
16.070 Introduction to Computers and Programming

April 11

Recitation 9

Spring 2002

Topics:

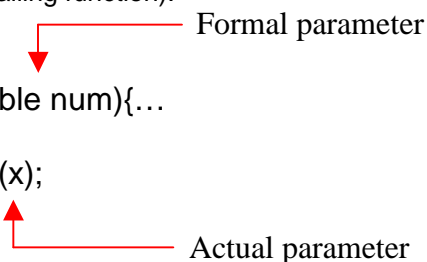
- Function Review
- Sorting and Searching
- Recursion
- Big O Notation
- Serial I/O
- Other

Function Review

$$f(x) = \sqrt{x}$$

$$f = \text{sqrt}(x)$$

1. Analogy: think of C functions as mathematical functions or as black boxes. You give the function some inputs, it calculates something, and it returns the some output.
2. Parameters: The inputs to your function are called parameters. The formal parameters are variables that get used inside your function. The actual parameters are variables that are used outside your function (inside the calling function).


double sqrt(double num){...

f = sqrt(x);

3. Do the formal and actual parameters have to use the same name? No! They are completely different variables. All of the following are legal statements.

Mathematical

$$F = \sqrt{4}$$

$$X = \sqrt{16}$$

$$C = \sqrt{X}$$

C

```
double F;  
F = sqrt(4.0);  
double X;  
X = sqrt(16.0);  
double C;  
double X;  
C = sqrt(X);
```

4. Do you include the type of the parameters when you call a function? No! You only include the type of the input and output variables when you define/declare/prototype the function. It's similar to declaring any other variable.

More on functions, scope, and memory later...

Sorting and Searching

How do you find a piece of data in a very long array?

1. Linear Search

- a. Start at the beginning
- b. Look at each element until you find what you are looking for.
- c. Code:

```
while(index < ARRAY_MAX && array[index] != target)
{
    index++;
}
```
- d. Array can be in any order.
- e. Worst case scenario: you look at all ARRAY_MAX elements.

2. Binary Search

- a. Start at the middle
- b. If target is less than middle element, then repeat with the left half of the array.
- c. If target is more than middle element, then repeat with the right half of the array.
- d. Code:

```
int BinarySearch( int target, int array[], int size)
{
    int start=0, end=size, middle;
    while(end-start > 0)
    {
        middle = end-start/2;
        if(target > array[middle])
            start = middle+1;
        if(target < array[middle])
            end = middle-1;
        if( target == array[middle])
        {
            printf("Found %d at index %d",target, middle);
            return middle;
        }
    }
}
```
- e. Array must be sorted in increasing order
- f. Worst case senario: you look at $\log(N)$ elements.

How do you put a large array in order?

1. When you create it.

2. Bubble Sort

- a. Start at the beginning
- b. Compare adjacent elements (element1 and element2)
- c. If element1 > element2 swap postions
- d. Compare next adjacent elements (element2 and element3)
- e. Repeat for entire array
- f. Repeat a-e ARRAY_SIZE times
- g. Code:

```
void BubbleSort( int array[], int size )
```

```

{
    int pass,i,tmp;
    for( pass=0; pass<size-1; pass++){
        for( i=0; i<size-1; i++){
            if( array[i]>array[i+1] ){
                tmp=array[i+1]; /* swap i,i+1 */
                array[i+1] = array[i];
                array[i] = tmp;
            } /* end if */
        } /* end for i */
    } /* end for pass */
} /* end BubbleSort */

```

3. Selection Sort

- a. Start at beginning
- b. Find the smallest element
- c. Swap with the first element
- d. Look at array minus first element
- e. Repeat steps a-d ARRAY_SIZE – 1 times
- f. Code: void SelectionSort(int array[], int size)

```

{
    int i, smallest, temp;
    for( i=0; i<size-1; i++){
        smallest = findIdxOfSmallest(array, i, size-1);
        tmp=array[smallest];
        array[smallest] = array[i];
        array[i] = tmp;
    } /* end for i */
} /* end SelectionSort */

```

- g. Must have some function findIdxOfSmallest() to find the smallest element of the array.

