Data Type

Fall 2014
Jinkyu Jeong
(jinkyu@skku.edu)
Syntax Rules Recap.

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

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

- **Constants**
  - 122.72  ‘a’  ‘+’

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

  - **Unary plus & minus (+, -)**
    - a = -26
    - +a = -26
    - + +a = -26
    - -a = 26
    - - -a = -26
Operators Recap.

<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>*   /  %</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
- long long is added in C99
Data Types: char

- **char**
  - All lower-case/upper-case alphabets, digits, special symbols
  - A single letter enclosed within '
  - Symbols combined with '\ (backslash)

\[ 'A', 'B', 'C', 'D', 'E', ..., 'Z', 'a', 'b', 'c', 'd', 'e', ..., 'z', '0', '1', '2', '3', '4', ..., '9', '!', '@', '#', '$', '%', ..., '), '][', '{', '}', '<', '>', '/', '\', '
\]

\[ '\', '\\', '\\', '\n', '\t', ',', '\a' \]
Data Types: char

- **char** is actually an integer type (??)
  - We use a table to map characters to 1-byte integer
  - Typically, ASCII code is used in C.

<table>
<thead>
<tr>
<th>32</th>
<th>48</th>
<th>0</th>
<th>64</th>
<th>@</th>
<th>80</th>
<th>P</th>
<th>96</th>
<th>`</th>
<th>112</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>33</td>
<td>49</td>
<td>1</td>
<td>65</td>
<td>A</td>
<td>81</td>
<td>Q</td>
<td>97</td>
<td>a</td>
<td>113</td>
<td>q</td>
</tr>
<tr>
<td>34</td>
<td>50</td>
<td>2</td>
<td>66</td>
<td>B</td>
<td>82</td>
<td>R</td>
<td>98</td>
<td>b</td>
<td>114</td>
<td>r</td>
</tr>
<tr>
<td>35</td>
<td>51</td>
<td>3</td>
<td>67</td>
<td>C</td>
<td>83</td>
<td>S</td>
<td>99</td>
<td>c</td>
<td>115</td>
<td>s</td>
</tr>
<tr>
<td>36</td>
<td>52</td>
<td>4</td>
<td>68</td>
<td>D</td>
<td>84</td>
<td>T</td>
<td>100</td>
<td>d</td>
<td>116</td>
<td>t</td>
</tr>
<tr>
<td>37</td>
<td>53</td>
<td>5</td>
<td>69</td>
<td>E</td>
<td>85</td>
<td>U</td>
<td>101</td>
<td>e</td>
<td>117</td>
<td>u</td>
</tr>
<tr>
<td>38</td>
<td>54</td>
<td>6</td>
<td>70</td>
<td>F</td>
<td>86</td>
<td>V</td>
<td>102</td>
<td>f</td>
<td>118</td>
<td>v</td>
</tr>
<tr>
<td>39</td>
<td>55</td>
<td>7</td>
<td>71</td>
<td>G</td>
<td>87</td>
<td>W</td>
<td>103</td>
<td>g</td>
<td>119</td>
<td>w</td>
</tr>
<tr>
<td>40</td>
<td>56</td>
<td>8</td>
<td>72</td>
<td>H</td>
<td>88</td>
<td>X</td>
<td>104</td>
<td>h</td>
<td>120</td>
<td>x</td>
</tr>
<tr>
<td>41</td>
<td>57</td>
<td>9</td>
<td>73</td>
<td>I</td>
<td>89</td>
<td>Y</td>
<td>105</td>
<td>i</td>
<td>121</td>
<td>y</td>
</tr>
<tr>
<td>42</td>
<td>58</td>
<td>10</td>
<td>74</td>
<td>J</td>
<td>90</td>
<td>Z</td>
<td>106</td>
<td>j</td>
<td>122</td>
<td>z</td>
</tr>
<tr>
<td>43</td>
<td>59</td>
<td>11</td>
<td>75</td>
<td>K</td>
<td>91</td>
<td>[</td>
<td>107</td>
<td>k</td>
<td>123</td>
<td>{</td>
</tr>
<tr>
<td>44</td>
<td>60</td>
<td>12</td>
<td>76</td>
<td>L</td>
<td>92</td>
<td>\</td>
<td>108</td>
<td>l</td>
<td>124</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>61</td>
<td>13</td>
<td>77</td>
<td>M</td>
<td>93</td>
<td>]</td>
<td>109</td>
<td>m</td>
<td>125</td>
<td>}</td>
</tr>
<tr>
<td>46</td>
<td>62</td>
<td>14</td>
<td>78</td>
<td>N</td>
<td>94</td>
<td>^</td>
<td>110</td>
<td>n</td>
<td>126</td>
<td>~</td>
</tr>
<tr>
<td>47</td>
<td>63</td>
<td>15</td>
<td>79</td>
<td>O</td>
<td>95</td>
<td>_</td>
<td>111</td>
<td>o</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Data Types: char

