Functions in C

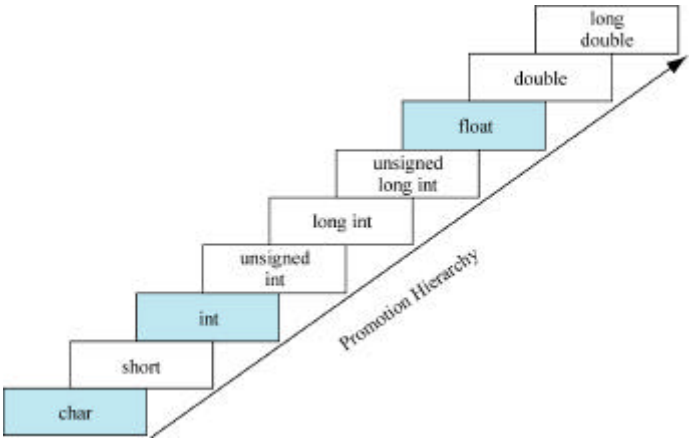
Comp 1402/1002

A Quick Note on Data Types

In C there are two options if a specific data type is provided and another is expected:

- The type is converted automatically
- The compiler insists you provide the right type.

Automatic Conversion



Forced Conversion (Casting)

- To go down the promotion hierarchy:

```
int x;
double y = 5.0;

x = (int) y;
y = (double) ‘c;
```

Developing Large Programs

- Difficult to manage long code
- Teams of people or lots of time required
- Easy forget or never know something

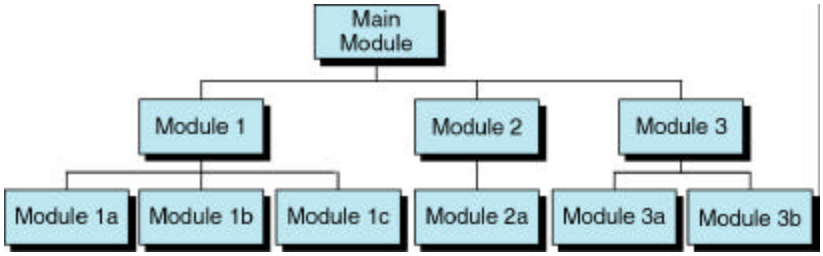
Result: Code should be organized

Top Down Design

Begins with a main module and divides it into related sub-modules. Each module is in turn divided into sub-modules.

The process of subdivision stops when a module is implicitly understood.

Module Example



Functions as Modules of Code

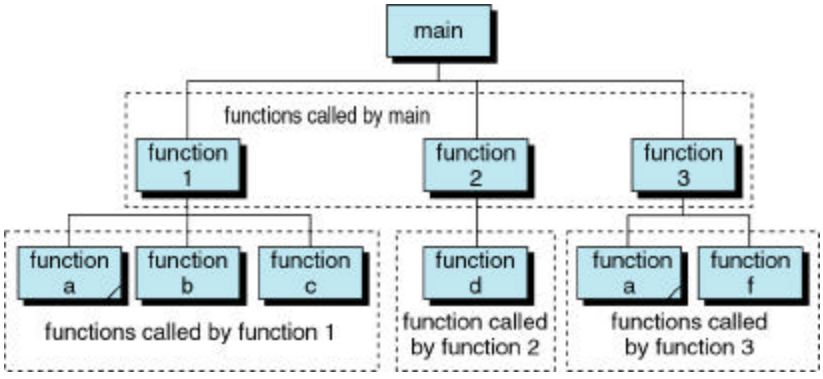
Functions are the modules in C

- Every program has a main function
- Sub modules are other functions
- Functions help organize and develop code

Functions Organize & Aid Development

- Allow organized development
- Develop and test small pieces of code
- Functions assignable to team members
- Allows code reuse