4. Merge Sort

- a. Split the array into two parts
- b. Merge Sort each part
- c. Shuffle the two parts back into one array
- d. Code: ...
- e. Recursive Algorithm (see side topic)
- f. Faster than Bubble or Selection Sort

Side Topic: Recursion

Do not use recursion in real time programs or any of your homework programs for this class. This section is FYI only.

A recursive function is a function that calls itself. (This topic is optional, but if you want to know more see page 210 in the C book.)

Example: Compute factorial recursively

```

Long factorial(int k)
{
    if (k == 0)
        return 1;
    else

```

```

        return k*factorial(k-1);
    }

```

Execution walk-through:

X=4

Ans = factorial(X)

Expand all calls to function factorial

```

    K = 4
    Return 4*factorial(3)
        K = 3
        Return 3*factorial(2)
            K = 2
            Return 2 * factorial(1)
                K = 1
                Return 1*factorial(0)
                    Return 1;

```

Now all the return statements propagate back up the sequence.

(Step 1) K = 4 Return 4*factorial(3) K = 3 Return 3*factorial(2) K = 2 Return 2 * factorial(1) K = 1 Return 1*1	(Step 2) K = 4 Return 4*factorial(3) K = 3 Return 3*factorial(2) K = 2 Return 2 * 1
(Step 3) K = 4 Return 4*factorial(3) K = 3 Return 3*2	(Step 4) K = 4 Return 4*6
(Step 5) Ans = 24;	

Order of Magnitude (Big O)

O = order of magnitude upper bound on execution time

How many operations does it take to do a task? As you add elements, how does the number of operations change?

Notation Rules

- Ignore constant factors $O(N+3) = O(N)$
- Ignore smaller terms $O(N^2 + N) = O(N^2)$
- Ignore base of logarithm $O(\log_{10} N) = O(\log N)$

Since $O(\text{foo})$ is the **upper bound** on execution time, to find the Big O of a process think of the worst case senario.

Example1: Linear Search

Worst case senario – have to look at all N elements of the array

$O(N)$

The execution time (number of operations) of a linear search is proportional to and grows as the number of elements in the array.

Example2: Selection Sort

Find the smallest element – N steps

Swap with first element – 1 step

Find and Swap N -1 times

$O((N-1)(N+1)) = O(N^2 - 1) = O(N^2)$

The execution time of a selection sort is proportional to and grows as the square of the number of elements in the array

Binary Search: $O(\log N)$

Bubble Sort: $O(N^2)$

Merge Sort: $O(N \log N)$

Which Big O is faster?

$$O(N^2) > O(N) > O(N \log N) > O(\log N) > O(1)$$

Warning: Constants hidden by the notation can be significant in the actual execution time.

Implementation of Serial Port Communication:

- Transmitting/Receiving bytes from/to the workstation
- Transmitting/Receiving bytes from/to the Handy Board
- Procedure for data transfer
- Example

In the final project you will need to transmit one character (1 byte) of information at a time between the Handy Board and the workstation.

Transmitting/Receiving from/to the workstation

The serial port library header file *serial_ws.h* (provided by 16.070) contains a function called *sio()* that is used to transmit/receive bytes from/to the workstation. The following prototype exists:

`DWORD sio(int io, LPVOID pdest, DWORD len);`

This function is available when including the *serial_ws.h* file in any .C program file:

`#include "serial_ws.h"`

Note: This library will only work under a Windows operating system.

