Declarations and Assignments
Declarations

• declaration ::= type identifier { , identifier }* ;
• All variables must be declared before they can be used
  • Type information for storage allocation by the compiler
  • Help the compiler to choose correct operations
• { and } surround a block, which consists of declarations and statements
  • The declarations must occur before the statements
  • The body of a function is a block
• Examples
  • int a, b, c;
  • float x, y = 3.14;
Simple Assignment Statements

In the form of,
- variable = expr ;
- = : assignment operator

Expressions are meaningful combinations of constants, variables, operators, and function calls
- a + b
- sqrt( 3.14 ) – sin( 3.0 / x)

1. The value of expr is computed
2. The value is assigned to variable

Examples
- x = x + 1;
- y = 3.14;
# Fundamental Data Types

<table>
<thead>
<tr>
<th>Integral types</th>
<th>char</th>
<th>signed char</th>
<th>unsigned char</th>
</tr>
</thead>
<tbody>
<tr>
<td>short (signed short int)</td>
<td>int (signed int)</td>
<td>long (signed long int)</td>
<td></td>
</tr>
<tr>
<td>unsigned short (unsigned short int)</td>
<td>unsigned (unsigned int)</td>
<td>unsigned long (unsigned long int)</td>
<td></td>
</tr>
<tr>
<td>Floating types</td>
<td>float</td>
<td>double</td>
<td>long double</td>
</tr>
<tr>
<td>Arithmetic types</td>
<td>Integral types + Floating types</td>
<td>Integral types + Floating types</td>
<td>Integral types + Floating types</td>
</tr>
</tbody>
</table>
### Data Type `char`

- A variable of type `char` can be used to hold small integer values (1 byte, 8 bit, 256 distinct values)
- Most machines use either ASCII or EBCDIC character code

<table>
<thead>
<tr>
<th>Character constants</th>
<th>‘a’</th>
<th>‘b’</th>
<th>‘c’</th>
<th>…</th>
<th>‘z’</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII values</td>
<td>97</td>
<td>98</td>
<td>99</td>
<td>…</td>
<td>112</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Character constants</th>
<th>‘A’</th>
<th>‘B’</th>
<th>‘C’</th>
<th>…</th>
<th>‘Z’</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII values</td>
<td>65</td>
<td>66</td>
<td>67</td>
<td>…</td>
<td>90</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Character constants</th>
<th>‘0’</th>
<th>‘1’</th>
<th>‘2’</th>
<th>…</th>
<th>‘9’</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII values</td>
<td>48</td>
<td>49</td>
<td>50</td>
<td>…</td>
<td>57</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Character constants</th>
<th>‘&amp;’</th>
<th>‘*’</th>
<th>‘+’</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII values</td>
<td>38</td>
<td>42</td>
<td>43</td>
</tr>
</tbody>
</table>
Data Type **char** (contd.)

- Either `char` ≡ `signed char` or `char` ≡ `unsigned char`
  - Depending on the compiler
- `signed char`: -128 ~ 127
- `unsigned char`: 0 ~ 255

| Alert           | 1a | 7 | Backslash | \ | 92 | Backspace | \b | 8 |
|-----------------|----|---|-----------|---|----|-----------|    |---|
| Carriage return | \r | 13| Double quote | \” | 34 | formfeed | \f | 12|
| Horizontal tab  | \t | 9 | newline    | \n | 10 | Null character | \0 | 0 |
| Single quote    | \’ | 39| Vertical tab | \v | 11 | Question mark | \? | 63|
Data Type `int`

- The integer values that are representable on a machine `int` is stored in 4 bytes.
- If `a` is a variable of type `int`, the range of values that `a` can take on is,
  - $-2^{31} \leq a \leq 2^{31} - 1$
- Integer overflow can occur, but the program continues to run.
short, long, and unsigned

- **short**: 2 byte integer
  - $-2^{15} \leq a \leq 2^{15} - 1$

- **long**: 4 byte integer
  - The compiler may provide more storage than int
  - $-2^{31} \leq a \leq 2^{31} - 1$

- **unsigned**: 4 byte unsigned integer
  - $0 \leq a \leq 2^{32} - 1$
### Suffixes for Data Type `int`

- The type of an unsuffixed integer constant
  - `int`, `long`, `unsigned long`
  - The system chooses the first of these types that can contain the value

<table>
<thead>
<tr>
<th>Suffix</th>
<th>Type</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>U or u</td>
<td>unsigned</td>
<td>25U, 0x32u</td>
</tr>
<tr>
<td>L or l</td>
<td>long</td>
<td>37L, 0x27L</td>
</tr>
<tr>
<td>UL or ul</td>
<td>unsigned long</td>
<td>56ul, 234UL</td>
</tr>
</tbody>
</table>
Floating Types

- Variables of this type hold real values
- float: 4 bytes, 6 decimal places
- double: 8 bytes, 15 decimal places
- long double: 8 bytes
  - The compiler may provide more storage than double
## Suffixes for Floating Types

- Any unsuffixed floating constant is of type double
- Examples
  - 3.14159
  - 314.159e-2F
  - 0e0
  - 1.
  - 314.159e-22

### Suffixes for Floating Types

<table>
<thead>
<tr>
<th>Suffix</th>
<th>Type</th>
<th>Examples</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>2.7L</td>
</tr>
</tbody>
</table>
**typedef**

- Allows the programmer explicitly associate a type with an identifier

```c
typedef char sval8b;
typedef int sval4B, sval32b;
typedef unsigned long size_t;

sval8b v;
sval4B x, y;
```
sizeof Operator

- sizeof ( object )
- Unary operator
- To find (returns) the number of bytes needed to store object
Type Conversion

- An arithmetic expression (such as $x + y$) has both a value and a type.
- Integral promotion
  - (signed or unsigned) char or (signed or unsigned) short can be used in any place where an int or unsigned int may be used.
  - If all the values of the original type can be represented by an `int`, then the value is converted to an `int`; otherwise it is converted to an `unsigned int`. 
Type Conversion (contd.)

