 bits
  – short \leq int \leq long

• float
  – typically 4 bytes (32bits)
  – double is 8 bytes
  – floating arithmetic is NOT always exact
    • refer <float.h> <limits.h>
Characters

- assume a single byte for a character even though it is represented as int
  - 256 distinct characters are possible

<table>
<thead>
<tr>
<th>name of character</th>
<th>written in C with \</th>
<th>corresponding integer value</th>
</tr>
</thead>
<tbody>
<tr>
<td>alert (bell)</td>
<td>\a</td>
<td>7</td>
</tr>
<tr>
<td>backslash</td>
<td>\</td>
<td>92</td>
</tr>
<tr>
<td>backspace</td>
<td>\b</td>
<td>8</td>
</tr>
<tr>
<td>carriage return</td>
<td>\r</td>
<td>13</td>
</tr>
<tr>
<td>double quote</td>
<td>&quot;</td>
<td>34</td>
</tr>
<tr>
<td>formfeed</td>
<td>\f</td>
<td>12</td>
</tr>
<tr>
<td>horizontal tab</td>
<td>\t</td>
<td>9</td>
</tr>
<tr>
<td>newline</td>
<td>\n</td>
<td>10</td>
</tr>
<tr>
<td>null character</td>
<td>\0</td>
<td>0</td>
</tr>
<tr>
<td>single quote</td>
<td>'</td>
<td>39</td>
</tr>
<tr>
<td>vertical tab</td>
<td>\v</td>
<td>11</td>
</tr>
<tr>
<td>question mark</td>
<td>?</td>
<td>63</td>
</tr>
</tbody>
</table>
char c = 'a';
    /* ASCII code for 'a' is 01100001 */
printf("%c", c);
    /* a is printed */
printf("%d",c);
    /* 97 is printed */
printf("%c%c%c",c,c+1,c+2);
    /* abc is printed */
char c;
int i;

for (i = 'a'; i <= 'z'; ++i)
    printf("%c", i);       /* abc ... z is printed */
for (c = 65; c <= 90; ++c)
    printf("%c", c);       /* ABC ... Z is printed */
for (c = '0'; c <= '9'; ++c)
    printf("%d ", c);       /* 48 49 ... 57 is printed */
Character Types

• ANSI C provides three types of char
  - char is either one of the followings
  - signed char       -128~127
  - unsigned char     0~255

• int
  - 16 bits for small/old computers
  - 32 bit for your computers
  - what if overflow occurs
    • depends on the CPU
<table>
<thead>
<tr>
<th>Suffix</th>
<th>Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>u or U</td>
<td>unsigned</td>
<td>37U, 127u</td>
</tr>
<tr>
<td>l or L</td>
<td>long</td>
<td>37L</td>
</tr>
<tr>
<td>ul or UL</td>
<td>unsigned long</td>
<td>37UL</td>
</tr>
</tbody>
</table>

- Suffixes can be appended to an integer constant to specify its type.
- The type of an unsuffixed integer constant is either `int`, `long`, or `unsigned long` depending on how large is integer number.
- ANSI C provides the three floating types to represent real numbers: `float`, `double` (working floating type), and `long double`.

<table>
<thead>
<tr>
<th>Suffix</th>
<th>Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>f or F</td>
<td>float</td>
<td>3.7F</td>
</tr>
<tr>
<td>l or L</td>
<td>long double</td>
<td>3.7L</td>
</tr>
</tbody>
</table>
• Examples of floating constants
  3.14159
  314.159e-2F /* of type float */
  0e0
  /* floating point zero 0.0 of type double */
  1.0 /* double 1.0 */

• Incorrect syntax for floating constants
  3.14,159 /* comma not allowed */
  314159 /* no decimal point or exponent */
  .e4 /* only decimal point not allowed */
  -3.14159 /* constant expression not a constant */
Floating Numbers