To transmit:

Set:

io = 2,

pdest = address of the first byte of an array of size = *len* to be transmitted

len = number of bytes to transmit,

Example: transmit the character 'c' on the serial port:

`char c='c';`

`sio(2, &c, sizeof(c));`

To receive:

Set:

io = 1,

pdest = address of the first byte of an array of size = *len* where the received data will be stored

len = number of bytes to receive,

Example: receive a character on the serial port and place it in the variable c:

`char c;`

`sio(1, &c, sizeof(c));`

The default port setting for the serial library is COM1. This can be changed to COM2 by editing the line:

```
const LPCSTR LPSZPORTNUM = "com1";
```

to be:

```
const LPCSTR LPSZPORTNUM = "com2";
```

Transmitting/Receiving from/to the Handy Board

The Handy Board library *serial_hb.c* contains the functions:

```
void disable_pcode_serial();  
void enable_pcode_serial();  
void serial_putchar(int c);  
int serial_popchar(void);
```

The HB's serial port is used to download code from a workstation. After the download has been completed, it is possible to gain control of the serial port by calling the function `disable_pcode_serial()`. This allows us to communicate with the HB without it confusing the communication with a code download.

The HB can then transmit a single byte at a time across the serial port by using

```
serial_putchar(int c).
```

Example: To transmit the character 'd' from the HB:

```
char c='d';  
serial_putchar(c);
```

The HB can then receive a single byte at a time from the serial port by using

```
serial_popchar(int c).
```

Example: To receive a character from the Workstation:

```
char c;  
c = serial_popchar();
```

Always call `enable_pcode_serial()` to re-enable the code download function of the serial port before exiting your program, otherwise Interactive C will not be able to communicate with it once you are done.

Procedure for Data Transfer

The following steps are required to allow serial data transfer after you have written your HB and Workstation code:

On the Handy Board (HB):

1. Switch on the HB.
2. Start IC.
3. Download *serial_hb.c* to the HB.
4. Download your HB program to the HB.
5. Exit IC.
6. Reset the HB.

On the Workstation:

1. Run your Workstation code.

Example

The combination of these HB and Workstation programs transmits a "Hello" message from the workstation to the Handy Board and the acknowledges by sending the message "Ditto" from the HB to the Workstation.

Workstation Code:

```
/* TJ, March 2001 */
/* Serial Data Transfer */
/* PS7, Problem 3 */
#include <stdio.h>
#include "serial_ws.h" /* <----- */
#define msg_length 5

int main(void) {
    int i=0; /* iteration variable */
    int dummy=0;
    char send_array[msg_length]={'H','e','l','l','o'};
    char receive_array[msg_length]={0};

    /* Send Data */
    printf("\nSending Data: ");
    for (i=0; i<msg_length; i++) {
        printf("\nSend #: %d: %c",i,send_array[i]);
        sio(2, &send_array[i], 1); /* <----- */
    }

    printf("\nSuccess!\n");

    /* Receive data */
    for (i=0; i<msg_length; i++) {
        printf("\n%d: ",i);

        while (receive_array[i]==0) {
            sio(1, &receive_array[i], 1);* <----- */
        }
        printf(" %c\n",receive_array[i]);
    }
    printf("\nReceive Success!\n\n");

    /* Echo received data */
    for (i=0; i<msg_length; i++)
        printf("%c",receive_array[i]);

    printf("\n\n");

    return 0;
} /* end main */
```

Handy Board Code:

```
/* TJ, March 2001 */
/* PS7, Problem 3 */
/* HB Serial I/O */
#define msg_length 5

int main(void)
{
    char in_data[msg_length];
    char out_data[]={'D','i','t','t','o'};
    int i=0; /* iteration variable */
```

```

printf("\nWaiting...");

while(start_button()==0);
    printf("\nReady:");

/* disable serial pcode serial routines */
disable_pcode_serial(); /* <----- */

/* serial receive */
for (i=0; i<msg_length; i++) {
    in_data[i]=serial_popchar(); /* <----- */
}

printf("\nHere we go: ");
sleep(1.0);

/* print received msg */
printf("\n");
for (i=0; i<msg_length; i++)
    printf("%c",in_data[i]);

/* transmit msg */
for (i=0; i<5; i++) {
    sleep(0.05); /* <----- */
    serial_putchar(out_data[i]); /* <----- */
}

/* Enable pcode serial routines */
enable_pcode_serial(); /* <----- */

return 0;

} /* end main */

```