Lecture 3: Overview

- Basic Types
- Type conversion
  - explicit
  - implicit
- Types in Expressions
- Shift Operators
- Conditional Operator
- Formatting Output
Basic Types in C

- **Integer types**
  - `char` Character, e.g. ‘a’, ‘b’, ‘1’, ‘*’
    - typical range \([-128, 127]\]
  - `short int` Short integer, e.g. -7, 0, 42
    - typical range \([-32768, 32767]\]
  - `int` Integer, e.g. -7, 0, 42
    - typical range \([-2147483648, 2147483647]\]
  - `long int` Long integer, e.g. -991, 91, 1231
    - typical range \([-2147483648, 2147483647]\]
  - `long long int` Very long integer, e.g. 1234511
    - typical range
      \([-9223372036854775808, 9223372036854775807]\]
  - Integer types can be
    - `signed` negative and positive values (and 0)
    - `unsigned` positive values only (and 0)
Basic Types in C

- Floating point types
  - `float` Floating point with single precision
    - Example `3.5f`, `-0.234f`, `10E8f`
  - `double` Floating point with double precision
    - Example `3.5`, `-0.23456789012`, `10E88`
  - `long double` Floating point with high precision
    - Example `12345678.123456E123l`

- Floating point values are in many cases *approximations* only!
  - Storage size of floating point values is fixed
  - Many values can only be represented as approximations
  - Example: `1.0/3.0 = .333...`
Type Conversion

• Explicit Type Conversion
  • types can be explicitly converted to other types, by use of the type cast operator:
    \((\text{type}) \text{ expression}\)
  • the target type is named explicitly in parentheses before the source expression
  • Examples:
    • \text{Float} = (\text{float}) \text{ LongInt}
      - converts the \text{long int} type into a \text{float} type
    • \text{Integer} = (\text{int}) \text{ Double}
      - converts the \text{double} type into an \text{int} type
      - any fractional part is truncated!
    • \text{Char} = (\text{char}) \text{ LongLongInt}
      - converts the \text{long long int} type into a \text{char} type
      - any out-of-range values are silently cut off!
Type Conversion

- Implicit Type Conversion
  - Type promotion
    - integral promotion
      - unsigned or signed char is promoted to unsigned or signed int before any operation
      - unsigned or signed short is promoted to unsigned or signed int before any operation
    - floating-point promotion
      - float is promoted to double before any operation
  - binary arithmetic operators are defined only for same types
    - Examples:
      - the smaller type is converted to the larger type
        » ShortInt * LongInt results in a long int type
        » LongDouble * Float results in a long double type
- Type coercion
  - most types are automatically converted to expected types
  - Example: Double = Float, or Char = LongInt
Types in Expressions

- Expressions are composed of constants, variables and operators, each of which has an associated type.

Example:
```c
short int s;
int i;
long int l;
float f;
double d;
l = 2 * s + i * f - 0.5 * d;
```
Shift Operators

• **Left-shift operator:**  \( x << n \)
  • shifts \( x \) in binary representation \( n \) times to the left
  • multiplies \( x \) \( n \) times by 2
  • Examples
    • \( 2x = x << 1 \)
    • \( 4x = x << 2 \)
    • \( x \times 2^n = x << n \)
    • \( 2^n = 1 << n \)

• **Right-shift operator:**  \( x >> n \)
  • shifts \( x \) in binary representation \( n \) times to the right
  • divides \( x \) \( n \) times by 2
  • Examples
    • \( x/2 = x >> 1 \)
    • \( x/4 = x >> 2 \)
    • \( x/2^n = x >> n \)
Conditional Operator

- Conditional evaluation of values in expressions
- Question-mark operator:
  \( \text{test} \ ? \ \text{true-value} : \ \text{false-value} \)
  - evaluates the \text{test}
  - if \text{test} is true, then the result is \text{true-value}
  - otherwise, the result is \text{false-value}

Examples:
- \((4 < 5) \ ? \ (42) : (4+8)\) evaluates to \(42\)
- \((2==1+2) \ ? \ (x) : (y)\) evaluates to \(y\)
- \((x < 0) \ ? \ (-x) : (x)\) evaluates to \(\text{abs}(x)\)
Translation

```c
int x=(test) ? true-expression : false-expression;

int x;
if (test) {
    x=true-expression;
} else {
    x=false-expression;
}
```
### Conversion Specifiers for Basic Types

