EME152 Discussion 1

September 29, 2021

Contact Information

Nicolas Ventura (he/him) “Nic”

M.S. student in Mechanical & Aerospace Engineering

nfventura@ucdavis.edu

EE (energy efficiency) student assistant at NERSC - I write a lot of code!

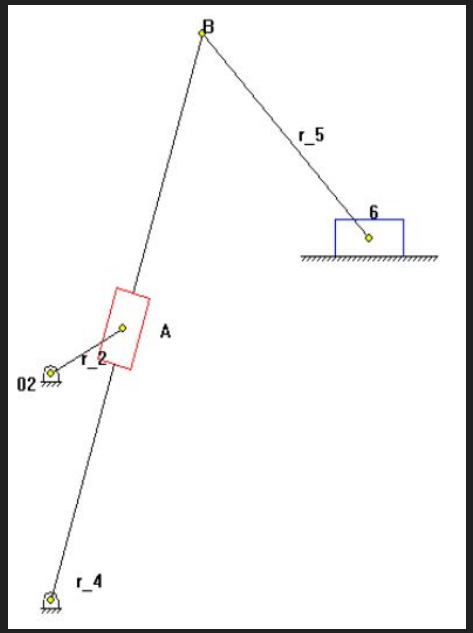
Class Discord

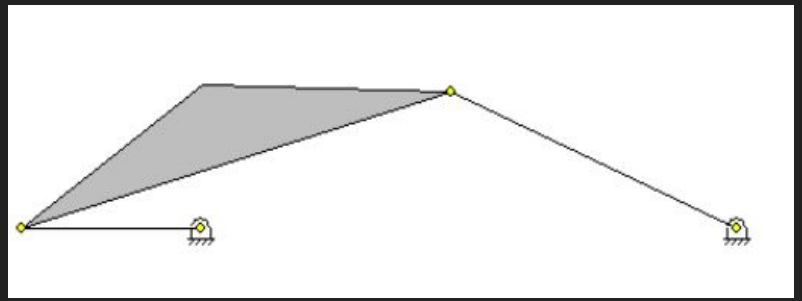
“UCD Engineering” > EME > #eme152

https://discord.gg/nbUBmHn6n8

About this class

It’s a coding heavy class! You will be using the mechanism toolkit in Ch, invented by Harry Cheng, to develop simulations and analyses on various linkages and mechanisms. Ch encompasses most **C** functions and some **C++** functions. Ch code is executed in **ChIDE** or the Ch command prompt.





Agenda

- Go over fundamentals of C programming language
  - C data types
  - printf
  - scanf
  - The math.h function library
  - Conditionals
  - Loops
    - Nested loops
- By the end of this discussion, you should be able to write some basic C programs

Discussion 1

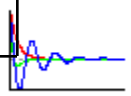
- Types
- printf() specifiers
- Representing mathematical formulas in C
- Conditional statements
- Loops
- Nested loops
- “break” and “continue” statements

Types

- Variables in C must be “declared” before they may be used.
- Each variable in C is a certain “type”. Different types of variables are useful for different things, or can hold a different amount of information.

printf() specifiers

Argument Type	Format-Control-String
binary number	“%b” in Ch only
char	“%c”
short	“%hd”
unsigned short	“%uhd”
int	“%d”
unsigned int	“%u”
long long	“%lld”
unsigned long long	“%ulld”
float	“%f”
double	“%lf”
string	“%s”
pointer	“%p”



The precision and field width for printf() specifiers

- Additional information may be provided to the specifiers to further refine output.
- %.10lf ← This tells printf() to print a double with 10 digits after the decimal point.
- %20.5lf ← this tells printf() to print a double that takes up to 20 character spaces. The left side of the number is padded with spaces to make up the difference.

printf() specifiers

```
> printf("%20lf\n", 5.4)
           5.400000

> printf("%5.10lf\n", 5.4)
5.4000000000

> printf("%.10lf\n", 5.4);
5.4000000000

> printf("%15.10lf\n", 5.4);
   5.4000000000

> printf("%8.2lf\n", 5.4);
    5.40
```

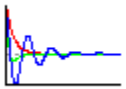
Mathematical Expressions

- Mathematical expressions in C are similar to normal mathematical expressions. Some things to keep in mind:
- The '^' character does not mean “to the power of” in C. Use the function **pow**() instead.
- Use parentheses to explicitly define the precedence of a mathematical expression.

