Introduction to C

Functions and Make

Instructor: Yin Lou

01/28/2011

Math Functions

- Many math functions are defined in math.h

Math Functions

- Many math functions are defined in math.h
  - pow(a, b) - Compute a^b
  - exp(a) - Compute e^a
  - log(a) - Compute natural logarithm
  - log10(a) - Compute common logarithm
  - sqrt(a) - Compute square root
  - fabs(a) - Compute absolute value
  - ceil/floor - Round up/down value
  - cos, sin, tan
  - acos, asin, atan

Functions

- Purpose of functions

Functions

- Purpose of functions
    - Breaks a program into pieces that are easier to understand
    - Makes *recursive* algorithms easier to implement
    - Promotes code reuse

Functions

- Purpose of functions
  - Breaks a program into pieces that are easier to understand
  - Makes *recursive* algorithms easier to implement
  - Promotes code reuse
- Disadvantage of functions

Functions

- Purpose of functions
  - Breaks a program into pieces that are easier to understand
  - Makes *recursive* algorithms easier to implement
  - Promotes code reuse
- Disadvantage of functions
  - Function calls add some memory and time overhead

Functions

- Purpose of functions
  - Breaks a program into pieces that are easier to understand
  - Makes *recursive* algorithms easier to implement
  - Promotes code reuse
- Disadvantage of functions
  - Function calls add some memory and time overhead
- Functions in C

Functions

- Purpose of functions
  - Breaks a program into pieces that are easier to understand
  - Makes *recursive* algorithms easier to implement
  - Promotes code reuse
- Disadvantage of functions
  - Function calls add some memory and time overhead
- Functions in C
  - Similar to methods in Java
  - But C functions do not belong to a class. Every function is visible everywhere in the program.

A Simple Function

Compute *base*^{exp}

```
int power(int base, int exp)
{
    int i, p = 1;
    for (i = 1; i <= exp; ++i)
    {
        p *= base;
    }
    return p;
}
```

Simple Function in Context

```
#include <stdio.h>

int power(int base, int exp); // function prototype

void main() // function definition
{
    int i = 3, j = 4;
    // function call
    printf("%d^%d is %d.\n", i, j, power(i, j));
}

int power(int base, int exp) // function definition
{
    int i, p = 1;
    for (i = 1; i <= exp; ++i)
    {
        p *= base;
    }
    return p;
}
```

Function Return Values

- If a function returns type void, then no return statement is needed.
- If a function returns another type, then a return statement is required along all possible execution paths.

Function Return Values

- If a function returns type void, then no return statement is needed.
- If a function returns another type, then a return statement is required along all possible execution paths.

What does this code do?

```
#include <stdio.h>

int foo(int arg)
{
    if (arg == 1)
    {
        return 1;
    }
}

void main()
{
    printf("%d\n", foo(0));
}
```

Call by Value

- Function arguments in C are passed *by value*

Call by Value

- Function arguments in C are passed *by value*
  - The *value* of the argument is passed, not a reference
  - Functions are given a new copy of their arguments
  - So a function can't modify the value of a variable in the calling function (unless you use pointers)

Call by Value

- Function arguments in C are passed *by value*
  - The *value* of the argument is passed, not a reference
  - Functions are given a new copy of their arguments
  - So a function can't modify the value of a variable in the calling function (unless you use pointers)

Example

```
#include <stdio.h>

int foo(int a)
{
    a = 3;
    return a;
}

void main()
{
    int a = 1, b;
    b = foo(a);
    printf("%d %d\n", a, b); // Output 1 3
}
```

Call by Value

Example

```
#include <stdio.h>

void swap(int a, int b)
{
    int t = a;
    a = b;
    b = t;
}

void main()
{
    int a = 1, b = 2;
    swap(a, b);
    printf("%d %d\n", a, b); // Output 1 2
}
```

Call by Value

- Call by value has advantages and disadvantages

Call by Value

- Call by value has advantages and disadvantages
  - Advantage: some functions are easier to write

