Lecture 4. Functions and Modules

- ***Functions*** **are the basic building blocks of C programs**
- **Programmer-defined functions**

  application-specific: good for only the application in which they appear

  general-purpose: good for a wide range of applications
- **Libraries hold collections of 'standard' general-purpose functions**

I/O	*Math*	*Strings*	*Other*	*…*
`printf`	`sqrt`	`strcmp`	`rand`	
`fprintf`	`sin`	`strcpy`	`malloc`	
`scanf`	`cos`	`strlen`	`atoi`	
…	…	…	…	

  Use standard functions whenever possible; reuse, don't reinvent
- **A function** ***declaration*** **gives the types of the arguments and the return type**
- **A function** ***definition*** **is also a declaration plus a function** ***body***
- **A function** ***body*** **is a compound statement that implements the function**
- **A function** ***call*** **invokes the named function, which executes, then returns**

  the ***caller*****, or calling function, is the function in which the function call appears**
  the ***callee*** **, or called function, is the function that is invoked**

Computing e^x

- **Goal: write a program to approximate e^x, where $e = 2.718282\ldots$**

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots$$

where $n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 3 \cdot 2 \cdot 1$

- **Compute e^x to a given precision: iterate until e^x changes by less than the precision**

For x = 1.0, precision = 0.0001

i	$x^i/i!$	e^x
1	1.000000	1.000000
2	1.000000	1.000000
3	0.500000	2.000000
4	0.166667	2.500000
5	0.041667	2.666667
6	0.008333	2.708333
7	0.001389	2.716667
8	0.000198	2.718056
9	0.000025	2.718254

```
% lcc ex.c
% a.out
Enter x and the precision:
1 .00001
e^1.000000 = 2.718282; should be 2.718282
% a.out
Enter x and the precision:
2 .0001
e^2.000000 = 7.389047; should be 7.389056
```

Computing e^x, cont'd

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

float epowx(float, float);

int main(void) {
    float precision, x, ex;

    printf("Enter x and the precision:\n");
    scanf("%f%f", &x, &precision);
    ex = epowx(x, precision);
    printf("e^%f = %f; should be %f\n", x, ex, exp(x));
    return 0;
}

float epowx(float x, float epsilon) {
    int i;
    float ex = 1.0, prevex = 0.0, num = 1.0, denom = 1.0;

    i = 1;
    while (fabs(ex - prevex) > epsilon) {
        prevex = ex;
        num *= x;
        denom *= i++;
        ex += num/denom;
    }
    return ex;
}
```

Dissecting ex.c

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

Includes the standard header `math.h`, which contains *declarations* for the standard library functions `exp` and `fabs`

```
float epowx(float, float);
```

This *function declaration*, or *prototype*, says that `epowx` is a function that takes 2 `float` arguments and returns a `float` value

Functions must be declared (or defined) before they are used

```
scanf("%f%f", &x, &precision);
```

Calls **`scanf` to read two floating-point values (`%f`) and store them in `x` and `precision`**

```
ex = epowx(x, precision);
```

Calls **`epowx` with the values of `x` and `precision` just read; `epowx` returns a `float`, which is stored in `ex`**

`main` is the *caller*, `epowx` is the *callee*

```
printf("e^%f = %f; should be %f\n", x, ex, exp(x));
```

Calls **`exp(x)` to compute the 'real' value of e^x, then *calls* `printf` with 4 arguments: a format string, the value of `x`, the value of `ex`, and the value returned by `exp`; *conversion specifier* `%f` prints the corresponding argument as a float**

Dissecting ex.c, cont'd

```
float epowx(float x, float epsilon) {
    ...
}
```

The *function definition* for `epowx`; `x` and `epsilon` are the function *parameters*, both `float`s, and `epowx` returns a value of type `float`; `{ ... }` contains the *body*

```
int i;
float ex = 1.0, prevex = 0.0, num = 1.0, denom = 1.0;
```

These *declarations* specify the *local variables* in `epowx` and initialize all but `i`

```
i = 1;
while (fabs(ex - prevex) > epsilon) {
    prevex = ex;
    num *= x;
    denom *= i++;
    ex += num/denom;
}
```

This loop adds terms in the series until the difference between successive values of `ex` is less than or equal to `epsilon`; `fabs` is a standard library function