Function and Organization



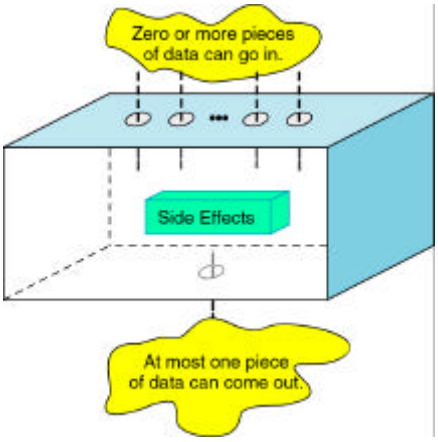
Calling Functions and Data

- Calling function passes zero or more parameters to called function

- Called function “returns” only one item

e.g. main passes parameters to function1

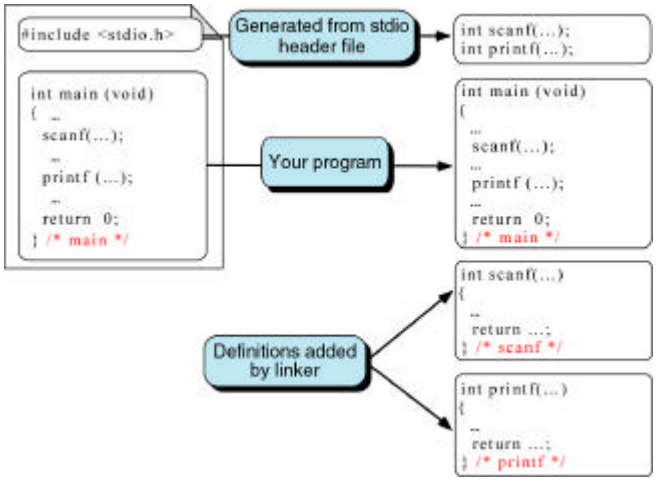
A Function



Libraries of Code

- Often languages come with libraries of code
- Provided “free of charge”
- Beware of Copyrights and Copylefts!
- Use library code whenever possible

Using a Standard C Library



Three Important C Libraries

stdio - Standard IO library

- File manipulation, input, output, definitions and manipulations of stdin stdout and stderr.

math – Math Library

- Defines most mathematical functions you would need.

stdlib – Standard library

- Memory and miscellaneous (some math here).

math & stdlib

Math

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

sometimes requires **`-lm`** as compiler option

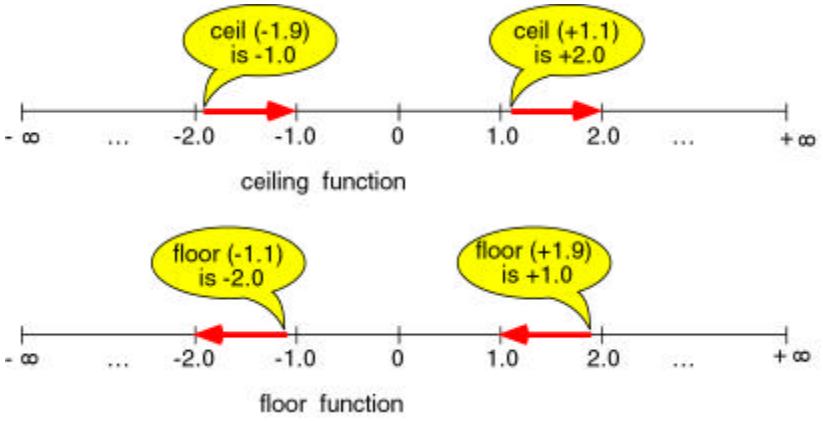
Stdlib

```
#include<stdlib.h>
```

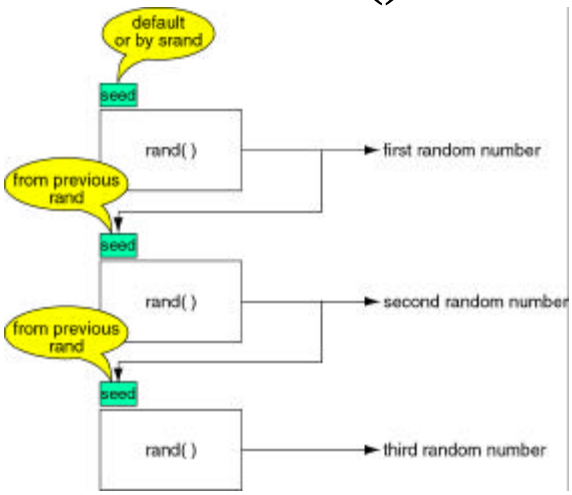
Some Library Functions

Name	Parameter	Returns	Example
`sqrt`	`double`	`double`	`sqrt(25.0)`
`pow`	`double`	`double`	`pow(2.0,3.0)`
`abs`	`int`	`int`	`abs(-10)`
`fabs`	`double`	`double`	`fabs(6.8)`
`ceil`	`double`	`double`	`ceil(1.7)`
`cos`	`double`	`double`	`cos(3.2)`
`sin`	`double`	`double`	`sin(3.2)`
`log`	`double`	`double`	`log(127.5)`

ceil() and floor()



rand()



