Esc101: Functions

Instructor: Krithika Venkataramani
Semester 2, 2011-2012

The contents of these slides are from the lecture slides of Prof. Arnab Bhattacharya

Functions

- Functions are useful
  - To repeat the same task
  - To modularize the program and increase the understanding
  - To correct errors (called "bugs") in the program
- They are defined using the following format
  return_type function_name (parameter_list) /*function header*/
  {
    ... // function body
  }
- They are called (from other functions) using
  variable = function_name (argument_list)

Function definition

- function_name is the name of the function
  - Follows the same rules as variable names
- return_type is the type of value that the function returns
  - If a function does not return anything, it is specified as void
- parameter_list is a comma-separated list that describes the parameters of the function including their name and type
- Function body may contain any valid C code
- Local variable declarations may be present
  - These are visible only within this specific function
- If there is a return type other than void, the function body must have a return statement
Function to add two integers

```c
#include <stdio.h>
int sum(int a, int b)
{
    int c;
    c = a + b;
    return c;
}
```

```c
void main()
{
    int x, y, z;
    scanf("%d", &x);
    scanf("%d", &y);
    z = sum(x, y);
    printf("%d
", z);
}
```

Function to add two floats

```c
#include <stdio.h>
int sum_i(int a, int b)
{
    int c;
    c = a + b;
    return c;
}
```

```c
float sum_f(float a, float b)
{
    float c;
    c = a + b;
    return c;
}
```

Use of two functions to add integers and floats

```c
int main()
{
    float a = 4.2, b = 3.6, c;
    int x = 4, y = 3, z;
    c = sum_f(a, b);
    z = sum_i(x, y);
    printf("%f + %f = %f
", a, b, c);
    printf("%d + %d = %d
", x, y, z);
}
```

Example programs with functions

```c
Write a function to find maximum of 2 integers. Use the function to find the maximum of 4 integers.
```

```c
#include<stdio.h>
int max(int x, int y)
{
    if (x > y)
        return x; //x is max
    else if (y > x)
        return y; //y is max
    else
    {
        printf("The integers are equal
");
        return x; //must return an integer
    }
}
```
Using the function for finding max of 4 integers

```c
void main()
{
    int a, b, c, d, m1, m2, m3;
    //read input
    m1 = max(a, b); //m1 is max of a and b
    m2 = max(c, d); //m2 is max of c and d
    m3 = max(m1, m2); //max of all 4 integers
    printf("Maximum of all 4 integers = %d", m3);
}
```

Scope of variables for blocks of code

- A variable retains its visibility only in the block where it is declared
- Variable declared in an outer block is visible in all inner blocks
- Variable declared in an inner block is not visible in outer blocks

Examples on scope of variables in blocks of code

- Error in line 5 as i is not visible outside the block
  1: {
  2:     int i = 5;
  3:     i --;
  4: };
  5: i ++;
- No problems below as i is visible in all inner blocks
  1: int i = 5;
  2: {
  3:     i ++;
  4: };
  5: i --;

Scope of variables for blocks of code

- If inner block declares a variable having the same name as that of a variable in the outer block, the one in the outer block is not visible in the inner block
- Variable in the outer block becomes visible once more when the inner block is completed
- Inner block variable may even have a different type
Examples on scope of variables in blocks of code

1: double i = 6.0;
2: {
3:     int i = 5;
4:     i = i*10;
5: }
6: if (i%10==0)
7:     printf("i is divisible by 10\n");

Error in line 6 as i is not an integer outside the block

Scope of variables in functions

- Rules are same for blocks in the function
- There is no inner and outer functions
- So, no variable of a function is visible in any other function
- This includes variables in the parameter list

Call by value
- Functions get access to the values of the variables
- They do not access the variables passed to them
- Important: Functions cannot change the value of a variable passed to it

Example on function call by value

```c
#include <stdio.h>
void swap ( int a, int b)
{
    int t = a;
    a = b;
    b = t;
    printf("In Swap Function: a = %d, b = %d\n", a,b);
}
void main ()
{
    int x = 3, y = 4;
    printf("x = %d, y = %d\n", x, y);
    swap (x, y);
    printf("x = %d, y = %d\n", x, y);
}
```