- IEEE 754 floating point standard:
  - Single precision: \((\text{sign})(\text{significand})*2^{\text{exp}}\)
    - 8 bit exponent (0~127) = (-63~64)
    - 23 bit significand
    - 1 bit sign
  - Double precision: \((11, 52, 1)\)
typedef char uppercase;
typedef int Inches, Feet;

uppercase FirstChar;
Inches length, width;
sizeof( ) operator

• returns the number of bytes
  – because some sizes are machine dependent
• guaranteed

\[
\text{sizeof(char)} = 1 \\
\text{sizeof(char)} \leq \text{sizeof(short)} \leq \text{sizeof(int)} \leq \text{sizeof(long)} \\
\text{sizeof(signed)} = \text{sizeof(unsigned)} = \text{sizeof(int)} \\
\text{sizeof(float)} \leq \text{sizeof(double)} \leq \text{sizeof(long double)}
\]
getchar( ) and putchar( )

- defined in <stdio.h>
  - getchar() reads in a character
  - putchar() writes out a character
    - to/from the standard device

```c
#include <stdio.h>

int main(void)
{
    int c;

    while ((c = getchar()) != EOF) {
        putchar(c);
        putchar(c);
    }
    return 0;
}
```
#include <stdio.h>

int main(void)
{
    int c;

    while ((c = getchar()) != EOF)
    {
        if (c >= 'a' && c <= 'z')
            putchar(c + 'A' - 'a');
        else
            putchar(c);
    }
    return 0;
}

capitalize.c
#include <math.h>
#include <stdio.h>

int main(void)
{
    double x;

    printf("The square root of x and x raised to the x power will be computed. \\
        ---");
    while (1) {
        /* do it forever */
        printf("Input x: ");
        scanf("%lf", &x);
        if (x >= 0.0)
            printf("x = ", x,
                   "sqrt(x) = ", sqrt(x),
                   "pow(x, x) = ", pow(x, x));
        else
            printf("Sorry, your number must be nonnegative.");
    }
    return 0;
}
Arithmetic Conversions

• Some data types are converted automatically in an expression and on an assignment
  int op int
  short op short => int
  int op float => float

• Some rules
  – small one is converted to a large one
    float op long
    long op double
    int op float
Automatic Conversions

- on an assignment \( d = i; \)  \( i \) is converted to the type of \( d \)

<table>
<thead>
<tr>
<th>Declarations</th>
<th>Declarations</th>
<th>Declarations</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>char c;</code></td>
<td><code>short s;</code></td>
<td><code>int i;</code></td>
</tr>
<tr>
<td><code>long l;</code></td>
<td><code>unsigned u;</code></td>
<td><code>unsigned long ul;</code></td>
</tr>
<tr>
<td><code>float f;</code></td>
<td><code>double d;</code></td>
<td><code>long double ld;</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Expression</th>
<th>Type</th>
<th>Expression</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c - s / i )</td>
<td><code>int</code></td>
<td>( u * 7 - i )</td>
<td><code>unsigned</code></td>
</tr>
<tr>
<td>( u * 2.0 - i )</td>
<td><code>double</code></td>
<td>( f * 7 - i )</td>
<td><code>float</code></td>
</tr>
<tr>
<td>( c + 3 )</td>
<td><code>int</code></td>
<td>( 7 * s * ul )</td>
<td><code>unsigned long</code></td>
</tr>
<tr>
<td>( c + 5.0 )</td>
<td><code>double</code></td>
<td>( ld + c )</td>
<td><code>long double</code></td>
</tr>
<tr>
<td>( d + s )</td>
<td><code>double</code></td>
<td>( u - ul )</td>
<td><code>unsigned long</code></td>
</tr>
<tr>
<td>( 2 * i / l )</td>
<td><code>long</code></td>
<td>( u - l )</td>
<td><code>system-dependent</code></td>
</tr>
</tbody>
</table>
Cast

• you can force explicit conversions
  – (double) i
  – (long) ('A' + 1.0)
  – f = (float) ( (int) d + 1) * (double)(x = 77)