- **char**: the memory size and range of values
  - Consistent across all the computer platforms
  - 1 byte is allocated for a single character: 256 values possible

```c
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 */
```
Data Types: char

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

- **int**
  - Signed integral number without a decimal point
  - **The memory size and range of values**

<table>
<thead>
<tr>
<th>Machine</th>
<th>Size</th>
<th>Range of values</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-byte</td>
<td>2 byte</td>
<td>-32768(-2(^{15})) ~ 32767(2(^{15})-1)</td>
</tr>
<tr>
<td>4-byte</td>
<td>4 byte</td>
<td>-2147483648(-2(^{31})) ~ 2147483647(2(^{31})-1)</td>
</tr>
<tr>
<td>8-byte</td>
<td>4 byte</td>
<td>-2147483648(-2(^{31})) ~ 2147483647(2(^{31})-1)</td>
</tr>
</tbody>
</table>

- Querying min/max by predefined macros
  - INT_MIN, INT_MAX are defined in `<limits.h>`
Data Types: Numeric Types

- Arithmetic of \texttt{int} type
  - Usual addition (+), subtraction (-), multiplication (*), division (/)

- Overflow
  - A value resulting from an operation goes outside the valid range
  - Such a value is interpreted as different
  - Example: 4-byte machine:
    
    -2147483648(-2^{31}) \sim 2147483647(2^{31}-1)

\[
\begin{align*}
2147483646 + 1 &= \,? \\
2147483647 + 1 &= \,? \\
2147483647 + 2 &= \,? \\
-2147483648 -1 &= \,? \\
-2147483648 -2 &= \,? \\
\end{align*}
\]
Data Types: Numeric Types

- **char**, **short**, **long** and **long long** (in C99)
  - Numeric data types, but the space for them varies

- **Sizes are machine dependent**
  - **short** and **int** are at least 16 bits
  - **long** is at least 32 bits
  - **long long** is at least 64 bits
  - **short** ≤ **int** ≤ **long** ≤ **long long**

* Linux x86_64

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>1 byte</td>
<td>-128 (2^7)</td>
<td>127(2^7-1)</td>
</tr>
<tr>
<td>short</td>
<td>2 bytes</td>
<td>-32768(-2^15)</td>
<td>32767(2^15-1)</td>
</tr>
<tr>
<td>int</td>
<td>4 bytes</td>
<td>-2147483648(-2^31)</td>
<td>2147483647(2^31-1)</td>
</tr>
<tr>
<td>long</td>
<td>8 bytes</td>
<td>-9223372036854775808(-2^63)</td>
<td>9223372036854775807(2^63-1)</td>
</tr>
<tr>
<td>long long</td>
<td>8 bytes</td>
<td>-9223372036854775808(-2^63)</td>
<td>9223372036854775807(2^63-1)</td>
</tr>
</tbody>
</table>
Data Types: Numeric Types

- **Unsigned vs. signed**
  - When data type is unsigned, it represents only positive integers
    - unsigned char, unsigned short, unsigned int, unsigned long
  - Signed is by default
    - int = signed int
    - short = signed short int
    - long = signed long int

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char</td>
<td>1 byte</td>
<td>0</td>
<td>255(2^8-1)</td>
</tr>
<tr>
<td>unsigned short</td>
<td>2 bytes</td>
<td>0</td>
<td>65535(2^{16}-1)</td>
</tr>
<tr>
<td>unsigned int</td>
<td>4 bytes</td>
<td>0</td>
<td>4294967295(2^{32}-1)</td>
</tr>
<tr>
<td>unsigned long</td>
<td>8 bytes</td>
<td>0</td>
<td>18446744073709551615(2^{64}-1)</td>
</tr>
<tr>
<td>unsigned long long</td>
<td>8 bytes</td>
<td>0</td>
<td>18446744073709551615(2^{64}-1)</td>
</tr>
</tbody>
</table>