Recap on Functions

- Code segments can be written in functions
- Functions are useful
  - if the same code segment is needed to be written many times
  - modularity: to understand the purpose of a code segment better
- Each function need to be defined separately
- A function can have multiple input parameters and one output value
- Scope: The variable names for input/output parameters as well as other variables in the function are only visible inside the function
- A function can be called from other functions
- When called, the argument values are passed to the function input parameters
Arrays in Functions

- Arrays can be passed as parameters to functions
- Arrays cannot be returned
- Arrays are not called by value
- Only the array name is passed, and its contents are not copied
- Functions receive the original array
- Important: Functions can change the values of array elements
  - The change in array is visible outside the function

Modifying arrays though functions

```c
#include <stdio.h>

void swap ( int a[])
{
    /* swapping elements in array*/
    int t = a[0];
    a[0] = a[1];
    a[1] = t;
}

int main ()
{
    int x[] = {3, 4}, y[] ={2,5};
    printf ("x[0] = %d, x[1] = %d\n", x[0] , x[1]) ;
    swap(x);
    swap(y);
    printf ("x[0] = %d, x[1] = %d\n", x[0] , x[1]) ; //function swaps elements
}
```

Passing one-dimensional arrays to functions

- Size of array is not required in the parameter list
- Suppose size of array in parameter list is \( p \)
- Suppose size of actual array that is passed is \( a \)
- If \( p = a \), no issues
- If \( p < a \), only the first \( p \) elements of array would be accessed in the function
- If \( p > a \), extra \( (p - a) \) elements of array in function are filled up with unknown values

Passing array dimension in parameter list

```c
#include <stdio.h>

void p_equal ( int a[2])
{
    printf (" Equal : a[0] = %d, a[1] = %d\n", a[0] , a[1]) ;
}

void p_less ( int a[1])
{
    printf (" Less : a[0] = %d\n", a[0]) ;
}

void p_more ( int a[3])
{
} //value of a[2] printed is unknown
```
Passing array dimension in parameter list (cont.)

```c
void main ()
{
    int x[2] = {3, 4};
    printf(" Original : x[0] = %d, x[1] = %d\n", x[0], x[1]);
    p_equal(x);
    p_less(x);
    p_more(x);
}
```

Sample program to search for a substring

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

int substring(char small[], char large[])
{
    int lengthsmall, lengthlarge, j, k, match=0;
    lengthsmall = strlen(small);
    lengthlarge = strlen(large);
    //search substring. If substring found, set match=1, else match=0
    return match;
}
```

```
for(j =0; j<=lengthlarge-lengthsmall; j++)
{ //outer loop: the start of the sequence in string 'large'
    match = 1;
    for (k = 0; k<lengthsmall; k++) //inner loop to check substring
        if(small[k]!=large[j+k])
            { /*not a substring if even one character in sequence is different*/
                match = 0;
                break; //no need to check further
            }
    if (match==1) //substring found
        {
            printf("Matching substring %s found at location %d from the
                beginning of string %s\n", small, j+1, large);
            break; //stop searching if one substring is found
        }
}
```

Calling the substring search function

```
void main()
{
    char str1[50], str2[20];
    int strfound;
    printf("Enter the larger string:");
    scanf("%s", str1);
    printf("Enter the substring to be searched for:");
    scanf("%s", str2);
    strfound = substring(str2, str1);
    if(strfound==1)
        printf("Substring found\n");
    else
        printf("Substring not found\n");
}
```
Returning multiple outputs using an array as input

- More than one output from a function can be obtained using arrays.
- The multiple outputs of the same variable type can be stored in an array, which is passed as an input parameter.
- This is because the change in arrays inside the function is visible outside.

Returning substring match and position of substring using an array

```c
#include<stdio.h>
#include<string.h>
#include<ctype.h> //needed for library function toupper() that converts a lowercase letter to an uppercase

void substring(char small[], char large[], int result[2])
{
    int lengthsmall, lengthlarge, j, k;
    lengthsmall = strlen(small);
    lengthlarge = strlen(large);
    for(j = 0; j <= (lengthlarge-lengthsmall); j++) //outer loop: the start of the sequence check in large
    {
        result[0] = 1;
        for(k = 0; k < lengthsmall; k++) //inner loop to check substring
        {
            if (toupper(small[k]) != toupper(large[j+k])) //independent of lower-case/upper-case letters
                result[0] = 0; /*not a substring if even one character in sequence is different*/
                break; //no need to check further
        }
        if (result[0] == 1) //substring found
            {
                result[1] = j+1; //index of the start sequence
                break; //stop searching if one substring is found
            }
    }
}
```

Getting multiple outputs from function

