CS1100
Introduction to Programming

Functions

Madhu Mutyam
Department of Computer Science and Engineering
Indian Institute of Technology Madras

Functions = Outsourcing

- Break large computing tasks into small ones
- Helps you to build on what others have done
  - You and others write functions
  - When you want to build a program, find out how to use the function and use it
- Using standard functions provided by the library
  - You are hidden from the implementation
  - Example – you don't have to worry about how *pow*(*m*, *n*) is implemented
- As engineers from different disciplines you will use and develop different set of functions

Modular Programming

- Subprograms
  - functions in C, C++, procedures and functions in Pascal
  - facilitate modular programming
    - Overall task is divided into modules
    - Each module - a collection of subprograms
  - a subprogram may be invoked at several points
    - A commonly used computation
  - hiding the implementation
  - changes can be incorporated easily

Example of Function Sets

- String manipulation
- Mathematical
- Finite Element Method
  - Used in structural analysis by Mechanical, Civil, Aero, etc. for stress calculations etc.
- Most function libraries cost a lot
  - Business opportunity – identify functions that are useful to your area of study, create libraries
- Functions for use in different software
  - Say, functions for web services

Power Function

```
#include <stdio.h>
int power (int, int);
main( ) {
for ( int i = 0; i < 20; i ++ )
   printf("%d %d %d\n", i, power(3,i), power(-4,i));
}
int power (int base, int n) {
   int i, p = 1;
   for ( i = 1; i <= n ; i ++)
      p  = p * base;
   return p;
}
```

function prototype

Computes the n^{th} power of base (1^{st} parameter)

Invocation with arguments

A block

Calling Power Function with i=3

```
printf("%d %d %d\n", i, power(3,i), power(-4,i));
```

27

-64

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

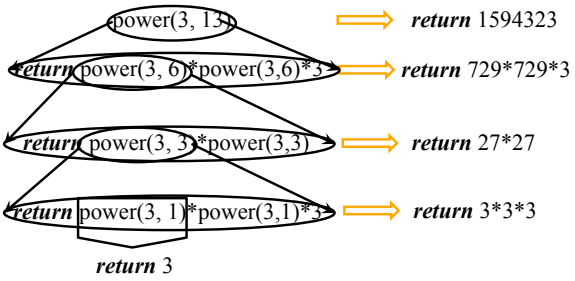
```
int power (int base, int n){
int i, p = 1;
for ( i = 1; i <= n ; i ++)
      p  = p * base;
return p;
}
```

Recursive Function Example

```
int power (int num, int exp) {
   int p;
   if (exp = = 1) return num;
   p = power(num, exp/2);
   if (exp % 2 = = 0) return p*p;
   else return p*p*num;}
```

The base case exp = 1
Guarantees termination

Recursive Function Example



Factorial (*n*)

$n! = 1 * 2 * 3 * .... * (n-2) * (n-1) * n$

Iterative version

```
int fact(int n){
   int i;
   int result;
   result = 1;
   for (i = 1; i <= n; i++)
         result = result * i;
   return result;
}
```

In practice *int* may not be enough!

Factorial (*n*) – Recursive Program

$n! = n * (n-1)!$

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

- Shorter, simpler to understand
- Uses fewer variables
- Machine has to do *more* work running this one!

Basics

- Function is a part of your program
  - It cannot be a part of any other function
  - main( ) is a function: it is the main function
    - Execution starts there or the control flow starts there
  - From there it can flow from one function to another, return after a computation with some values, probably, and then flow on

Basics

- Transfer of control is affected by calling a function
  - With a function call, we pass some parameters
  - These parameters are used within the function
  - A value is computed
  - The value is returned to the function which initiated the call
  - The calling function can ignore the value returned
  - It could use it in some other computation
  - A function could call itself, these are called *recursive function* calls