Call by Value

- Call by value has advantages and disadvantages
  - Advantage: some functions are easier to write

```
int power(int base, int exp)
{
    int result = 1;
    for (; exp >= 1; --exp)
    {
        result *= base;
    }
    return result;
}
```

Call by Value

- Call by value has advantages and disadvantages
    - Advantage: some functions are easier to write

```
int power(int base, int exp)
{
    int result = 1;
    for (; exp >= 1; --exp)
    {
        result *= base;
    }
    return result;
}
```

- Disadvantage: sometimes youd like to modify an argument (e.g. swap() function)
    - Well see how to do this using pointers later

Recursion

Example

```
int fact(int n)
{
    if (n == 0)
    {
        return 1;
    }
    else
    {
        return n * fact(n - 1);
    }
}
```

Declaration and Definition

Declaration

A declaration announces the properties of a variable (primarily its type).

Example:

```
extern int n;
extern double val[];
```

Definition

A definition also causes storage to be set aside.

Example:

```
int n;
double val[MAX_LEN];
```

Manage Your Project

- It's always recommended to modularize your project. How?

Manage Your Project

- It's always recommended to modularize your project. How?
- Write functions and paste them in new file?

Manage Your Project

- It's always recommended to modularize your project. How?
- Write functions and paste them in new file?
- Definitions and decelerations are shared among a lot of source files. How to centralize this, so that there is only one copy to get and keep right as the program evolves?

Manage Your Project

- It's always recommended to modularize your project. How?
- Write functions and paste them in new file?
- Definitions and decelerations are shared among a lot of source files. How to centralize this, so that there is only one copy to get and keep right as the program evolves?
- We could use header files.

Header File

- Place common material in a header file.
- Can be *included* as necessary.

Header File

- Place common material in a header file.
- Can be *included* as necessary.

Example: mymath.h

```
int fact(int n);
int power(int base, int exp);
```

Example

power.c

```
#include "mymath.h"

int power(int base, int exp)
{
    int result = 1;
    int i;
    for (i = 1; i <= exp; ++i)
    {
        result *= base;
    }
    return result;
}
```

fact.c

```
#include "mymath.h"

int fact(int n)
{
    if (n == 0)
    {
        return 1;
    }
    else
    {
        return n * fact(n - 1);
    }
}
```

Example

main.c

```
#include <stdio.h>
#include "mymath.h"

void main()
{
    printf("%d\n", power(5, 3));
    printf("%d\n", fact(5));
}
```

How to Compile

```
$ gcc -O main.c fact.c power.c -o test
$ ./test
125
120
```

Introduction to make

- Large projects have complex dependencies.

Introduction to make

- Large projects have complex dependencies.
- There is a UNIX command called make that can help you compile your project.

Introduction to make

- Large projects have complex dependencies.
- There is a UNIX command called make that can help you compile your project.
- You write a file named Makefile, which just sits in the same directory as your project.

Introduction to make

- Large projects have complex dependencies.
- There is a UNIX command called make that can help you compile your project.
- You write a file named Makefile, which just sits in the same directory as your project.
- Describes what source files are used to build which object files, what headers they depend on, and so forth.

Introduction to make

- Large projects have complex dependencies.
- There is a UNIX command called make that can help you compile your project.
- You write a file named Makefile, which just sits in the same directory as your project.
- Describes what source files are used to build which object files, what headers they depend on, and so forth.

Makefile

```
defult:
    gcc main.c fact.c power.c -o test
clean:
    rm test
```

Another Example

Makefile

```
CC:=gcc
OPTIONS:=-O2 -shared -fPIC
LIB_PATH:=-pthread

SRC_DIR:=src
DST_DIR:=bin

default:
    $(CC) $(OPTIONS) $(LIB_PATH)  \
    $(SRC_DIR)/*.c -o $(DST_DIR)/libMath.so
clean:
    cd $(DST_DIR); rm libMath.so
```