Newton’s Method Program

Setup main function

Declare variables

Newton’s Method
- Set initial guess
- for loop
  - Calculate new guess
    - need derivative
  - Exit?

Print results
Save each guess?

Setup main function

Declare variables

Newton’s Method
Set initial guess
for loop
Calculate new guess
need derivative
Exit?

Print results
Save each guess?

```
#include…

int main( void )
   {

   int i…
   float x1,…

   x1 =…

   for( i=0; i<20; i++ )
      {

      x2 = x1 - …

      if( fabs(x2-x1) < 1e-6 ) break;

      x1 = x2;
      }

   printf( “Results are…” );

   return 0;
   }
```

C functions

A function is a piece of a program that can be re-used
- Modular programming concept – break large task into smaller parts
- Program sections that can be re-used
- Typically, you <u>pass</u> them some data to work on and they <u>return</u> some result

Basic function structure:

Name of function

```
float myfunction( float x )
       {
       float y;
       y = x * x;
       return y;
       }
```

This function takes a number and just squares that number.

Type of variable to be returned

Type of variable being received

```
#include <stdheaders.h>

float square( float );

int main( void )
{
        float x, y;

        printf( "Enter a number to square : " );
        scanf( "%f", &x );

        y = square( x );

        printf( "The square of %f is %f\n", x, y );

        return 0;
}

float square( float a )
{
        float b;

        b = a * a;

        return b;
}
```

Must declare functions – called a function prototype

Set y to the returned value

Call the function, passing it the variable to work on (square)

Function receives a value, stored in the variable a

Square a, store as b

End function and return b

Using C functions with the Newton's program

New guess is calculated from the general Newton's formula:

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$$

Your program can either:

1) write in the actual function, here, or

2) use C functions and put the actual function in its own place in the program.

```
x2 = x1 – ( x1 * x1 - 5 ) / ( 2 * x1 )
```

```
x2 = x1 – f(x1) / fderiv(x1)
```

```
float f( float x )
{
    return x * x – 5;
}
```

```
float fderiv( float x )
{
    float h = 0.0001;
    return (f(x+h) – f(x-h)) / (2*h);
}
```

C functions and "visibility"

The variable b is "invisible" from the main function.

The variable x is "invisible" from the square function.

x had to be passed to the square function.

Any variables needed by a function must be passed to it.

```
#include <stdheaders.h>

float square( float );

int main( void )
{
        float x, y;

        printf( "Enter a number to square : " );
        scanf( "%f", &x );

        y = square( x );

        printf( "The square of %f is %f\n", x, y );

        return 0;
}

float square( float a )
{
        float b;

        b = a * a;

        return b;
}
```

This time, pass <u>two</u> variables to the function.

```
#include <stdheaders.h>

float add( float, float );

int main( void )
{
        float x1, x2, y;

        printf( "Enter two numbers to add : " );
        scanf( "%f %f", &x1, &x2 );

        y = add( x1, x2 );

        printf( "The sum is %f\n", y );

        return 0;
}

float add( float a1, float a2 )
{
        float b;

        b = a1 + a2;

        return b;
}
```

Return the sum of those two numbers.

Can only return <u>one</u> value from a function.

One (poor?) way around the visibility issue – make variables <u>global</u>

A variable declared within a function is <u>local</u> to that function. A variable declared outside of any function is <u>global</u>.

```
#include <stdheaders.h>

void square( void );
float x1, x2, ya1, ya2;

int main( void )
{
        printf( "Enter two numbers to square : " );
        scanf( "%f %f", &x1, &x2 );

        square();

        printf( "The square of %f is %f\n", x1, ya1 );
        printf( "The square of %f is %f\n", x2, ya2 );

        return 0;
}

void square( void )
{
        ya1 = x1 * x1;
        ya2 = x2 * x2;

        return;
}
```

Exit test – the other option

1. When the guesses get closer together than some small value
2. When the “zero” is close enough to zero

```
if( fabs( x2 – x1 ) < 1e-6 ) break;
```

```
if( f(x2) < 1e-6 ) break;
```

Use double instead of float?

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$$

Big number

Small number

Instead of “float”, make every variable a “double”

Newton's solutions for the square root of 5

Floating point variables are type float (~6 digit precision)

Step	Estimate (start at 2.0)	Error (estimate - sqrt(5))
0	2.00000000000000000000	-2.36068010330200000000e-001
1	2.25000143051147000000	1.39334201812744000000e-002
2	2.23611021041870000000	4.22000885009766000000e-005
3	2.23606801033020000000	0.00000000000000000000e+000

Floating point variables are type double (~16 digit precision)

Step	Estimate (start at 2.0)	Error (estimate - sqrt(5))
0	2.00000000000000000000	-2.36067977499790000000e-001
1	2.25000000000002000000	1.39320225002342000000e-002
2	2.23611111111111000000	4.31336113200231000000e-005
3	2.23606797791580000000	4.16014334092552000000e-010
4	2.23606797749979000000	0.00000000000000000000e+000

Floating point variables are type qfloat (~104 digit precision)

Step	Estimate (start at 2.0)	Error (estimate - sqrt(5))
0	2.00000000000000000000	-2.3606797749978969641e-01
1	2.2500000000000000000	1.39320225002103035908e-02
2	2.2361111111111111111	4.31336113214147019374e-05
3	2.23606797791580400276	4.16014306351350830924e-10
4	2.23606797749978969645	3.86991595958341229309e-20
5	2.23606797749978969641	3.34879120065566327223e-40
6	2.23606797749978969641	2.50761663295405193342e-80
7	2.23606797749978969641	0.00000000000000000000e+00

Solution using Linear Interpolation

Floating point variables are type double (~16 digit precision)

Step	Estimate (start at 2.0, 3.0)	Error (estimate - sqrt(5))
1	2.20000000	-3.60679775e-2
2	2.23076923	-5.29874673e-3
3	2.23529412	-7.73859853e-4
4	2.23595506	-1.12921320e-4
5	2.23605150	-1.64753539e-5
6	2.23606557	-2.40372930e-6
7	2.23606763	-3.50699539e-7
8	2.23606793	-5.11663769e-8

Debugger Tutorial