Mathematical Formulae

$$\sin(x) + \frac{5x^2 + x}{2x}$$

```
#include <math.h> // for sin()
double result, x;

// May initialize x to something here.
result = sin(x) + (5*x*x + x) / (2*x);
```



Conditional Statements

- A conditional statement may be implemented in a couple methods in C. The most basic and commonly used method is the “if” statement. Some pseudocode example usage follows:

Conditional Statements

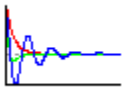
```
if(expr1) {
    do_something();
    may_do_multiple_things_in_each_block();
} else if (expr2) {
    do_something_else();
} else {
    do_default_action();
}

/* Note that the "if else" and "else" blocks are optional.
   The following is perfectly valid */
if(expr) {
    do_something;
}
```

Loops

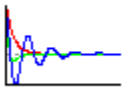
- Computers excel at performing the same or similar task many times. A loop is simply a piece of code that may be executed many times.

Loops

- Create a program that calculates the n'th prime number.



```
/* File: nth_prime_nested.c */
#include <stdio.h>
#include <math.h>
int main() {
  int num_primes = 0;
  int i = 1, is_prime, n;
  printf("Which prime number do you wish to find? : ");
  scanf("%d", &n);
  printf("Calculating...\n");
  for(i = 1; num_primes < n; i++) {
    /* For each iteration, check whether i is prime.
       If it is, increment
     * our variable keeping track of the number of primes */
    is_prime = 1;
    for(j = 2; j < (sqrt(i) + 1); j++) {
      if ( (i % j) == 0) { // If this is zero,
                           // then the number cannot be prime
        is_prime = 0;
      }
    }
    if( is_prime ) {
      num_primes++;
    }
  }
  i--;
  printf("The prime is %d\n", i);
  return 0;
}
```



Loops

- Example Output:

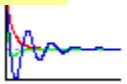
```
dko@boxzor:~/School/eme5/fall08/disc3$ ch ./nth_prime.c
Which prime number do you wish to find? : 100
Calculating...
The prime is 541
dko@boxzor:~/School/eme5/fall08/disc3$ ch ./nth_prime.c
Which prime number do you wish to find? : 1000
Calculating...
The prime is 7919
dko@boxzor:~/School/eme5/fall08/disc3$
```

Nested Loops

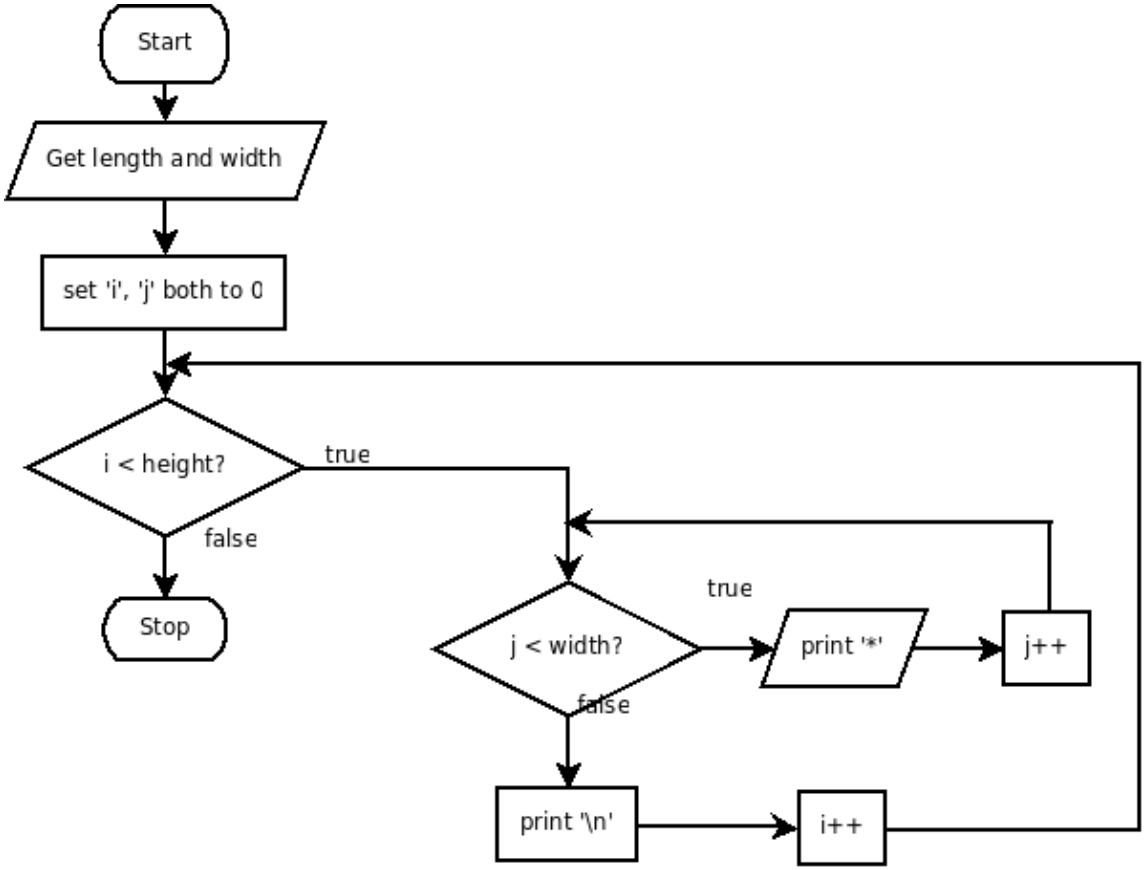
- Write a program to draw a rectangle of '*' characters. The length and width of the rectangle must be specified by the user.

