

ENGG1003 - Friday Week 3

More Sequence Examples
Maybe More Flow Control

Brenton Schulz

University of Newcastle

March 6, 2020

Assessment Task Rules

...Jump to rules PDF

Easy(ish) Assessment Task Example

Write a C program which generates a sequence of numbers:

$$x_1, x_2, x_3, \dots$$

with the iterative equation:

$$x_n = 3x_{n-1} + 2x_{n-2}$$

and initial conditions:

$$x_1 = 3, x_2 = 1$$

The program should exit after printing (x_8 or an $x_n > 100$).

Easy(ish) Assessment Task Example

The program's output format is:

```
n x<newline>
```

For the values given, the output is:

```
1 3.000000
2 1.000000
3 9.000000
4 29.000000
5 105.000000
```

Easy(ish) Assessment Task Example

- ▶ What do we need to do?
 - ▶ Set up variables
 - ▶ Give some initial values
 - ▶ Implement the equation
 - ▶ Print the initial values
 - ▶ Write a `while()` loop
 - ▶ Get the exit condition correct
 - ▶ Print results
 - ▶ Wrap the whole thing in `main()`

Set up variables

Question didn't specify, but lets assume float

```
1 float xn, xnm1, xnm2;  
2 int n;
```

Give some initial values

Question gave us:

$$x_1 = 3, x_2 = 1$$

Be careful with `xnm1` and `xnm2`, where are we starting?

```
1 float xn, xnm1 = 1, xnm2 = 3;  
2 int n = 3; // The first unknown is x for n=3
```

Implement the equation

$$x_n = 3x_{n-1} + 2x_{n-2}$$

```
1 float xn, xnm1 = 1, xnm2 = 3;  
2 int n = 3; // The first unknown is x for n=3  
3  
4 xn = 3.0*xnm1 + 2*xnm2;
```

That calculates x_3 , but how does the program “advance in time”?

Implement the equation

Shift all the variables “forward in time” with:

```
1 float xn, xnm1 = 1, xnm2 = 3;
2 int n = 3; // The first unknown is x for n=3
3
4 xn = 3.0*xnm1 + 2*xnm2;
5 xnm2 = xnm1;
6 xnm1 = xn;
```

Print the initial values

```
1 float xn, xnm1 = 1, xnm2 = 3;
2 int n = 3; // The first unknown is x for n=3
3
4 // x1 and x2 given so just hard code n
5 printf("1 %f\n", xnm2);
6 printf("2 %f\n", xnm1);
7
8 xn = 3.0*xnm1 + 2*xnm2;
9 xnm2 = xnm1;
10 xnm1 = xn;
```

Write a `while()` loop

We need to calculate x_n more than once, so:

```
1 float xn, xnm1 = 1, xnm2 = 3;
2 int n = 3; // The first unknown is x for n=3
3
4 // x1 and x2 given so just hard code n
5 printf("1 %f\n", xnm2);
6 printf("2 %f\n", xnm1);
7
8 while( /* something */ ) {
9     xn = 3.0*xnm1 + 2*xnm2;
10    xnm2 = xnm1;
11    xnm1 = xn;
12 }
```

Get the exit condition correct

The value of n goes from 1 to 8, and x_n must remain below 100:

```
1 float xn, xnm1 = 1, xnm2 = 3;
2 int n = 3; // The first unknown is x for n=3
3 // x1 and x2 given so just hard code n
4 printf("1 %f\n", xnm2);
5 printf("2 %f\n", xnm1);
6 while( (n <= 8) && (xn < 100) ) {
7     xn = 3.0*xnm1 + 2*xnm2;
8     xnm2 = xnm1;
9     xnm1 = xn;
10    n++;
11 }
```

Print results

```
1 float xn, xnm1 = 1, xnm2 = 3;
2 int n = 3; // The first unknown is x for n=3
3 // x1 and x2 given so just hard code n
4 printf("1 %f\n", xnm2);
5 printf("2 %f\n", xnm1);
6 while( (n <= 8) && (xn < 100) ) {
7     xn = 3.0*xnm1 + 2*xnm2;
8     xnm2 = xnm1;
9     xnm1 = xn;
10    n++;
11    printf("%d %f\n", n, xn);
12 }
```

Wrap the whole thing in `main()`

NB: This code still has errors. Search for and debug live.

```
1 #include <stdio.h>
2 main() {
3     float xn, xnm1 = 1, xnm2 = 3;
4     int n = 3; // The first unknown is x for n=3
5     // x1 and x2 given so just hard code n
6     printf("1 %f\n", xnm2);
7     printf("2 %f\n", xnm1);
8     while( (n <= 8) && (xn < 100) ) {
9         xn = 3.0*xnm1 + 2*xnm2;
10        xnm2 = xnm1;
11        xnm1 = xn;
12        n++;
13        printf("%d %f\n", n, xn);
14    }
```



Is the solution optimal?

- ▶ Some marks are allocated to reducing variable count
- ▶ It tests your understanding of how the = operation works
- ▶ Lets look at the maths:

```
1 xn = 3.0*xnm1 + 2*xnm2;  
2 xnm2 = xnm1;  
3 xnm1 = xn;  
4 n++;  
5 printf("%d %f\n", n, xn);
```

- ▶ Do we need *all* those variables?

