Advanced Software Techniques

Common C Programming Errors

In no particular order ...

```
FILE *fp = fopen (filename, "r");
fgets (buffer, sizeof (buffer), fp);
fclose (fp);

char *p = (char *)malloc (1000);
strcpy (p, "hello");
```

#1: Neglecting to check for returned error codes

```
char *p = NULL, *q = NULL;
char buffer[1000] = "";
printf ("Enter some text: ");
fgets (buffer, 1000, stdin);
p = strstr (buffer, "foobar");
q = strstr (p, "barfoo");
```

#1: Neglecting to check for returned error codes

```
int array[100] = {0};
int offset = 0;
char buffer[100] = "";
printf ("Enter the offset to initialize to zero (0 to 99): ");
fgets (buffer, 100, stdin);
offset = atoi (buffer);
array[offset] = 0;
```

#2: Array index checking

```
int x = 5;
int y = 3;
if (x == 10);
{
        y = 0;
}
printf ("y: %d\n", y);
```

#3: Misplaced semicolons

```
int x = 0;
while (x < 10);
{
        printf ("x: %d\n", x);
        x++;
}
```

#3: Misplaced semicolons

```
for (x = 0; x < 10; x++);
{
        printf ("x: %d\n", x);
}
```

#3: Misplaced semicolons

```
int x = atoi (buffer);
if (x = 0)
{
        // some code
}
```

#4: Assignment vs. Comparison

```
int x = atoi (buffer);
if (0 = x)    // better way
{
        // some code
}
```

#4: Assignment vs. Comparison

```
int x = 0;
int y = 0;
int z = 0;
x = 5;
/* setting x to 5
y = 6;
/* setting y to 6 */
z = x + y;
printf ("z: %d\n", z);
```

#5: Missing end of comment

```
switch (value)
{
case 0:
			// do some work
			break;
case 1:
			// do some work but accidentally fall-through to next case
case 2:
			// do some work
			break;
case 3:
			// do some work
			/* INTENTIONAL FALL-THROUGH TO NEXT CASE */
default:
			// do default processing
			break;
}
```

#6: Accidental fall-through on switch

```
int foo (int value)
{
if (value == 5)
                        return 1;
}
```

#7: Not All Codes Paths Return a Value

```
c:\tmp\t.cpp(8) : warning C4715: 'foo' : not all control paths return a value
```

#7a: Ignoring compiler warnings!

It is **expected** that all code you submit in all courses should compile without warnings!

```
warning C4996: 'sscanf': This function or variable may be unsafe. Consider using sscanf_s instead.
```

```
#pragma warning(disable:4996)
```

“Microsoftisms”

```
char str[]   = "foobar";
char str2[] = "foobar";
if (str == str2 )
                    printf ("found foobar!\n");
else
                    printf ("didn't find foobar!\n");
```

#8: String/pointer comparisons

```
char *foo2 (int x)
{
        char buffer[100] = "";
        sprintf (buffer, "%08lx", x);
        return buffer;
}
```

#9: Returning Variables Out-of-Scope

```
void main(void)
{
// code goes here
}
```

#10: Returning nothing from main()

```
int x = 123456;
char name[14] = "fred";
int y = 567890;

strcat(name, " flintstone");
```

#11: Overflowing strings

#12: Inadvertently-escaped characters

```
char filename[81] = "c:\test.txt";
```