Add Functions to Your Program

- A program is a set of variables, and assignments to variables
- Now we add functions to it
  - Set of variables
  - Some functions including main( )
  - Communicating values to each other
  - Computing and returning values for each other
- Instead of one long program, we now write a structured program composed of functions

Features

- C program -- a collection of functions
  - function main ( ) - mandatory - program starts here.
- C is not a block structured language
  - a function cannot be defined inside another function
  - only variables can be defined in functions / blocks
- Variables can be defined outside of all functions
  - global variables - accessible to all functions
  - a means of sharing data between functions - caution
- Recursion is possible
  - a function can call itself - directly or indirectly

Function – General Form

return-type function-name (argument declarations)

```
{
    declaration and statements
    return expression;
}
```


Function Definition in C

- ***return-type*** function-name (argument declarations)
  {variable/constant declarations and statements}
- Arguments or parameters:
  - the means of giving input to the function
  - type and name of arguments are declared
    - names are formal - local to the function
- Return value: for giving the output value
  - ***return*** (expression); -- optional
- To invoke a function

  function-name(exp1,exp2,…,expn);

No function declarations here!

Matching the number and type of arguments

Function Prototype

- Used by the compiler to check the usage
  - prevents execution-time errors
- Defines
  - the number of parameters, type of each parameter,
  - type of the return value of a function
- Ex: function prototype of power function:

  int power (int, int);
  - no need for naming the parameters
- Function prototypes are given in the beginning

More on Functions

- To write a program
  - You could create one file with all the functions
  - You could/are encouraged to identify different modules and write functions for each module in a different file
  - Each module will have a separate associated header file with the variable declaration global to that module
  - You could compile each module separately and a .o file will be created
  - You can then cc the different .o files and get an a.out file
  - This helps you to debug each module separately

Running with Less Memory

- Functions
  - Provided to break up our problem into more basic units
  - Control flow – flows from function to function, saving the current context, changing contexts, then returning…..
  - Helps the program to run with lesser memory, but slightly slower than a monolithic program without functions
- Typically functions communicate using the arguments and return values

Call by Value

- In C, function arguments are passed "by value"
  - values of the arguments given to the called function in temporary variables rather than the originals
  - the modifications to the parameter variables do not affect the variables in the calling function
- "Call by reference"
  - variables are passed by reference
    - subject to modification by the function
  - achieved by passing the "address of" variables

Call by Value – An Example

```
main( ) {
int p = 1, q = 2, r = 3, s;
int test(int, int, int);
…;
s = test (p, q, r); …     /* s is assigned 9 */
}          /* p,q,r don't change, only their copies do */

int test( int a, int b, int c){
   a ++; b ++; c ++;
   return (a + b + c);
}
```

Function prototype

Function call

Function definition

Call by Reference

```
#include <stdio.h>
void quoRem(int, int, int*, int*);    /*addresses or pointers*/
main( ){
   int x, y, quo, rem;
   scanf("%d%d", &x, &y);
   quoRem(x, y, &quo, &rem);
   printf("%d  %d", quo , rem);
}

void quoRem(int num, int den, int* quoAdr, int* remAdr){
   *quoAdr  =  num / den;   *remAdr  =  num % den;
}
```

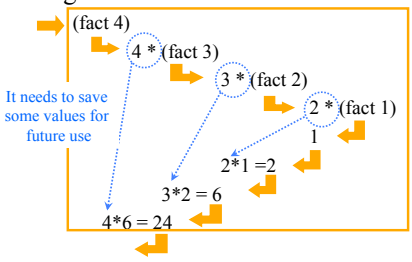
Passing addresses

Does not return anything

Pending Computations

- In this recursive version the calling version still has pending work after it gets the return value.

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



Tail Recursion

```
int fact(n)
{    return fact_aux(n, 1); }

int fact_aux(int n, int result)
{
if (n == 1) return result;
return fact_aux(n - 1, n * result)
}
```

Auxiliary variable

The recursive call is in the return statement. The function simply returns what it gets from the call it makes. The calling version does not have to save any values!