Today’s Goal: Review and Continue examples of very simple C programs.
Make sure you develop an understanding of what you are doing, not just typing codes in!

Review of key C language elements from Lab 1.

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

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

- **headers which include standard input and output; math operations: +/-*/

```
double x,y;
int j,k;
```

- **discussion of base 2

```
printf("\n Enter x\n");
```

- **\n for new line

```
scanf("%lf",&x)
scanf("%i",&j)
```

- **must tell scanf the type of variable being read in

```
do
{
} while(j>0);
```

- **Another type of loop. Execute commands between { and } as long

```
for (j=0;j<20;j=j+2)
{
}
```

- **don’t put a semicolon after for: (j=0;j<20;j=j+2); is very bad!**
if (x==y)
{
}
** Execute commands between { and } if statement is true.
** == is ‘logical equals’
** != is ‘logical not equals’
** && is ‘logical and’
** || is ‘logical or’

if (x<y)
{
}
else
{
}
** A variation on the most simple ‘if’ statement.

[0B] Review of steps in creating a program

Use editor (notepad,...) to type in program
Give the file a useful name, eg add.c
Compile the code (deal with any errors): gcc add.c
Default name of executable is add.exe
Rename the executable: ren a.exe add.e
Or name it while compiling: gcc add.c -o add.e

Pros and cons of an integrated development environment (IDE) like “Visual Studio”:
Built in compiler; initializes some code elements automatically; color
Can be (very) slow!

[1] How fast are computers?
Typical CPU is 3 GHz (gigahertz) = 3 x 10^9 sec^{-1}. Roughly speaking this means you can
do 3 x 10^9 arithmetical operations (addition, subtraction, multiplication, division ... ) each second. When you write more complicated codes, it is a very good habit to estimate the
number of operations needed to run the code so you can make a rough guess at the execution
time. Obviously this has not been an issue for us so far, since our codes have been doing just
a handful of operations.

Type in this code which solves the quadratic equation:

```c
#include <stdio.h>
#include <math.h>
int main()
{
    double a,b,c,root1,root2;
    printf(" Please enter a, b, and c \n");
    scanf("%lf %lf %lf",&a,&b,&c);
    root1 = (-b + sqrt(b*b-4.*a*c) ) / (2.*a);
    root2 = (-b - sqrt(b*b-4.*a*c) ) / (2.*a);
    printf("\n First root is %lf \n",root1);
    printf("\n Second root is %lf \n",root2);
    return 0;
}
```

You will need to compile with gcc geom.c -lm
The -lm links your code to the math libraries which includes sqrt, exp, cos, log, · · ·.  (The header <math.h> only tells the computer about the four elementary math operations: addition, subtraction, multiplication, division.)
What’s ‘bad’ about this code?
Why ‘(2.*a)’ and not ‘2.*a’?

Type in this improved code to solve the quadratic equation:

```c
#include <stdio.h>
#include <math.h>
int main()
{
    double a,b,c,root1,root2;
    printf(" Please enter a, b, and c \n");
    scanf("%lf %lf %lf",&a,&b,&c);
    if (b*b-4.*a*c>0)
    {
        root1 = (-b + sqrt(b*b-4.*a*c) ) / (2.*a);
        root2 = (-b - sqrt(b*b-4.*a*c) ) / (2.*a);
        printf("\n First root is %lf ",root1);
        printf("\n Second root is %lf ",root2);
    }
    else
    {
        printf("\n Discriminant is negative! No roots!\n");
    }
    printf("\n ");
    return 0;
}
```
Type in a code to sum a geometric series:

```c
#include <stdio.h>
#include <math.h>
int main()
{
    double a, sum;
    int j, N;
    printf("Enter a \n");
    scanf("%lf", &a);
    printf("Enter N \n");
    scanf("%i", &N);
    printf(" j sum ");
    sum = 0.0;
    for (j = 0; j < N; j++)
    {
        sum = sum + pow(a, j);
        printf(" %i %12.6lf 
", j, sum);
    }
    return 0;
}
```

If you compile with

gcc geom.c

something goes wrong. Can you fix it? Hint, see the instructions for [3].

Type in a code to sum an arithmetic series:

```c
#include <stdio.h>
#include <math.h>
int main()
{
    int sum = 0;
    int j, N;
    printf("Enter N");
    printf("\n");
    scanf("%i", &N);
    for (j = 0; j < N + 1; j++)
    {
        sum = sum + j;
    }
    printf("the sum is %30i \n", sum);
    return 0;
}
```
Write a code for the Taylor’s series for the exponential:

```c
#include <stdio.h>
#include <math.h>
int main(void)
{
    int j,N;
    long int fact;
    double x,sum;
    printf("Enter N \n");
    printf("\n");
    scanf("%i",&N);
    printf("Enter x\n");
    scanf("%lf",&x);
    sum=1.;
    fact=1;
    for (j=1; j<N; j=j+1)
    {
        fact=fact*j;
        sum=sum+pow(x,j)/fact;
        printf("\n  %i  %lf",j,sum);
    }
    printf("\n");
    return 0;
}
```

Run your code for \( x = 0.6 \) and \( N = 10 \). Compare to the value you get for \( e^{0.6} \) using a calculator. Run your code for \( x = 2.4 \) and \( N = 10 \). Compare to the value you get for \( e^{2.4} \) using a calculator. Run your code for \( x = 5.7 \) and \( N = 10 \). Compare to the value you get for \( e^{5.7} \) using a calculator. Think about what’s going on and why.
[PS1-3] Modify the program in [4] to deal with all three possible values of the discriminant. Write a short paragraph describing geometrically what those three cases correspond to. That is, how is the parabola oriented with respect to the x and y axes in the three different cases? (Drawing a picture is actually best!)

[PS1-4] Run your geometric series code for $a = 0.3, N = 10$ and for $a = 0.8, N = 10$. Write a paragraph which derives the correct answer, and which gives the outputs of your code for both cases. Does your code give the right answer? If not, explain what’s going wrong.

[PS1-5] Run your arithmetic series code for $N = 10$. Write a paragraph which derives the correct answer, and which gives the output of your code.
Run your arithmetic series code for $N = 60000$. Is your output correct?
Run your arithmetic series code for $N = 65535$. Is your output correct?
Run your arithmetic series code for $N = 65536$. Is your output correct?
Figure out what’s special about the number 65536 and explain why your code breaks.

For those of you with previous coding experience, try these problems:

[1] Write a code to read in the slopes $m_1$ and $m_2$ and intercepts $b_1$ and $b_2$ of two lines. Find their point of intersection. As in the quadratic equation code, there are some special cases you need to consider. What do they correspond to geometrically?

[2a] Find the root (solution) of $f(x) = e^x - x - 5 = 0$ by the ‘bisection’ method. That is, read in two points $a$ and $b$ (with $a < b$) which bracket the root. Compute the value of $f$ at the midpoint $c = (a + b)/2$. If $f(c)$ has the same sign as $f(a)$ replace $a$ by $c$. If $f(c)$ has the same sign as $f(b)$ replace $b$ by $c$. At each step the distance between $a$ and $b$ (between which the root lives) decreases by a factor of 2. Continue this process to the desired accuracy.

[2b] Find the root (solution) of $f(x) = e^x - x - 5 = 0$ by Newton’s method (from your calculus course).

Which method is better, bisection or Newton?