Nested Loops

```
/* File: box.c */
/* This program prints a box with side lengths specified by the user. */
#include <stdio.h>
int main()
{
  int width, height;
  int i,j;

  printf("Please enter the width of the box: ");
  scanf("%d", &width);
  printf("Please enter the height of the box: ");
  scanf("%d", &height);

  for(i = 0; i < height; i++) {
    for(j = 0; j < width; j++) {
      printf("*");
    }
    printf("\n");
  }
  return 0;
}
```



Nested Loops



“break” and “continue”

- Both “break” and “continue” statements are used within loops.
- The statements are used to cause the program to “jump” to another position.
- The “break;” statement causes the program to jump out of the enclosing loop.
- The “continue;” statement causes the program to jump to the next iteration of the loop

“break” and “continue”

```
1:  #include <stdio.h>
2:
3:  int main()
4:  {
5:    int i;
6:    for(i = 0; i<30; i++) {
7:      if(i == 3) {
8:        continue;
9:      } else if (i == 5) {
10:       break;
11:     } else {
12:       printf("%d ", i);
13:     }
14:   }
15:   printf("\n");
16:   return 0;
17: }
```

A Mathematical Formula with a Single Variable

Calculate function

sinc(x) = sin(x)/x

from -10 <= x <= 10 with a step size 5.

sinc(0) = sin(0)/0 is NaN inside a program. But, it can be proved that sin(0)/0 is 1 in calculus.

Generate data points (x, y) for x in the range x0 <= x <= xf with step size xstep linearly. The number of points

```
n = (xf - x0)/xstep + 1;
```

Each data point can be calculated by

```
for(i = 0; i <n; i++) {
    x = x0 + i*xstep;
    y = f(x);
}
```

```
/* File: forsinc.c */
#include <stdio.h> /* for printf() */
#include <math.h>  /* for sin() and fabs() */
#include <float.h> /* for FLT_EPSILON */

int main() {
    double x, x0, xf, xstep, result;
    int i, n;

    printf("    x      sinc(x)\n");
    printf("  ---------------\n");
    x0 = -10.0;                  /* initial value */
    xf = 10.0;                   /* final value */
    xstep = 5.0;                 /* step size */
    n  =  (xf - x0)/xstep + 1;   /* num of points */
    for(i = 0; i < n; i++) {
       x = x0 + i*xstep;         /* value x */
       if(fabs(x) < FLT_EPSILON)
          result = 1.0;
       else
          result = sin(x)/x;
       printf("%8.4f %8.4f\n", x, result);
    }
    return 0;
}
```

Output:

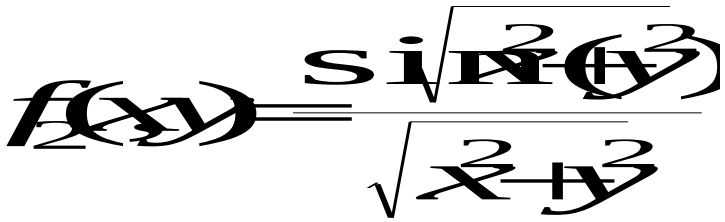
```
  x        sinc(x)
 ---------------
-10.0000  -0.0544
 -5.0000  -0.1918
  0.0000   1.0000
  5.0000  -0.1918
 10.0000  -0.0544
```

Note:

- `sin(x)` can be calculated using a function declared in header file **math.h.**
- Use a loop control variable of int type.
- Use format specifier `"%8.4"` with field width 8 and 4 digits after the decimal point.