- ▶ In this case: yes

```
1 xn = 3.0*xnm1 + 2*xnm2;  
2 xnm2 = xnm1;  
3 xnm1 = xn;  
4 n++;  
5 printf("%d %f\n", n, xn);
```

- ▶ We can't overwrite xnm1 before shifting it into xnm2
- ▶ Result must be stored in xn first

Another Isolated Example

- ▶ What if the equation was:

$$x_n = 0.2x_{n-1}$$

- ▶ This will *work*:

```
1 xn = 0.2*xnm1;  
2 xnm1 = xn;
```

- ▶ But because we never need `xnm1` elsewhere this is more optimal:

```
1 xn = 0.2*xn;
```

- ▶ Marks (above a pass) may be allocated to variable optimisation

Hard Assessment Task Example

Write a C program which generates two sequences of numbers:

$$x_0, x_1, x_2, \dots$$

$$y_0, y_1, y_2, \dots$$

with the coupled iterative equations:

$$x_n = 0.6x_{n-1} + 0.2y_{n-1}$$

$$y_n = 0.1x_{n-1} + 0.9y_{n-1}$$

and initial conditions:

$$x_0 = 5$$

$$y_0 = 0$$

Hard Assessment Task Example

$$x_n = 0.6x_{n-1} + 0.2y_{n-1}$$

$$y_n = 0.1x_{n-1} + 0.9y_{n-1}$$

- ▶ Lets have an attempt at implementing the equations
- ▶ We need *at least* two variables:
 - ▶ `float xn`
 - ▶ `float yn`
- ▶ Lets also use two “previous” variables:
 - ▶ `float xnm1`
 - ▶ `float ynm1`

Hard Assessment Task Example

$$x_n = 0.6x_{n-1} + 0.2y_{n-1}$$

$$y_n = 0.1x_{n-1} + 0.9y_{n-1}$$

- Our calculation code can then be:

```
1 xn = 0.6*xnm1 + 0.2*ynm1;  
2 yn = 0.1*xnm1 + 0.9*ynm1;  
3 xnm1 = xn;  
4 ynm1 = yn;
```

- **Question:** Do we need all these variables?

Hard Assessment Task Example

$$x_n = 0.6x_{n-1} + 0.2y_{n-1}$$

$$y_n = 0.1x_{n-1} + 0.9y_{n-1}$$

► **Counter-question:** What is wrong with this?

```
1 xn = 0.6*xn + 0.2*yn;  
2 yn = 0.1*xn + 0.9*yn;
```

Hard Assessment Task Example

$$x_n = 0.6x_{n-1} + 0.2y_{n-1}$$

$$y_n = 0.1x_{n-1} + 0.9y_{n-1}$$

- ▶ **Counter-question:** What is wrong with this?

```
1 xn = 0.6*xn + 0.2*yn;  
2 yn = 0.1*xn + 0.9*yn;
```

- ▶ Why doesn't mathematics convert into code?

Hard Assessment Task Example

- ▶ Mathematics is *instant*

Hard Assessment Task Example

- ▶ Mathematics is *instant*
- ▶ Code is evaluated line by line

Hard Assessment Task Example

- ▶ Mathematics is *instant*
- ▶ Code is evaluated line by line
- ▶ Variables can *change* between lines, resulting in the wrong equation being implemented
- ▶ The previous slide was *actually* doing:

$$\begin{array}{ll} x_n = 0.6x_{n-1} + 0.2y_{n-1} & x_n = 0.6*x_n + 0.2*y_n; \\ y_n = 0.1x_n + 0.9y_{n-1} & y_n = 0.1*x_n + 0.9*y_n; \end{array}$$

Hard Assessment Task Example

- Observe the correct subscripts:

$$x_n = 0.6x_{n-1} + 0.2y_{n-1}$$

$$y_n = 0.1x_{n-1} + 0.9y_{n-1}$$

- In the 2nd equation we need x_{n-1} but the first equation would destroy that value
- We *must* use an extra variable to store x_{n-1} for y_n to be calculated correctly

Hard Assessment Task Example

- ▶ Aside: You may see coupled equations vaguely like this in signals and systems theory

$$x_n = 0.6x_{n-1} + 0.2y_{n-1}$$

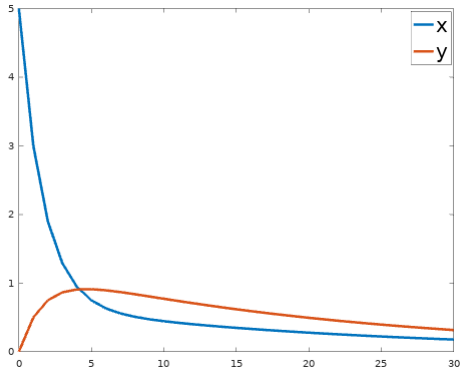
$$y_n = 0.1x_{n-1} + 0.9y_{n-1}$$

- ▶ Lets write C code with the minimum variables:

```
xtmp = x; // store xn before we lose it
x = 0.6*x + 0.2*y; // Original xn value lost
y = 0.1*xtmp + 0.9*y; // stored xn used, yn
```

- ▶ ...And implement

Results



This graph was generated by copying the output data into a file then loading it in MATLAB.