- Type conversions can occur across an assignment or when the operands of a binary operator are evaluated.
- Examples
  - `int i; float f;`
    - `i + f`, the operand `i` gets promoted to a float and the expression `i + f` as a whole has type `float`.
  - `double d; int i;`
    - `d = i`, `i` is converted to double and assigned to `d`.
- Other names: automatic conversion, implicit conversion, coercion, promotion, widening.
Type Conversion (contd.)

char c;  short s;  int i;  long l;
unsigned u; unsigned long ul;
float f;  double d;  long double ld;
c-s/i (int), u*3.0-i (double)
c+4 (int), c+3.0 (double)
d+s (double), 7*i/l (long)
u*4-i (unsigned), f*8-i (float)
8*s*ul (unsigned long), ld + c (long double)
u-ul (unsigned long), u-l (system dependent)
There are explicit type conversion called casts

Examples

- (double) i
- (long) (‘A’ + 1.0)
- (double) (x = 22)
- (float) i + 3 ≡ ((float) i) + 3
- d = (double) i / 5
printf

- The programmer need to declare the function that is used in his/her program
- Function prototypes
  - A way of declaring functions
    - int foo(int a);
    - float bar(void);
  - Printf’s prototype is included in stdio.h
- Printf is passed a list arguments, control_string and other_arguments
  - control_string
    - a string that may contain conversion specification or formats
printf (contd.)

- `printf("abcde");`
- `printf("%s", "abcde");`
- `printf("%c%c%c%c%c", 'a', 'b', 'c', 'd', 'e');`
- `printf("%c%4c%4c", 'a', 'b', 'c');`

<table>
<thead>
<tr>
<th>c</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>Decimal integer</td>
</tr>
<tr>
<td>e</td>
<td>Floating-point number in scientific notation</td>
</tr>
<tr>
<td>f</td>
<td>Floating-point number</td>
</tr>
<tr>
<td>g</td>
<td>In the e-format or f-format, whichever is shorter</td>
</tr>
<tr>
<td>s</td>
<td>string</td>
</tr>
</tbody>
</table>
**scanf**

- Analogous to printf, but used for input
- scanf is passed a list arguments, control_string and other_arguments
  - other_arguments
    - addresses
  - `scanf("%d", &x);`
    - &x: the address of x
    - The format %d is matched with &x
      - Characters in the input stream as a decimal integer and store it at the address of x
When reading numbers, it skips white space (blanks, newlines, and tabs), but when reading in a character, white space is not skipped.

<table>
<thead>
<tr>
<th>c</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>Decimal integer</td>
</tr>
<tr>
<td>f</td>
<td>Floating-point number (float)</td>
</tr>
<tr>
<td>LF or lf</td>
<td>Floating-point number (double)</td>
</tr>
<tr>
<td>s</td>
<td>string</td>
</tr>
</tbody>
</table>
Call by Value

- Arguments to functions are always passed by value.
- The variables passed as arguments to functions are not changed.
- The expression passed as an argument to a function is first evaluated, and its value is passed to the function.

```
int foo(int x, int y);
foo(a + b, c);
```

```
4 3 5
+ 7 5
```

```
int foo(int x, int y)
```
```c
#include <stdio.h>

int main(void)
{
    char c1, c2;
    int i;
    float x;
    double y:

    printf("\%s\%s", "Input two characters, ",
           "an int, a float, and a double: ");
    scanf("\%c\%c\d\%f\lf", &c1, &c2, &i, &x, &y);
    printf("The data you typed in:\n");
    printf("\%c \%c \%d \%f \%e\n", c1, c2, i, x, y);

    return 0;
}
```
fedoraduo:/home/jlee> gcc day02.c
fedoraduo:/home/jlee> a.out
Input two characters, an int, a float, and a double: Ef 234 3.14 3.14159LF
The data you typed in:
E f 234 3.140000 3.141590e+00
fedoraduo:/home/jlee>
Exercises

Use the following code to print out a list of powers of 2 in decimal, hexadecimal, and octal

```c
int i, val = 1;

for(i = 1; i < 35; i = i + 1) {
    printf("%15d%15u%15x%15o\n", val, val, val, val);
    val = val * 2;
}
```
Try the following program:

```c
#include <math.h>
#include <stdio.h>
int main(void)
{
    double two_pi = 2.0 * M_PI; /* in math.h */
    double h = 0.1;
    double x;
    for(x = 0.0; x < two_pi; x = x + h)
        printf("%5.1f: %.15e\n", x,
                sin(x) * sin(x) + cos(x) * cos(x));
    return 0;
}
```
Exercises (contd.)

- Write a program that prints a table of trigonometric values for \( \sin() \), \( \cos() \), and \( \tan() \)
  - The angles in your table go from 0 to \( 2\pi \) in 20 steps