A290/A590 – Meeting 2 Guide
C – The Basics

Goal: We want to get some of the real basic things out "on the table." Most of you know something about all of these, but we still need to be sure we all understand them in more or less the same way.

I. Data Types.

Hoover mentions 4 basic data types: **int, float, double,** and **char**. You may be expecting others to exist, like **bool** or **string**, but C does not recognize these as first-level data types.

What do each of these types allow us to store? It all depends on your system. **int** could be 4 bytes (32 bits), but it could also be 8 bytes (64 bits). **float** will probably be 8 bytes (64 bits), but could only be 4 bytes (32 bits). **double** will be 8 bytes (64 bits) but could be 16 bytes (128 bits). If we add the **long** modifier to **int** or **double,** it is hard to predict what we will get.

There is one way to check, and that is to write some code that will reveal these values for our system.

II. Basic C "Boilerplate" – Creating our first source file.

Let's get a file created that we can use it to test a variety of things. **Be prepared to hand in this file.** Login to your burrow account and let's create a folder for this course.

mkdir A290C

cd A290C

Start emacs. As soon as it opens, save to set your filename: **buffet.c**.

[NOTE: if your emacs always starts with a message instead of a "blank screen," you can get that to stop by doing the following:

1. At the command prompt for silo (not in emacs), type
   ```
   echo '(setq inhibit-startup-message t)' > ~/.emacs
   ```
2. Start emacs and see if this solves the problem. If it does not, you may need to type:
   ```
   Source .emacs
   ```
3. Then try again. If you still see the startup message, please let one of us know.]

First, we need to get a basic comment block at the head of the file. Here is what I will expect at the beginning of every file you submit to me.

```
/*Filename: buffet.c *
*Part of: First A290 program *
*Created by: Jeff Whitmer *
*Created on: 8/27/2025 *
*Last Modified by: Jeff Whitmer *
*Last Modified on: 8/27/2025 *
*/
```

Then get the basics of any C program in place. [Do not include this line. This is part of the guide.]

```
#include <stdio.h>

 int main()

{
      /* The body of your source code will go here */

      return (0);
}
```

You may even want to save the above in another file that you name **template.c** which you could use every time you create a new file.

Now that we have the basics, let's use C to tell us about our system data types and their sizes. Add the following to the body of your **buffet.c** file.

```
        printf("sizeof(char) == %zu\n", sizeof(char));
        printf("sizeof(short) == %zu\n", sizeof(short));
        printf("sizeof(int) == %zu\n", sizeof(int));
        printf("sizeof(long) == %zu\n", sizeof(long));
        printf("sizeof(float) == %zu\n", sizeof(float));
        printf("sizeof(double) == %zu\n", sizeof(double));
        printf("sizeof(long double) == %zu\n", sizeof(long double));
        printf("sizeof(long long) == %zu\n", sizeof(long long));
```

Most of this is designed to give us nice looking output. That's what "%d" and "\n" help us with.

At least some of this "backslash" group of control codes will be important in our early work.

\b backspace
\f form feed
\n newline
\r carriage return
\t horizontal tab
\" double quote
\' single quote
\0 null (string terminator)
\\ backslash
\v vertical tab
\a alert
\? a question mark

In addition, we need to be aware of the "%" control.

%+4d	Forces the sign + or – to be displayed Use below in printf & scanf
% d	Prints a space in place of + sign
%*d	As above using a variable printf("N=% d", Fortnum, age);
%*.*f	As above using 2 variables
%04d	Pad with leading zeros
%4d	Format to 4 chars.
%-4d	Left justify
%5.3f	Format to 5 chars for full num, 3 decimal places cprintf("The average is %5.4f",average);
%-5.3f	As above but LEFT justified
%c	Character
%d	Integer
%f	Floating point / Double
%G	Force a Decimal point
%ld	Long Integer
%p	Hexadecimal
%s	String
%ud	Unsigned Integer
%zu	Print variable of unknown size, i.e., size_t

III. Compiler Basics.

Before we add anything more, let's compile this to see what the output looks like.

For this, it is very simple. We'll learn more about **gcc** later, but right now we can do:

```
gcc buffet.c –o firsttest
```

Remember that the **–o** lets me set a different name for the ***.exe**. If you forget to do this, your output will be in the file **a.out**. This only becomes a problem if you compile different files, as each compilation will replace the previous version of **a.out**.

Type `firsttest` and see what you get for output. Nothing? Try `./firsttest.` Did that work?

IV. Variables, Values and Data Types.

Let's get a better idea of how the different data types can impact how our data is displayed. By extension, choosing the "wrong" data type could result in unusual output as values could be truncated in unexpected ways.

Add the following to your buffet file (from page 36 of Hoover):

```
int   x,y;
char  a;
float f,e;
double d;

x=4;
y=7;
a='H';
f=-3.4;
d=54.123456789;
e=54.123456789;
```

The above should go immediately after "main()" and before our printf statements from above. Notice we are setting a total of 6 variables, of 4 different types.

Now add the following after your other printf statements (again from Hoover):

```
printf("%d %c %f %lf\n",x,a,e,d);
printf("%d %c %.9f %.9lf\n",x,a,e,d);
```

`[Watch for the difference between "1" (one) and "l" (el) on-screen]`

Save your file.

Recompile and let's look at the results.

Compare them to Hoover (page 36) if you have it.

What does this tell us?

V. A Bit of Math [We probably will not get to this until next time].

Let's add some math to our file.

Again from Hoover, with a few changes since everything is in one file.

`int r1,r2,r3,r4,r5;` declares 5 more variables of type int.

For the math, let's put it all at the end of our file. We want to end up with this:

```
r1=x+y;
r2=x-y;
r3=x/y;
r4=x*y;
printf("%d %d %d %d\n",r1,r2,r3,r4);
```

```
r3++;
r4--;
r5=r4%r1;
printf("%d %d %d\n",r3,r4,r5);
```

Is this what you expected? Let's walk through it if we need to.

VI. What's Next?

We'll have a go at some simple programming (functions) next and introduce some additional key concepts including signed versus unsigned types and their uses.

NOTES: