Functions
Function Definition

- In the form of
  - type func_name ( param_list ) { decls stmts }

- return_statement ::= return ; | return expr ;

```c
int fact( int n )
{
    int i, p;
    for (p = 1, i = 2; i <= n; i++)
        p *= i;
    return p;
}
```
Function Prototypes

- In the form of,
  - `type func_name (param_type_list);`
- Functions must be declared before they are used
  - Forward declaration
- Examples
  
  ```
  void foo( int, float );
  ≡ void foo( int a, float b);
  ```
Call by Value Revisited

```c
#include <stdio.h>
int sum( int m );
int main( void )
{
    int n = 5;
    printf( "sum = %d\n", sum( n ) );
    printf( "n = %d\n", n );
    return 0;
}

int sum( int n );
{
    int s = 0;
    for ( ; n > 0; n-- )
        s += n;
    return s;
}
```

A large program is written in a separate directory with many .c and .h files.

```c
#include <stdio.h>
#define N 10
void foo( int a );
int bar( int b );

#include "myheader.h"
int main( void )
{
    int n = N;
    printf("%d",foo(n));
    return 0;
}

#include "myheader.h"
void foo( int m )
{
    int p;
    printf("%d", bar( p*m ));
    return;
}

#include "myheader.h"
int bar( int m )
{
    return m+m;
}
```
Assertions

- In the form of,
  - `assert( expr )`
- In the standard header file `assert.h`
  - Need to include `assert.h` to use assertions
- If `expr` is 0, the system will print a message, and the program will be aborted
- Examples
  ```c
  int main(void)
  {
      int a, b;
      ...
      b = foo( a );
      assert( b > 0 );
      ...
  }
  ```
Scope Rules

- Identifiers are accessible only within the block in which they are declared
- Examples

```java
{  
    int a = 1, b = 2, c = 3;
    {
        int b = 4;
        float c = 6.3;
        a = b;
        {
            int c;
            c = b;
        }
    }
}
```
Auto and Extern Storage Classes

- auto
  - Variables declared within function bodies and blocks
    - Temporary storage for the lifetime of the function or block
    - Default and implicit
    ```
    auto int a, b, c;
    ```
  
- extern
  - Variables declared outside a function
    - Permanent storage for the lifetime of the program
    - Implicit
  - All functions have external storage class
  - extern tells the compiler to look for the variable in this file or some other file
  ```
  extern int a, b, c;
  ```
Auto and Extern Storage Classes (contd.)

main.c
================================
int a, b, c;
void foo(void);
int main(void)
{
    a = b = c = 1;
    foo();
    printf("%d %d %d\n", a, b, c);
    return 0;
}

foo.c
=======

void foo(void)
{
    extern int a;
    int b, c;
    a = b = c = 4;
    return;
}
Register and Static Storage Classes

- **register**
  - Tells the compiler that the variable should be stored in a register if possible
  - For performance
    ```
    register int i;    /* ≡ register i */
    for (i = 0; i < n; i++) ...
    ```

- **static**
  - To make a local variable to retain its value when the block is reentered
  - Or, to use a variable as a local variable to the file it is declared
    ```
    void foo(void)
    {
        static int cnt = 0;
        cnt++;  
    }
    ```
Register and Static Storage Classes (contd.)

foo.c
===============
void foo(void)
{
    ...
}
static int s;
void bar(void)
{
    ... = s + 3;
}

main.c
===============
int main(void)
{
    extern int s;
    ...
    x = s * y;
    ...
}
Register and Static Storage Classes (contd.)

- Static functions
  - Visible only within the file in which they are defined

```c
static int foo(void);
void bar(void)
{
    ... foo() ...
}
static int foo(void)
{
    ...
}
```
A function is said to be recursive if it calls itself, either directly or indirectly

```c
int sum(int n)
{
    if (n <= 1)
        return n;
    else
        return (n + sum(n - 1));
}
```
Recursion (contd.)

- **Factorial**

  ```c
  int fact(int n)
  {
    if (n <= 1)
      return 1;
    else
      return (n * fact(n - 1));
  }
  ```
Recursion vs. Iteration

```c
int fact(int n)
{
    if (n <= 1) return 1;
    else return (n * fact(n - 1));
}

int fact(int n)
{
    int r = 1;
    for( ; n > 1; n--)
        r *= n;
    return r;
}
```
Exercises

Execute the following program multiple times and see what happens

```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <time.h>

int main(void)
{
    int a, b, i, cnt = 0;
    srand( time( NULL ) );
    for( i = 0; i < 1000; i++ ) {
        a = rand() % 5 + 1;
        b = rand() % 10 + 1;
        if ( b - a <= 1 )
            continue;
        assert( b - a > 5 );
        printf("%4d\n", ++cnt);
    }
    return 0;
}
```
Exercises (contd.)

- Write a coin-tossing program that uses the random-number generator `rand()` in the standard library to simulate the toss
Exercises (contd.)

- Tower of Hanoi
  - There are three towers labeled a, b, and c
  - There are $n$ disks on tower a in the decreasing order of their size from the bottom
  - Move all the disks on tower a to tower c
  - Any tower may be used for the intermediate placement of disks
  - Only one disk may be moved at a time
  - At no time can a larger disk be placed on a smaller disk

- Write a program that uses recursion to produce a list of moves that shows how to accomplish the task
- The number of the disks is the user input