```
return ex;
```

This *return statement* returns the value of `ex` to the caller

Scope (a.k.a. Visibility)

- The ***scope*** of an identifier is that part of the program in which the identifier can be used
- ***Declarations*** of parameters and local variables ***introduce new identifiers***

  The scope of a function parameter is the body of the function

  The scope of a local variable extends from its declaration to the end of the compound statement in which the declaration appears

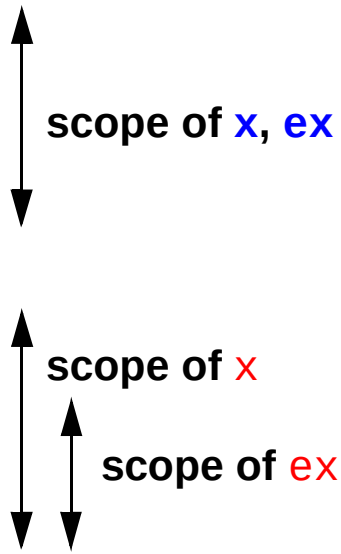
- Identifiers in different scopes are ***unrelated***, even if they have the same name

```
int main(void) {
    float precision, x, ex;

    …
    return 0;
}

float epowx(float x, float epsilon) {
    int i;
    float ex = 1.0, prevex = 0.0, …;

    …
    return ex;
}
```



Scope, cont'd

- Cannot declare the same identifier ***twice*** in the same scope

```
float epowx(float x, float epsilon) {
    int x;
    …          <-- error!
}
```

- Local declarations 'hide' parameter declarations and outer-level local declarations

```
f(int x, int a) {
    int y, b;
    y = x + a*b;
    if (…) {
        int a, b;
        …
        y = x + a*b;
    }
}
```

a hides parameter a; b hides outer-level local b

- Some consider it poor style to hide outer-level identifiers

Arguments and Locals

- ***Local*** **variables are** ***temporary*** **variables**

  Created **upon entry to the function in which they are declared**

  Destroyed **upon return**

- ***Arguments*** **are transmitted** ***by value***

  the values of the actual arguments are ***copied*** **to the formal parameters**

- **Arguments are** ***initialized local variables*** **and can be used just like any locals**

```
/* Illustrate call-by-value. */
#include <stdio.h>

void f(int a, int x) {
    printf("a = %d, x = %d\n",
        a, x);
    a = 3;
    {
        int x = 4;
        printf("a = %d, x = %d\n",
            a, x);
    }
    printf("a = %d, x = %d\n", a, x);
    x = 5;
}
```

```
int main(void) {
    int a = 1, b = 2;

    f(a, b);
    printf("a = %d, b = %d\n",
        a, b);
    return 0;
}
```

```
% lcc args.c
% a.out
a = 1, x = 2
a = 3, x = 4
a = 3, x = 2
a = 1, b = 2
%
```

- **Some consider it poor style to modify arguments**

Global Variables

- A ***global variable*** is defined or declared outside of functions
- Globals are ***'permanent'*** variables

  Created when the program begins; ***destroyed*** when the program terminates
- The ***scope*** of global is from the point of declaration to the end of the file

  in file `foo.c`:

```
int main(void) {
    ...
}

int max = 0;

void compute(...) {
    ...
}
```

`max` ***cannot*** be used here

`max` ***can*** be used here

- Parameters and locals 'hide' globals with the same names

```
void compute(...) {
    int max;
    ...
}
```

local `max` hides global `max`

- Global variables ***are*** initialized to 0 by default
  (some consider it poor style to rely on this feature)

Modules

- A ***module*** is a set of related global variables and functions in one or more files
- `extern` ***declarations*** make globals and functions accessible from ***other files***

  in file `baz.c`:

```
extern int max;
void dump(...) {
    ...
}
```

The `max` defined in `foo.c` ***can*** be used here

- General-purpose ***modules*** are often packaged in ***two*** files

The ***interface***	a header file (a `.h` file) of ***declarations*** for the variables and functions
The ***implementation***	a `.c` file of ***definitions*** for those variables and functions

- Implementations can be ***compiled separately***, and the compiled code can be stored in ***libraries***

Modules, cont'd

random.h:

```
extern int random(void); /*
    returns a random number in the range 0..2147483646. */

extern int seed; /* Initial seed for random(); default 0. */
```

random.c:

```
/*
Random number generator; see Press et al.,
Numerical Recipes in C, 2/e, 278-9.
*/
#include "random.h"

int seed = 0;

int random(void) {
    int k;

    seed ^= 123459876;
    k = seed/127773;
    seed = 16807*(seed - k*127773) - 2836*k;
    if (seed < 0)
        seed += 2147483647;
    k = seed;
    seed ^= 123459876;
    return k;
}
```