GLG 490 Numerical Methods
Intro to C Programming – part 1

In this course, we will learn the fundamentals of C programming. Entire courses can be taught on the subject, so we will restrict ourselves to the basics and learn enough to perform most scientific computations. For those that wish to learn more or have a text which may further explain the fundamentals, the following text is one which I recommend:

Title: C
Author: Klaus Schroder
Publisher: Addison-Wesley
ISBN: 0-201-75878-4

The easiest way to learn programming is by example, and that is how we will proceed in this course. Many examples will be provided in these course notes, and the parts that are to be typed are marked as bold.

Compiling your first code

In the simplest manner, a code involves a source file and an executable file. We will see that we may also have include files as well, but we'll worry about that later. The source file (often called source code) contains the text that lists your programming instructions. It is conventional to give these files the suffix .c (e.g. *mycode.c*). The compiler is a program that converts your programming text to the binary language that the computer can understand. This binary file is called the executable. Once your code is compiled and the executable file is created, you run your code by running the executable.

There are many compilers on the market today, but most scientists use one created by the freeware GNU project. In this class we will exclusively use the GNU compiler which is often called the gcc compiler. One thing to note is that different compilers may be picky with the style of code that you write. We will write completely in standard C language, so all compilers should be able to understand our codes, however, you may notice that different compilers may produce different warning messages. For example the gcc compiler on kaibab may (and likely will) give different warnings than a compiler on your own Linux machine.

Just a note about warnings and errors. A warning means that the compiler is pointing out to you that there may be a problem with your code, but it isn't certain. In fact, we will probably find many warnings in this course, but most of them are harmless. With a warning, the compiler will complete the compilation of the code and produce an executable file. An error, on the other hand, means that the compiler found something major wrong with your code, and the compilation ends, and the executable file is not created.

First lets write the "skeleton" of every C code. Every code that you write will start with these lines. Make a file named *mycode.c* and type the following:

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

int main(void)
{




return 0;
}
```

In C, all of the code that you write is contained within the main function named “main”.
Now, lets instruct the code to do something. Add the following line in the main function:

fprintf(stdout,”Hello World\n”);

Now, lets compile the code. First exit the text editor. On the Linux command line type:

gcc mycode.c

If it worked successfully, you should see the created executable file named *a.out*

Run the executable by typing **a.out**

We can name our executable something other than *a.out* by compiling it this way.

gcc –o mycode.x mycode.c

Try this. Note, *mycode.x* can be replaced with any name that you would like, but its conventional to use the same name as the source code, but with a .x instead of a .c

Working with numerical data types (integers, floats, and doubles), variables, and operators

Type out the following code in the file *tutorial1.c*. Afterwards, the instructor will go through and explain line by line:

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

int main(void)
{

/* this is what a comment looks like */

/* declarations */
int number_of_apples, number_of_oranges;
int number_of_people;
int total_number_of_fruit;

float apples_per_person;
float oranges_per_person;

double number1, number2;
double my_result;

/* end of declarations */

number_of_apples=20;
number_of_oranges=15;
number_of_people=12;

total_number_of_fruit=number_of_apples+number_of_oranges;

apples_per_person=number_of_apples/number_of_people;
oranges_per_person=(number_of_oranges*1.0)/(number_of_people*1.0);

fprintf(stdout,"There are %d pieces of fruit total\n\n", total_number_of_fruit);
fprintf(stdout,"There are %f apples per person\n", apples_per_person);
fprintf(stdout,"There are %f oranges per person\n\n", oranges_per_person);

number1=27.1;
number2=21.2;

my_result=number1+number2;
fprintf(stdout,"The sum of the numbers is %f\n",my_result);
```

```
my_result=number1-number2;
fprintf(stdout,"Number 1 is bigger than number 2 by %f\n",my_result);

my_result=number1*number2;
fprintf(stdout,"The product of numbers is %f\n",my_result);

my_result=number1/number2;
fprintf(stdout,"Number 1 divided by number 2 is %f\n",my_result);



return 0;
}
```

Compile and run your code. The output should look like this:

```
There are 35 pieces of fruit total

There are 1.000000 apples per person
There are 1.250000 oranges per person

The sum of the numbers is 48.300000
Number 1 is bigger than number 2 by 5.900000
The product of numbers is 574.520000
Number 1 divided by number 2 is 1.278302
```

Whats wrong with the number of apples per person? This is an example of the pitfalls of integer division, something to be very careful about. See how we rectified the problem when calculating the number of oranges. You must always do this if you don't want to make this mistake!

Now, go through a fix your code to correctly determine the number of apples per person. Comment out the old part and modify accordingly:

```
/*
apples_per_person=number_of_apples/number_of_people;
*/
apples_per_person=(1.0*number_of_apples)/(1.0*number_of_people);
```

Words not to use as variables

auto, break, case, char, const, continue, default, do, double, else, enum, extern, float, for, goto, if, int, long, register, return, short, signed, sizeof, static, struct, switch, typedef, union, unsigned, void, volatile.

Conditionals, Looping, and if

Write the following code as *tutorial2.c*

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

/* This code was written by me to examine why integer division may be useful */

int main(void)
{

int number_of_required_vans;
int number_of_students;
int number_of_instructors=3;
float number_of_instructors_per_student;
int number_of_seated_people;
int max_students=30;
int people_per_van=6;
int total_number_of_people;
int modulus;

for (number_of_students=1; number_of_students<=max_students;number_of_students++)
{

        /* examine how integer division is used to calculate this */

        total_number_of_people=number_of_students+number_of_instructors;

        number_of_required_vans=total_number_of_people/people_per_van;

         /* using the modulus, % */

        if (total_number_of_people%people_per_van > 0 )
        {
              number_of_required_vans++;
        }
```

```
        number_of_instructors_per_student=(number_of_instructors*1.0)/(number_of_students*1.0
        );

        fprintf(stdout,"students: %d  vans: %d  instructors_per_student: %f\n",
number_of_students, number_of_required_vans,  number_of_instructors_per_student);

}


return 0;
}
```

Conditionals

>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
==	Equal to
!=	Not equal to
&&	AND
||	OR

More on conditionals

Type the following code, and name the file *tutorial3.c*

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

int main(void)
{

int number;

        for (number=0; number<=20;number=number+2)
        {

                fprintf(stdout,"\nnumber: %d\n", number);

                if (number<5)
                {
                    fprintf(stdout,"This number is less than 5\n");
```

```
        }
        else if ((number >=5) && (number<=10))
        {
            fprintf(stdout,"This number is between 4 and 11\n");
        }
        else
        {
            fprintf(stdout,"The number is greater than 10\n");
        }

        if ((number==4)||(number==6))
        {
            fprintf(stdout,"The number is either 4 or 6\n");
        }

        if (number!=8)
        {
            fprintf(stdout,"The number is not 8\n");
        }
        else
        {
            fprintf(stdout,"Wow, this number must be 8\n");
        }

    }



return 0;
}
```