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

<table>
<thead>
<tr>
<th>Operator precedence (order from top to down)</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>0   [] . -&gt;</td>
<td>left to right</td>
</tr>
<tr>
<td>! ~ ++ -- + - * &amp; (type) sizeof</td>
<td>right to left</td>
</tr>
<tr>
<td>* / % (binary)</td>
<td>left to right</td>
</tr>
<tr>
<td>+ - (binary)</td>
<td>left to right</td>
</tr>
<tr>
<td>&lt;&lt; &gt;&gt;</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; !=</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>= += -= *= /= %= &amp;= ^=</td>
<td>= &lt;&lt;= &gt;&gt;=</td>
</tr>
<tr>
<td>,</td>
<td>left to right</td>
</tr>
</tbody>
</table>
Declarations, Expressions, and Assignment

#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

<table>
<thead>
<tr>
<th>char</th>
<th>signed char</th>
<th>unsigned char</th>
</tr>
</thead>
<tbody>
<tr>
<td>short</td>
<td>int</td>
<td>long</td>
</tr>
<tr>
<td>unsigned short</td>
<td>unsigned</td>
<td>unsigned long</td>
</tr>
<tr>
<td>float</td>
<td>double</td>
<td>long double</td>
</tr>
</tbody>
</table>
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

• A suffix can be appended to a floating constant to specify its type (without suffix, by default it will be 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. /* 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: \((\text{sign})(\text{significand}) \times 2^{\text{exp}}\)
    - 8 bit exponent (0~127) = (-63~64)
    - 23 bit significand
    - 1 bit sign
  - Double precision: \((11, 52, 1)\)

Float: precision 6 sig. figures; range 10^{-38} to 10^{+38}
- \(0.d_1d_2d_3d_4d_5d_6 \times 10^n\)

Double: 15, 10^{-308} to 10^{308}
- \(0.d_1d_2d_3d_4d_5d_6d_7d_8d_9d_{10}d_{11}d_{12}d_{13}d_{14}d_{15} \times 10^n\)
<table>
<thead>
<tr>
<th>Data Type Definition</th>
<th>typedef</th>
</tr>
</thead>
<tbody>
<tr>
<td>typedef char</td>
<td>uppercase;</td>
</tr>
<tr>
<td>typedef int</td>
<td>Inches, Feet;</td>
</tr>
<tr>
<td>uppercase</td>
<td>FirstChar;</td>
</tr>
<tr>
<td>Inches</td>
<td>length, width;</td>
</tr>
</tbody>
</table>
```c
sizeof(char) = 1
sizeof(char) ≤ sizeof(short) ≤ sizeof(int) ≤ sizeof(long)
sizeof(signed) = sizeof(unsigned) = sizeof(int)
sizeof(float) ≤ sizeof(double) ≤ 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;
}
Mathematical Functions

#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 = %15s%22.15e
            sqrt(x) = %15s%22.15e
            pow(x, x) = %15s%22.15e
            
            ");
        else
            printf("Sorry, your number must be nonnegative.
            ");
    }
    return 0;
}

• many mathematical functions are available from the math library
  – include <math.h>
  – link with the library “gcc -lm code.c”
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>
</tr>
</thead>
<tbody>
<tr>
<td>char c;</td>
</tr>
<tr>
<td>short s;</td>
</tr>
<tr>
<td>long l;</td>
</tr>
<tr>
<td>unsigned u;</td>
</tr>
<tr>
<td>float f;</td>
</tr>
<tr>
<td>double d;</td>
</tr>
<tr>
<td>int i;</td>
</tr>
<tr>
<td>unsigned long ul;</td>
</tr>
<tr>
<td>long double ld;</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>int</td>
<td>u * 7 - i</td>
<td>unsigned</td>
</tr>
<tr>
<td>u * 2.0 - i</td>
<td>double</td>
<td>f * 7 - i</td>
<td>float</td>
</tr>
<tr>
<td>c + 3</td>
<td>int</td>
<td>7 * s * ul</td>
<td>unsigned long</td>
</tr>
<tr>
<td>c + 5.0</td>
<td>double</td>
<td>ld + c</td>
<td>long double</td>
</tr>
<tr>
<td>d + s</td>
<td>double</td>
<td>u - ul</td>
<td>unsigned long</td>
</tr>
<tr>
<td>2 * i / l</td>
<td>long</td>
<td>u - l</td>
<td>system-dependent</td>
</tr>
</tbody>
</table>
Cast

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