

CS 211

Debug Mode
and
Command Line Arguments

Debugging a Program

- One Common Approach: Adding lots of print statements

```
int binarySearch(int arr[], int l, int r, int x, int* numComps) {
    if (r >= l) {
        int mid = l + (r - l)/2;
        *numComps++;
        if (arr[mid] == x) return mid;
        if (arr[mid] > x) return binarySearch(arr, l, mid-1, x, numComps);
        return binarySearch(arr, mid+1, r, x, numComps);
    }
    return -1;
}
```

Debugging a Program: Step 1 – Wrong Result

- Called from main()

```
int main() {  
    int arr[10] = { 1, 3, 5, 7, 9, 11, 13, 15, 17, 19};  
    int numComps = 0;    int pos = 0;  
  
    pos = binarySearch ( arr, 0, 9, 11, &numComps);  
    printf ("Result: %d, numComps: %d\n", pos, numComps);  
  
    return 1;  
}
```

Produces the Following Result:

Result: 5, numComps: 0

Debugging a Program: Step 2 – Add printf()s

```
int binarySearch(int arr[], int l, int r, int x, int* numComps) {
    if (r >= l) {
        int mid = l + (r - l)/2;
        *numComps++;
        printf ("mid val: %d, mid pos: %d, numComp: %d\n",
                arr[mid], mid, *numComps);
        if (arr[mid] == x)    return mid;
        if (arr[mid] > x)    return binarySearch(arr, l, mid-1, x, numComps);
        return binarySearch(arr, mid+1, r, x, numComps);
    }
    return -1;
}
```

Debugging a Program: Step 3 – Finding Error

Now Produces the Following Result:

mid val: 9, mid pos: 4, numComp: 1

mid val: 15, mid pos: 7, numComp: 3

mid val: 11, mid pos: 5, numComp: 5

Result: 5, numComps: 0

Binary Search values looks good

Pos:	0	1	2	3	4	5	6	7	8	9
Values:	1	3	5	7	9	11	13	15	17	19

numComps values are wacky!

May need to add more printf() statements!

Debugging a Program: Step 4 – Fix Code

```
int binarySearch(int arr[], int l, int h, int x, int* numComps) {
    if (r >= l) {
        int mid = l + (h - l)/2;
        (*numComps)++;
        printf ("mid val: %d, mid pos: %d, numComp: %d\n",
                arr[mid], mid, *numComps);
        if (arr[mid] == x)    return mid;
        if (arr[mid] > x)    return binarySearch(arr, l, mid-1, x, numComps);
        return binarySearch(arr, mid+1, r, x, numComps);
    }
    return -1;
}
```

Debugging a Program: Step 5 – Verify the Fix

Now Produces the Following Result:

mid val: 9, mid pos: 4, numComp: 1

mid val: 15, mid pos: 7, numComp: 2

mid val: 11, mid pos: 5, numComp: 3

Result: 5, numComps: 3

Pos:	0	1	2	3	4	5	6	7	8	9
Values:	1	3	5	7	9	11	13	15	17	19

Should test function with a few more target values

Debugging a Program: Step 6 – Remove Code

Remove added printf() statement

Hopefully you don't remove something needed by mistake

```
int binarySearch(int arr[], int l, int r, int x, int* numComps) {  
    if (r >= l) {  
        int mid = l + (r - l)/2;  
        (*numComps)++;  
        if (arr[mid] == x)    return mid;  
        if (arr[mid] > x)    return binarySearch(arr, l, mid-1, x, numComps);  
        return binarySearch(arr, mid+1, r, x, numComps);  
    }  
    return -1;  
}
```


Debugging a Program: Step 7 – Re-Verify

Still Produces the Following Result:

mid val: 9, mid pos: 4, numComp: 1

mid val: 15, mid pos: 7, numComp: 2

mid val: 11, mid pos: 5, numComp: 3

Result: 5, numComps: 3

Pos:	0	1	2	3	4	5	6	7	8	9
Values:	1	3	5	7	9	11	13	15	17	19

Create a Debug Mode for your Program

- Remove the final 2 steps from the Debugging Process
- Add code that will cause Debugging Printf() Statements to execute only when command line flag is given
- If flag is never given in “production mode”, Debugging Printf() Statements don’t execute
- “production staff” never knows it exists

Create a Debug Mode for your Program

- Add code that will cause Debugging Printf() Statements to execute only when command line flag is given
 - ./a.out -d
- If flag is never given in “production mode”, Debugging Printf() Statements don’t execute
 - ./a.out
- “production staff” never knows it exists

Code to be added in “Global Code Space”

```
#include <sdtio.h>
```

```
#define TRUE 1
```

```
#define FALSE 0
```

```
int DebugMode;
```

- Allows DebugMode variable to be used throughout the program

Code to be added in main()

```
int main( int argc, char** argv)
{
    DebugMode = FALSE;
    int i;
    for ( i = 0 ; i < argc ; i++ )
        if ( strcmp (argv[i], "-d") == 0 )
            DebugMode = TRUE;
```

Code to be added in for each debug printf()

```
if ( DebugMode == TRUE )  
    printf ("mid val: %d, mid pos: %d, numComp: %d\n",  
           arr[mid], mid, *numComps);
```

This code can be added into any function without changing any parameters since DebugMode is globally defined.

Using a Global Variable?

- Wait a minute?? Aren't Global Variables BAD???
 - Yes, they are. Use of Global Variables in an APP should be avoided.
- The DebugMode variable is not an “application variable”
- The DebugMode variable is a “development variable”
 - It is being used to help the development process
- This technicality gives us enough reason to “violate” the principle that wants us to always avoid Global Variables
 - Don't have the “clutter” functions with an extra parameter
 - Save the parameters for the “application variables”