* Linux x86_64
Declaring Numeric Constants

<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 the integer number.
Data Types: 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: \((11, 52, 1)\)

  \[
  \text{Float: precision 6 sig. figures; range } 10^{-38} \text{ to } 10^{+38} \\
  \quad \cdot \quad 0.d_1d_2d_3d_4d_5d_6 \times 10^n
  \]

  \[
  \text{Double: 15, } 10^{-308} \text{ to } 10^{308} \\
  \quad \cdot \quad 0.d_1d_2d_3d_4d_5d_6d_7d_8d_9d_{10}d_{11}d_{12}d_{13}d_{14}d_{15} \times 10^n
  \]
Data Types: Floating Numbers

- **float**
  - Real numbers can be represented, but significant digits are **approximated** up to 6 or 7 digits

- **The memory size and range of values**

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Range of positive values</th>
</tr>
</thead>
<tbody>
<tr>
<td>float</td>
<td>4 byte</td>
<td>$1.18 \times 10^{-38} \sim 3.40 \times 10^{38}$</td>
</tr>
<tr>
<td>double</td>
<td>8 byte</td>
<td>$2.23 \times 10^{-308} \sim 1.80 \times 10^{308}$</td>
</tr>
<tr>
<td>long double</td>
<td>16 byte</td>
<td>$3.36 \times 10^{-4932} \sim 1.19 \times 10^{4932}$</td>
</tr>
</tbody>
</table>

- **Querying min/max by predefined macros**
  - FLT_MIN, FLT_MAX, DBL_MIN, DBL_MAX are defined in `<float.h>`
Data Types: Floating Numbers

- Arithmetic of float type
  - Usual addition (+), subtraction (-), multiplication (*), division (/)
  - Significant digits are limited by approximation

\[
\begin{align*}
0.1234567 + 0.00000008 &= ? \\
12345670 + 8 &= ? \\
123.4567 + 100000 &= ? \\
0.123457 &
\end{align*}
\]

\[
\begin{align*}
0.123457 &
\end{align*}
\]

\[
\begin{align*}
12345678 &
\end{align*}
\]

\[
\begin{align*}
100123.453125 &
\end{align*}
\]
Data Types: Floating Numbers

- Mixed arithmetic between int and float types
  - The type of results is of type float
  - The results of comparison operation between int and float values are also done as you expect.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 + 1 == ?</td>
<td>3.0</td>
</tr>
<tr>
<td>2 * 1 == ?</td>
<td>2.0</td>
</tr>
<tr>
<td>3 / 2 == ?</td>
<td>1.5</td>
</tr>
<tr>
<td>3 % 2 == ?</td>
<td>1.0</td>
</tr>
<tr>
<td>2.0 + 1.0 == ?</td>
<td>3.0</td>
</tr>
<tr>
<td>2.0 * 1.0 == ?</td>
<td>2.0</td>
</tr>
<tr>
<td>2.0 / 1.0 == ?</td>
<td>2.0</td>
</tr>
<tr>
<td>3.0 % 2.0 == ?</td>
<td>1.0</td>
</tr>
<tr>
<td>2 &lt; 1 ?</td>
<td>True</td>
</tr>
<tr>
<td>2.0 &gt; 1 ?</td>
<td>False</td>
</tr>
<tr>
<td>2.0 &lt;= 1.0 ?</td>
<td>True</td>
</tr>
</tbody>
</table>
Declaring Floating Constants

- 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>1 or L</td>
<td>long double</td>
<td>3.7L</td>
</tr>
</tbody>
</table>
Declaring Floating Constants

- **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 */
Data Type Definition

- **typedef**
  - Explicit definition of types by programmer

```
typedef char   uppercase;
typedef int   INCHES, FEET;

#if __WORDSIZE == 64
    typedef unsigned long int   uint64_t;
#else
    typedef unsigned long long int uint64_t;
#endif
```
sizeof( ) operator

- Returns the number of bytes of types or variables
  - Some sizes are machine dependent
- Guaranteed

```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

```
#include <stdio.h>

int main(void)
{
    int    c;

    while ((c = getchar()) != EOF) {
        putchar(c);
        putchar(c);
    }
    return 0;
}
```
Example: `getchar() / putchar()`

```c
#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
Many mathematical functions are available from the math library

- include <math.h>
- link with the library

\$ gcc -lm code.c

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

int main(void)
{
    double x;

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

- Integral promotions
  - char, short → int (or unsigned int)

- Usual arithmetic conversions
  - Small one is converted to a large one
  - If floating number exists, converts to floating types
    ```
    float op long → float
    long op double → long
    long op unsigned int → unsigned long
    ```

- On an assignment `d = i;`
  - `i` is converted to the type of `d`
### Automatic Conversions

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

- Explicit conversions

  - (double) i
  - (long) (‘A’ + 1.0)
  - f = (float) ( (int) d + 1) * (double)(x = 77)