```c
void main()
{
    char str1[50], str2[20];
    int result[2] = {0,0};
    printf("Enter the larger string:");
    scanf("%s", str1);
    printf("Enter the substring to be searched for:");
    scanf("%s", str2);
    substring(str2, str1, result); //2 outputs stored in array result
    if(result[0] == 1)
        {
            printf("Substring found
");
            printf("Matching substring %s found at location %d from the beginning of string %s\n", str2, result[1], str1);
        }
    else
        printf("Substring not found\n");
}
```
Passing 2-D arrays to functions

- Second size, i.e., number of columns is required in the parameter list
- void mul( int a[ ][3], int b[ ][2], int c[ ][2], int size )
  
  /* multiplication of a matrix of size 2x3 (a) with a matrix of size 3x2 (b) */
  int i, j, k;
  for (i = 0; i < size; i++)
    for (k = 0; k < 2; k++)
      { c[i][k] = 0;
        for (j = 0; j < 3; j++)
          c[i][k] = c[i][k] + a[i][j] * b[j][k];
      }

- Can be called using mul(a, b, c, 2);
- For multi-dimensional arrays, all sizes except the first are required.

Matrix multiplication cont.

- void main ()
  
  { int a[2][3], b[3][2], c[2][2];
    int i, j, k;
    for (i = 0; i < 3; i++)
      for (j = 0; j < 2; j++)
        scanf("%d", &a[i][j]); /*reading elements of a*/
    for (j = 0; j < 3; j++)
      for (k = 0; k < 2; k++)
        scanf("%d", &b[j][k]); /*reading elements of b*/
    mul(a, b, c, 2); /*c is the matrix product of a and b*/
  }

Recursion

- A function can call itself
- When a function is defined in terms of itself, it is called a recursive function and the process is called recursion
- Between calls, the argument must change
- There must be at least one base case in the recursive function
- The argument values must proceed towards the base case
- If these are not satisfied, the recursive function will not terminate

Function to compute Factorial by recursion

- # include <stdio.h>
- long int fact (int n)
  
  { if ((n == 1) || (n == 0))  //base case
      return 1;
    return n * fact(n - 1); /*argument change proceeding to base case*/
  }

- int main ()
  
  { int n;
    scanf("%d", &n);
    printf("Factorial = %ld", fact(n));
  }
Double recursion
A function may call itself twice (or multiple times)

```c
#include <stdio.h>
int choose(int n, int k)
{
    printf("Calling %d choose %d\n", n, k);
    if ((k == 0) || (n == k)) //base case
        return 1;
    return choose(n - 1, k) + choose(n - 1, k - 1); //calling itself twice
}
int main()
{
    int n, k;
    scanf("%d", &n);
    scanf("%d", &k);
    printf("%d choose %d is %d\n", n, k, choose(n, k));
}
```

Mutual recursion
When two functions are defined in terms of each other, they are called mutually recursive functions and the process is called mutual recursion

```c
int odd(int n)
{
    if (n == 1) //base case
        return 1;
    return even(n - 1);
}
int even(int n)
{
    if (n == 1) //base case
        return 0;
    return odd(n - 1);
}

int main()
{
    int n;
    scanf("%d", &n);
    if (n <= 0)
    {
        printf("%d must be a positive integer \n");
        return -1; //program ends
    }
    if (odd(n)) // if (odd(n) == 1)
        printf("%d is odd \n", n);
    else
        printf("%d is even \n", n);
}
```

Mutual recursion to find if a number is even or odd

```c
int main()
{
    int n;
    scanf("%d", &n);
    if (n >= 0)
    {
        printf("%d must be a positive integer \n");
        return -1; //program ends
    }
    if (odd(n)) // if (odd(n) == 1)
        printf("%d is odd \n", n);
    else
        printf("%d is even \n", n);
}
```