<table>
<thead>
<tr>
<th>Type</th>
<th>printf()</th>
<th>scanf()</th>
</tr>
</thead>
<tbody>
<tr>
<td>long double</td>
<td>%Lf</td>
<td>%Lf</td>
</tr>
<tr>
<td>double</td>
<td>%f</td>
<td>%lf</td>
</tr>
<tr>
<td>float</td>
<td>%f</td>
<td>%f</td>
</tr>
<tr>
<td>unsigned long long</td>
<td>%llu</td>
<td>%llu</td>
</tr>
<tr>
<td>long long</td>
<td>%lldd</td>
<td>%lldd</td>
</tr>
<tr>
<td>unsigned long</td>
<td>%lu</td>
<td>%lu</td>
</tr>
<tr>
<td>long</td>
<td>%ld</td>
<td>%ld</td>
</tr>
<tr>
<td>unsigned int</td>
<td>%u</td>
<td>%u</td>
</tr>
<tr>
<td>int</td>
<td>%d</td>
<td>%d</td>
</tr>
<tr>
<td>short</td>
<td>%hd</td>
<td>%hd</td>
</tr>
<tr>
<td>char</td>
<td>%c</td>
<td>%c</td>
</tr>
</tbody>
</table>
Formatted Output

- Formatted output using `printf()`
  - standard format specifiers for integral values
    - `unsigned long long %llu`
    - `long long %lld`
    - `unsigned long %lu`
    - `long %ld`
    - `unsigned int %u`
    - `int %d`
    - `short %hd`
  - standard format specifiers for floating point values
    - `long double %Lf`
    - `double %f`
    - `float %f`
Formatted Output

- Detailed formatting sequence for integral values
  - `% flags width length conversion`
  - `flags`
    - (none) standard formatting (right-justified)
    - - left-justified output
    - + leading plus-sign for positive values
    - 0 leading zeros
  - `field width`
    - (none) minimum number of characters needed
    - integer width of field to be filled with output
  - `length` modifier
    - (none) int type
    - h short int type
    - l long int type
    - ll long long int type
  - `conversion` specifier
    - d signed decimal value
    - u unsigned decimal value
    - o (unsigned) octal value
    - x (unsigned) hexadecimal value using characters 0–9, a–f
    - X (unsigned) hexadecimal value using characters 0–9, A–F
Detailed formatting sequence for floating-point values

• % flags width precision length conversion

• flags
  • (none) standard formatting (right-justified)
  • – left-justified output
  • + leading plus-sign for positive values
  • 0 leading zeros

• field width
  • (none) minimum number of characters needed
  • integer width of field to be filled with output

• precision
  • (none) default precision (e.g. 6)
  • . int number of digits after decimal point (for f, e, or E),
    maximum number of significant digits (for g, or G)

• length modifier
  • (none) float or double type
  • L long double type

• conversion specifier
  • f standard floating-point notation (fixed-point)
  • e or E exponential notation using (e or E)
  • g or G standard or exponential notation (using e or E)
Math Library Functions

- C standard math library
  - standard library supplied with every C compiler
  - predefined mathematical functions
    - e.g. $\cos(x)$, $\sqrt{x}$, etc.
- Math library header file
  - contains math function declarations
  - `#include <math.h>`
- Math library linker file
  - contains math function definitions (pre-compiled)
    - library file `libm.a`
  - compiler needs to `link` against the math library
  - use option `-llibraryname`
  - Example: `gcc MathProgram.c -o MathProgram -lm`
Math Library Functions

- Functions declared in `math.h` (part 1/2)

  - `double sqrt(double x);` \[ \sqrt{x} \]
  - `double pow(double x, double y);` \[ x^y \]
  - `double exp(double x);` \[ e^x \]
  - `double log(double x);` \[ \log(x) \]
  - `double log10(double x);` \[ \log_{10}(x) \]
  - `double ceil(double x);` \[ \lceil x \rceil \]
  - `double floor(double x);` \[ \lfloor x \rfloor \]
  - `double fabs(double x);` \[ |x| \]
  - `double fmod(double x, double y);` \[ x \mod y \]
Math Library Functions