Getting Random Numbers

Seed the generator:

```
unsigned int s = 2990;
srand(s);
```

Get “random” number”

```
int x = rand( );
/* returns 0 to RAND_MAX */
```

Nesting Function Calls

```
int i=10;
double y;

y = cos( pow( (double) i, 2)));
```

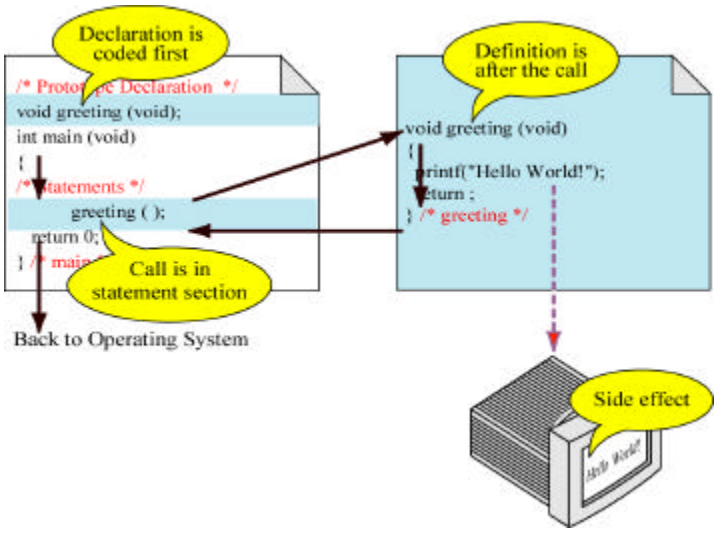
Convert i to a double (unnecessary)
Compute i squared
Compute cosine of i square

User Defined Functions

Three places a function name is used:

- Declaration (prototype)
- Call
- Definition

Declarations, Calls & Definitions



Declarations (Prototypes)

- Inform the calling method of the function
- Otherwise “implicit declaration” warning
  (may not be there on linking)

```
return_type function_name (parameter list);
```

Example Declarations

```
double area(double length, double width);
double perimeter(int radius);
```

Identical to:

```
double area(double, double);
double perimeter(int)
```

No Polymorphism!

C is unlike C++ and JAVA

Can only declare/define a function name **once** per program

Parameters and return type are irrelevant

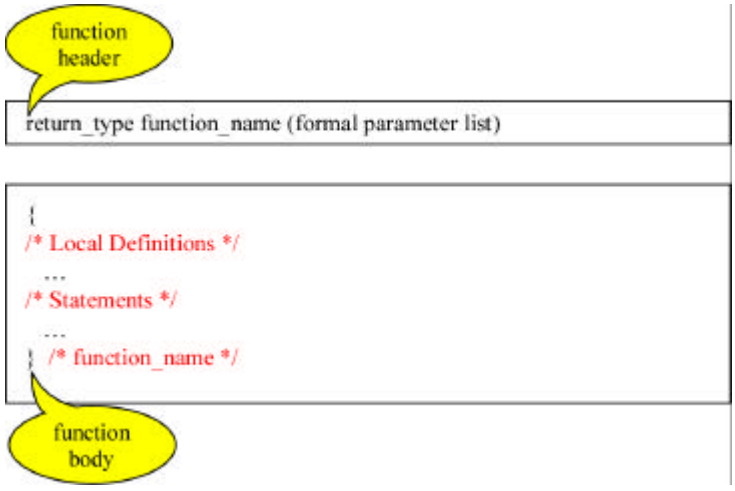
Name functions carefully (e.g. **`fabs, abs, labs`**)

Header Files

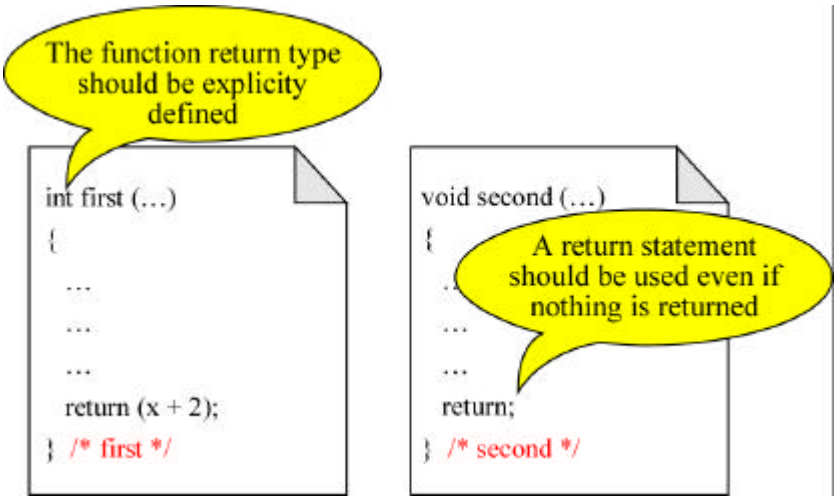
.h files usually contain function and constant declarations

