Newton's Method Program

Setup main function

Declare variables

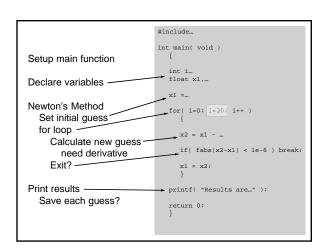
Newton's Method

- Set initial guess
- for loop

Calculate new guess need derivative

Exit?

Print results
Save each guess?



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 pass them some data to work on and they return some result Basic function structure: Name of function float myfunction(float x) This function takes a number and just squares that number. return y; Type of variable Type of variable being received to be returned

```
#include <stdheaders.h>
float square( float );

int main( void )

float x, y;

printf( "Enter a number to square : " );

scanf( "%f", &x );

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

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

return 0;

float square( float a )

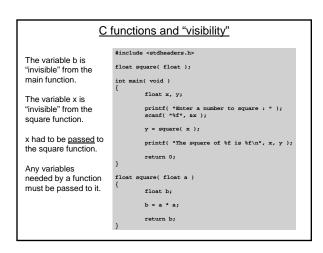
float b;

b = a * a;

return 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. $x_2 = x_1 - (x_1 * x_1 - 5) / (2 * x_1)$ $x_2 = x_1 - f(x_1) / fderiv(x_1)$ float f(float x){ float f(float x){ float f(float x)} { return f(float x)} }



```
#include <stdheaders.h>
                         float add( float, float );
                         int main( void )
                                  float x1, x2, y;
This time, pass \underline{two}
                                  printf( "Enter two numbers to add : " );
scanf( "%f %f", &x1, &x2 );
variables to the
function.
                                  y = add( x1, x2 );
                                  printf( "The sum is %f\n", y );
                         float add( float al, float a2 )
Return the sum of
those two numbers.
                                b = a1 + a2;
Can only return one
                                  return b;
value from a function.
```

One (poor?) way around the visibility issue— make variables global #include <stdheaders.h> void square(void); float xl, x2, yal, ya2; int main(void) { printf("Enter two numbers to square : "); square(); printf("The square of %f is %f\n", xl, yal); printf("The square of %f is %f\n", x2, ya2); return 0; } void square(void) { yal = x1 * xl; ya2 = x2 * x2; return; }

Exit test – the other option	
 When the guesses get closer together than some small value 	
When the "zero" is close enough to zero	
if(fabs(x2 - x1) < 1e-6	6) break;
if(f(x2) < 1e-6) break;	
Use double instead of float?	
$x_2 = x_1 - \underbrace{f(x_1)}_{f'(x_1)}$ 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	2.0000000000000000000000000000000000000	-2.36068010330200000000e-001
1	2.25000143051147000000	1.39334201812744000000e-002
2	2.23611021041870000000	4.22000885009766000000e-005
3	2.23606801033020000000	0.000000000000000000000000000000000000

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

	r locating point variables are type double (* To digit precision)		
	Step	Estimate (start at 2.0)	Error (estimate - sqrt(5))
I	0	2.0000000000000000000000000000000000000	-2.36067977499790000000e-001
I	1	2.25000000000002000000	1.39320225002342000000e-002
	2	2.236111111111111000000	4.31336113200231000000e-005
	3	2.23606797791580000000	4.16014334092552000000e-010
	4	2.23606797749979000000	0.000000000000000000000000000000000000

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

Step	Estimate (start at 2.0)	Error (estimate - sqrt(5))
0	2.000000000000000000000	-2.3606797749978969641e-01
1	2.25000000000000000000	1.39320225002103035908e-02
2	2.23611111111111111111	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	0.00000000000000000000000000e+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	