• Functions declared in `math.h` (part 2/2)
  
  - `double cos(double x);` \( \cos(x) \)
  - `double sin(double x);` \( \sin(x) \)
  - `double tan(double x);` \( \tan(x) \)
  - `double acos(double x);` \( \arccos(x) \)
  - `double asin(double x);` \( \arcsin(x) \)
  - `double atan(double x);` \( \arctan(x) \)
  - `double cosh(double x);` \( \cosh(x) \)
  - `double sinh(double x);` \( \sinh(x) \)
  - `double tanh(double x);` \( \tanh(x) \)
Math Library Functions

• **Program example: Function.c (part 1/3)**

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

/* function definition */

double f(double x)
{
    return cos(x);
} /* end of f */
```

...
Math Library Functions

- Program example: Function.c (part 2/3)

```c
... /* main function */

int main(void)
{
    /* variable definitions */
    double hi, lo, step;
    double x, y;

    /* input section */
    printf("Please enter the lower bound: ");
    scanf("%lf", &lo);
    printf("Please enter the upper bound: ");
    scanf("%lf", &hi);
    printf("Please enter the step size: ");
    scanf("%lf", &step);

    ... 
```
Math Library Functions

• Program example: `Function.c` (part 3/3)

```c
/* computation and output section */
for(x = lo; x <= hi; x += step)
{
    y = f(x);
    printf("f(%10g) = %10g\n", x, y);
} /* rof */

/* exit */
return 0;
} /* end of main */

/* EOF */
```
Math Library Functions

• Example session: Function.c

```c
% vi Function.c
% gcc Function.c -o Function -Wall -ansi -lm
% Function
Please enter the lower bound: -0.5
Please enter the upper bound: 1.0
Please enter the step size: .1
f(-0.5) = 0.877583
f(-0.4) = 0.921061
f(-0.3) = 0.955336
f(-0.2) = 0.980067
f(-0.1) = 0.995004
f(-2.77556e-17) = 1
f(0.1) = 0.995004
f(0.2) = 0.980067
f(0.3) = 0.955336
f(0.4) = 0.921061
f(0.5) = 0.877583
f(0.6) = 0.825336
f(0.7) = 0.764842
f(0.8) = 0.696707
f(0.9) = 0.62161
f(1) = 0.540302
%```
Standard Library Functions

• Standard C library
  • standard library supplied with every C compiler
  • predefined standard functions
    • e.g. `printf()`, `scanf()`, etc.

• C library header files
  • input/output function declarations `#include <stdio.h>`
  • standard function declarations `#include <stdlib.h>`
  • time function declarations `#include <time.h>`
  • etc.

• C library linker file
  • contains standard function definitions (pre-compiled)
    • library file `libc.a`
  • compiler *automatically links* against the standard library
    (no need to supply extra options)
Standard Library Functions

• Functions declared in stdlib.h (partial list)
  • int abs(int x);
  • long int labs(long int x);
    • return the absolute value of a (long) integer x
  • int rand(void);
    • return a random value in the range 0 – RAND_MAX
    • RAND_MAX is a constant integer (e.g. 32767)
  • void srand(unsigned int seed);
    • initialize the random number generator with value seed
  • void exit(int result);
    • exit the program with return value result
  • void abort(void);
    • abort the program (with an error result)
Standard Library Functions

- Random number generation
  - Standard library provides *pseudo* random number generator
    - `int rand(void);`
  - Pseudo random numbers are a sequence of values seemingly random in the range 0 – `RAND_MAX`
    - Computer is a *deterministic* machine
    - Sequence will always be the same
  - Start of sequence is determined by *seed* value
    - `void srand(unsigned int seed);`
Seeding the Random Number Generator

- Sometimes a repeatable sequence is good
  - Enables us to repeat numerical simulations
  - Useful for debugging
- Sometimes it is bad
  - Slot machines & other games
  - Encryption algorithms
- Trick: Initialize random sequence with current time
  - header file `time.h` declares function `unsigned int time()`
  - `time(0)` returns number of seconds since Jan 1, 1970
  - at beginning of program, use: `srand(time(0));`