- Single include gives many declarations
- Important if functions are in another file

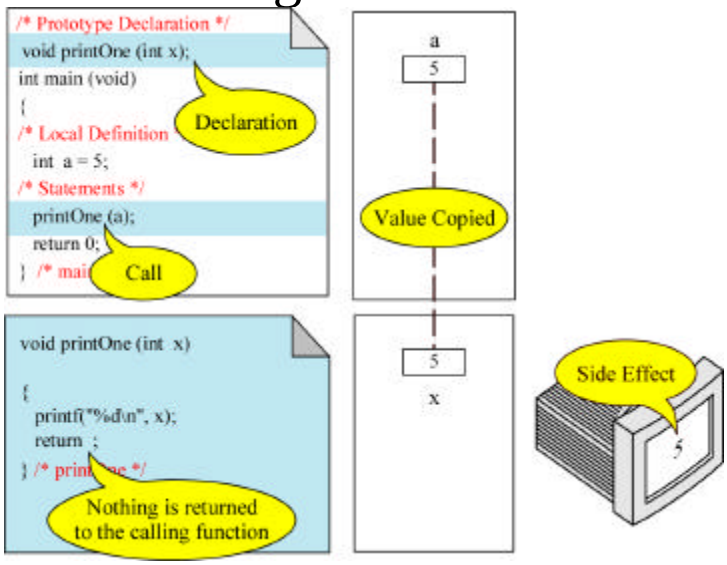
Function Definition



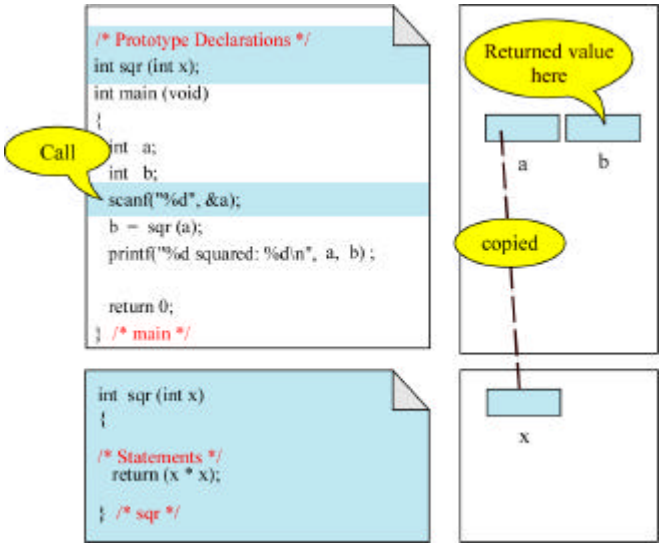
Return Type



Calling a Function



Calling a Function



Parameters

Parameters can be passed two ways:

- Pass by value
- Pass by address

Pass by Value

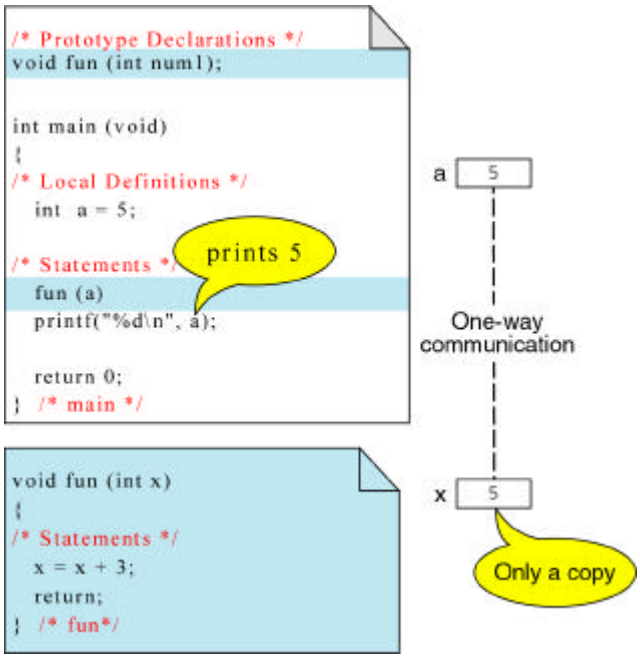
Copy the value:

From the caller

Into the formal parameter of the called

This is a one-way communication!