A Mathematical Formula with Multiple Variables

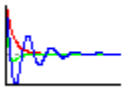
Calculate function



for x from -10 <= x <= 10 with a step size 10 and

y from -10 <= y <= 10 with a step size of 10.

```
/* File: forsinr.c */
#include <stdio.h>
#include <math.h>

int main() {
    double x, x0, xf, xstep,
           y, y0, yf, ystep, result;
    int i, j, nx, ny;

    printf("      x           y    sinr(x,y)\n");
    printf("  ----------------------------\n");
    x0 = -10.0;
    xf = 10.0;
    xstep = 10.0;
    nx  =  (xf - x0)/xstep + 1; /* num of points for x */
    y0 = -10.0;
    yf = 10.0;
    ystep = 10.0;
    ny  =  (yf - y0)/ystep + 1; /* num of points for y */
```



```
    for(i = 0; i < nx; i++) {
       x = x0 + i*xstep;          /* caclulate value for x */
       for(j = 0; j <ny; j++) {
          y = y0 + j*ystep;       /* calculate value for y */
          r = sqrt(x*x + y*y);
          result = sin(r)/r;
          printf("%10.4f %10.4f %8.4f\n", x, y, result);
       }
    }
    return 0;
}
```

Output:

sin(0)/0 is NaN

```
         x          y        sinr(x,y)
-----------------------------------
  -10.0000   -10.0000    0.0707
  -10.0000     0.0000   -0.0544
  -10.0000    10.0000    0.0707
    0.0000   -10.0000   -0.0544
    0.0000     0.0000       NaN
    0.0000    10.0000   -0.0544
   10.0000   -10.0000    0.0707
   10.0000     0.0000   -0.0544
   10.0000    10.0000    0.0707
```

Note:

- Square root function sqrt() is declared in header file **math.h**.
- Use nested loops for x and y. Each has its own loop control variable of int type.

Calculating the Square Root of Number Using Newton's Method

Based on Newton's method for finding a root of an equation, the square root $x = \sqrt{a}$ can be calculated by the formula

$$x_{i+1} = \frac{1}{2}\left(x_i + \frac{a}{x_i}\right)$$

where x_{i+1} is a function of the previous term x_i. It first starts with an initial guess x_0 for a root, then calculates successive approximate roots $x_1, x_2, \ldots, x_i, x_{i+1}, \ldots$ A *convergence criterion* for numerical computation is a condition to terminate an iteration. Calculate the square root $\sqrt{3}$ with the initial guess $x_0 = a$ and the convergence criterion $|x_{i+1} - x_i| <$ `FLT_EPSILON`. The program also stops if the number of iteration is larger than 100.

Calculating square root of 3.

```
/* File: sqrtx.c
   Calculate square root sqrt(a) for a = 3.0 using Newton's method */
#include <stdio.h>
#include <math.h>  /* for fabs() */
#include <float.h> /* for FLT_EPSILON */

#define N 100      /* the maximum number of iteration */

int main() {
   int i;
   double a, x0, x1, x2;

   a = 3.0;        /* sqrt(a) with a = 0.3 */
   x0 = a;         /* an initial guess for x0 */

   x1 = x0;                    /* set x1 to x0 */
   for(i = 1; i <= N; i++) {
      x2 = (x1+a/x1)/2.0;      /* Newton's recursive formula */
      if(fabs(x2-x1) < FLT_EPSILON)
        break;
      x1 = x2;     /* update value x1 for next iteration */
   }
   if(i < N) {     /* number of iteration is less than N */
     printf("sqrtx(%.2f) = %f\n", a, x2);
     printf("sqrt(%.2f)  = %f\n", a, sqrt(a));
     printf("Number of iterations = %d\n", i);
   }
   else {          /* number of iteration equals N */
     printf("sqrtx failed to converge\n");
   }
   return 0;
}
```

Output:

```
sqrtx(3.00) = 1.732051
sqrt(3.00)  = 1.732051
Number of iterations = 5
```

The output indicates that
only 5 iterations are needed to converge