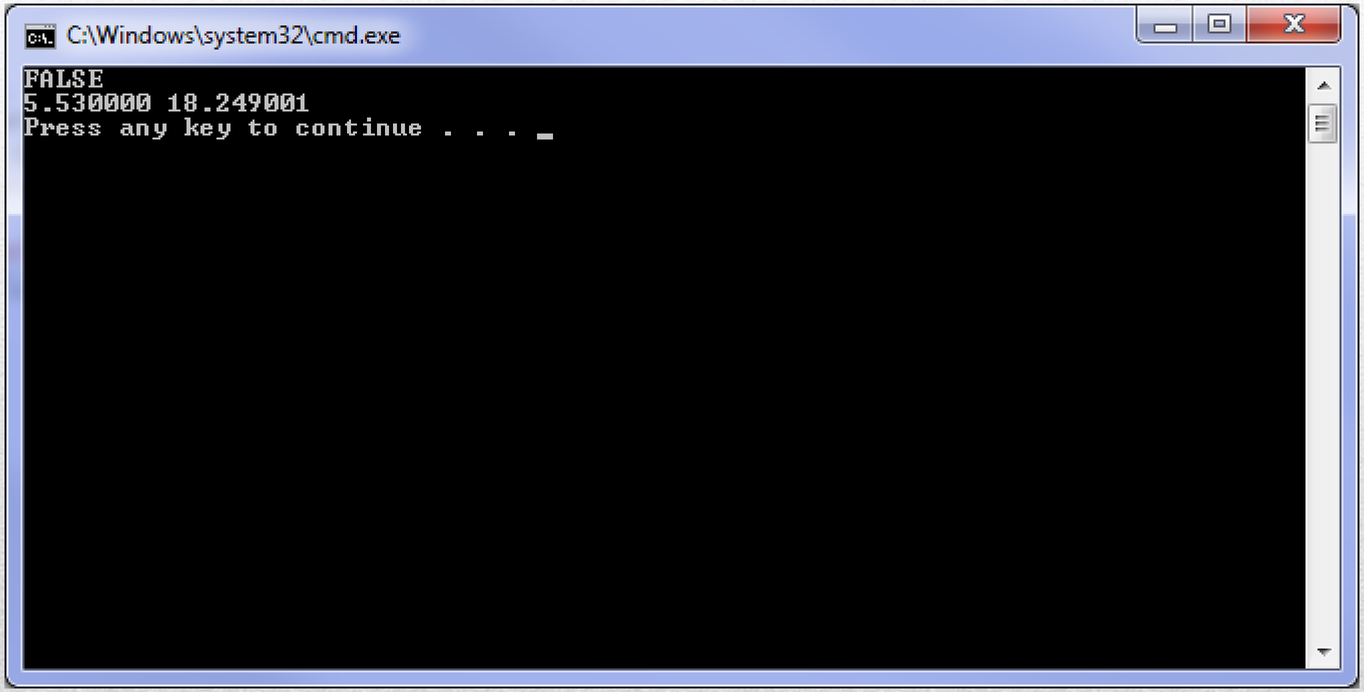
Need two backslashes or this gets interpreted as a tab (\t)

```
if( isdigit(var) == TRUE)
```

#13: Comparing to TRUE

```
float d = 5.53f;
float e = 18.249f;
if( e == (d * 3.3) ) // should be TRUE
		printf("TRUE\n");
else
		printf("FALSE\n");
printf("%f %f\n", d, e);
```

#14: Floating-point Equality


C:\Windows\system32\cmd.exe

```
FALSE
5.530000 18.249001
Press any key to continue . . .
```

#15: Order Side-Effects

```
int a = b * ++b;
```

results depends on compiler

a[b] = ++b;

results depends on compiler

#15: Order Side-Effects

```
int values[] = {004,
                007,
                011, // == 9
                210 // == 210
                };
```

#16: Accidental Octals

```
if( (a == b) & (c == d) )
```

#17: Doubled Operators

```
i = j * m % 10;
          is
i = (j * m) % 10;
          not
i = j * (m % 10);
```

#18: Brackets vs. Precedence

```
i = *pNumber++;
        is
i = *(pNumber++);
```

#18: Brackets vs. Precedence

```
if( i = func() || j = func2() )
           is
if( i = (func() || j) = func() )
```

but this won't compile

#18: Brackets vs. Precedence

```
if( (i = func()) && (j = func2()) )
        might not assign j if func() returns 0
```

Compiler dependent

#19: Trusting "if" Completion – Short Circuit

```
// #include <math.h>

float f = (float) atof(“14.3”);
/* compiler assumes int returned! */
```

#20: Not using #include

```
char *p = malloc(10);
...
free(p);
...
*p = 100;
```

#21: Using freed memory

- http://www.drpaulcarter.com/cs/common-c-errors.php
- http://www.andromeda.com/people/ddyer/topten.html

For more information …