Pass by Value



Parameters

- Copy of the value is placed on the stack
- Transfer of control goes to the function
- Function can pick the parameters off the stack
- Original variables are untouched by function

Pass by Address

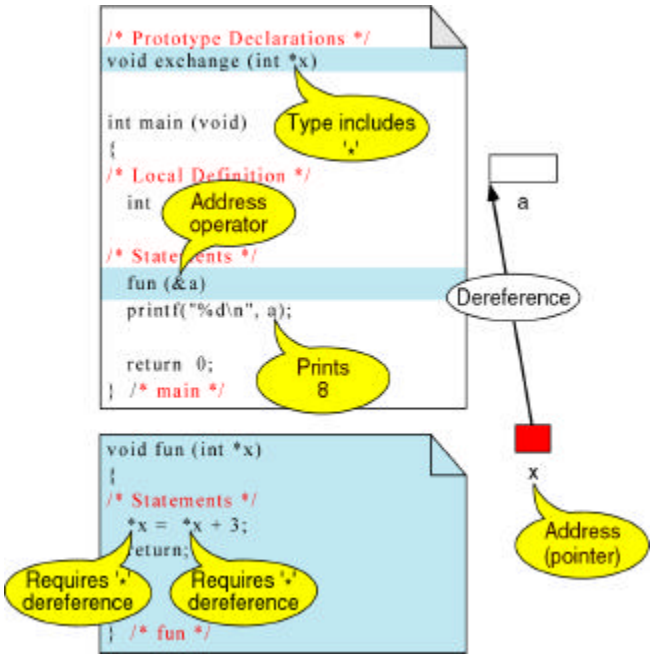
Instead of copying a value:

Copy the memory address!!!

Then work with that address.

Variables containing addresses are: **pointers**

Pass by Address



Address and Dereference

& is the address operator:

```
int a = 5;
&a        /* is the address to an int   */
          /* type is (int *)            */
```

* Is the dereference operator:

```
int *x; /* x is a pointer to an int */
*x = 12; /* contents of x equal 12 */
```

Scope and Duration

Global Scope

Anything defined is visible from its definition to the end of the program.

Local Scope

Good from their definition to the end of their block (normally a function)

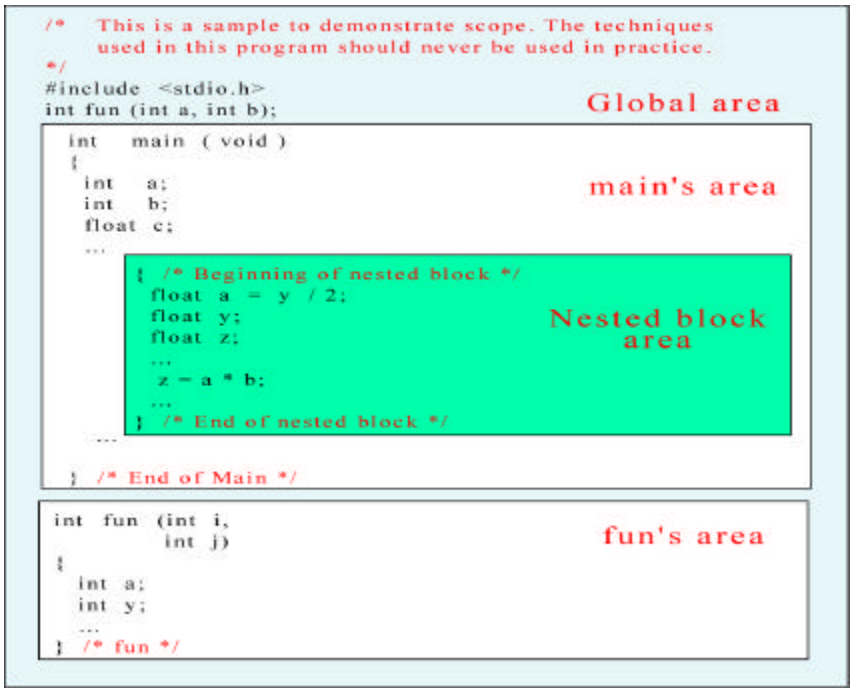
Scope and Duration

Global Variables

- Placed outside the functions that use them
- Available to all functions
- Global variables should be avoided

Local Variables

- Scope is extent of their block
- Placed on the stack
- Destroyed on exit from function



`static`

If a local variable is modified with static then:

- Variable exists from program beginning to end
- Only available in the function
- Initialized with 0