# A290/A590 – Meeting 3 Guide
# C – More of 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. Variables, Values and Data Types. [Picking up from Meeting Guide 2]

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?

## II. A Bit of Math.

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.

## III. Aside: Versions of C.

It is important to at least be aware that there are different versions of C, and they will not all "interpret" your code in precisely the same way. Here is a brief list of the main variations:

C99,
C89 (also called "original" ANSI or ISO C),
"Traditional" C (circa 1978) [also known as "K&R" for Kernighan and Ritchie, the original manual], and
Clean C (intersection of C99 and C++).

One of the consequences of this, and the fact that even the most recent "flavors" of C contain components to make them compatible with legacy code, is that we can write the following:

```
int main(void)
       {
              return(0);
       }

main()
       {
       }

void main()
       {
       }

main(void)
       {
              return 0;
       }

void main()
       {
              return;
       }

int main(void)
       {
              return;
       }
```

and all of these versions of "main" will work.

We will try to stay consistent with Hoover unless or until we come up against problems. The one exception is we will be sure to include `return(0);` at the end of our `main`.

## IV. Signed, Unsigned, and Precision.

We've seen the various sizes, in bytes, of the various data types. However, for any that involve numbers, like **int, float,** and **double**, we need to also be aware of the "unsigned" option and its impact.

By default, all of these are signed, meaning they can represent from some maximum negative number through 0 to some maximum positive number. For example:

int: -2,147,483,648 -> +2,147,483,647 (if 32 bits)

Unsigned allows only positive values, so

uint: 0 -> +4,294,967,295 (if 32 bits)

## V. Simple "Programs."

Let's try one of Hoover's basic functions in order to examine some additional features of C. He goes through a series of development steps in his example in Section 1.4 (pp. 31-35).

Here's the final version of his code as published on pg. 34. Hoover names it `squares3.c`. We want to adopt it (or a version of it) into our buffet.c file.

```
#include <stdio.h>

int main(void)

{

int i,j,number;

printf("Enter a number: ");
scanf("%d",&number);
i=1;
while (i*i <= number)
  {
  j=1;
  while (j < i)
    {
    if (i*i + j*j == number)
      printf("Found: %d + %d\n",i*i,j*j);
    j++;
    }
  i=i+1;
  }
  return(0);
}
```

**BEFORE** we add anything to our buffet.c, can we figure out what this will do? Why are there two "while" loops?

## VI. More updates/additions to buffet.c.

Let's add the 3 variables to the group at the top of our file.

```
int i,j,number;
```

Let's add the rest of the code for **square3.c** at the end of our file

```
printf("Enter a number: ");
scanf("%d",&number);
i=1;
while (i*i <= number)
  {
  j=1;
  while (j < i)
```

```
    {
    if (i*i + j*j == number)
      printf("Found: %d + %d\n",i*i,j*j);
    j++;
    }
  i=i+1;
  }
```

Let’s save and compile. Now run some tests.

What is the output if our number is not the sum of squares?

## VII. Loops.

So we used several while loops. What other kinds are there? How are they different?

## VIII. Conditionals.

We also used a conditional. What is it? What are the options?

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

## IX. What’s Next.

First, we will work more on scanf then on to conditionals and loops as well as arrays and strings (over the next several weeks).

**NOTES:**

**[continued on next page.]**

Here is Hoover's code for his first 3 while loop examples on pages 31-35.

**Squares1**

```
      /* This program tries to identify the largest integer whose
      ** square is less than the given input number. It doesn't quite
      ** work, and needs fixing before proceeding to the next part
      ** of the programming problem. (pg 32) */

#include <stdio.h>

int main(void)

{
int i,number;

printf("Enter a number: ");
scanf("%d",&number);
i=1;

while (i*i < number)
  i=i+1;

printf("%d is the largest square within %d\n",i*i,number);

return(0);
}
```

**Squares2a**

```
      /* This program tries to identify the largest integer whose
      ** square is less than the given input number. It fixes some
      ** of the problem from the first version, but still isn't
      ** quite right. (pg 33) */

#include <stdio.h>

int main(void)

{
int i,number;

printf("Enter a number: ");
scanf("%d",&number);
i=1;

while (i*i < number)
  i=i+1;
i--;

printf("%d is the largest square within %d\n",i*i,number);

  return(0);
}
```

**Squares2b**

```
      /* This program tries to identify the largest integer whose
      ** square is less than the given input number. It works. (pg 34) */

#include <stdio.h>

int main(void)

int i,number;

printf("Enter a number: ");
scanf("%d",&number);
i=1;

while (i*i <= number)
  i=i+1;
i--;

printf("%d is the largest square within %d\n",i*i,number);

  return(0);
}
```