Homework:

Write a program in C++ to solve (by Newton's method) the transcendental equation: $f(x) = a - x * \tanh(x) = 0$, given a.

Read a and an initial guess for x from command line.
Solve iteratively. At location x, $f(x) != 0$ as you guessed x.
Taylor series:

$$f(x+dx) = f(x) + f'(x)\, dx + f''(x)\, dx^2/2 + \ldots$$

Assume $f(x+dx) = 0$, then, assuming $f''(x)$ term is small,

$dx = -f(x)/f'(x)$, where $dx = x_{new} - x$

$x_{new} = x - f(x)/f'(x)$, then repeat.

When to stop iteration?

Use a C++ function in this program. Try a=1.

Sum 16 numbers

```
/* sum 16 numbers

*/
#include <iostream>
using namespace std;  //without this, need std::cout; std::endl

int main()
 {
   int a[16];   //declare an integer array; NOTE the [ brackets]
 // initialize the array
        a[0] = 12; a[1] = 8; a[2] = 3; a[3] = 5;   //more than 1 assignment statement per line
        a[4]= 17; a[5] = 20; a[6] = 17; a[7] = 2;
        a[8] = 7; a[9] = 2; a[10] = 1; a[11] =14;
        a[12] = 2; a[13] = 4; a[14] = 14; a[15]=21;

        int sum = 0;  //initialize the sum
        for (int i = 0; i<16; i++)
          sum +=a[i];

        cout << "sum = " << sum << endl;

        return 0;
        }
```

Arrays

1-D float a[4];
a[0] = 2.5; //assign values to elements

or int b[] = {4, 3, 2,1}; //Note defined via { }

2-D int c[2][2];
int c[2][2] = { {2, 3}, {3, 4}};

```
#include <iostream>
using namespace std;
// check array
int main()
{ int a[2][2] = {{0, 2}, {0, 3}};  //note commas
  cout << a[0][0] <<endl;
  cout << a[1][1] <<endl;
  }
```

Output:

```
0
3
```

```
/* sum 16 numbers

   */
#include <iostream>
using namespace std;  //without this, need std::cout; std::endl

int main()
 {
   int a[16] = {12, 8, 3, 5, 17, 20, 17, 2, 7, 2, 1, 14, 2,4, 14, 21};

   int sum = 0;  //initialize the sum
   for (int i = 0; i<16; i++)
     sum +=a[i];                //note: 16 additions
   cout << "sum = " << sum << endl;

   return 0;
   }
```

```
/* sum 16 numbers input via command line

   */
#include <iostream>
using namespace std;  //without this, need std::cout; std::endl

int main()
 {
   int a;
   cout << "sum 16 numbers: enter numbers below:” << endl;
   int sum = 0;  //initialize the sum
   for (int i = 0; i<16; i++)
    {
     cout << "number "<< i+1 <<": ";
     cin >> a;
     sum +=a;
     }

   cout << "sum = " << sum << endl;

   return 0;
   }
```

Three Looping Structures

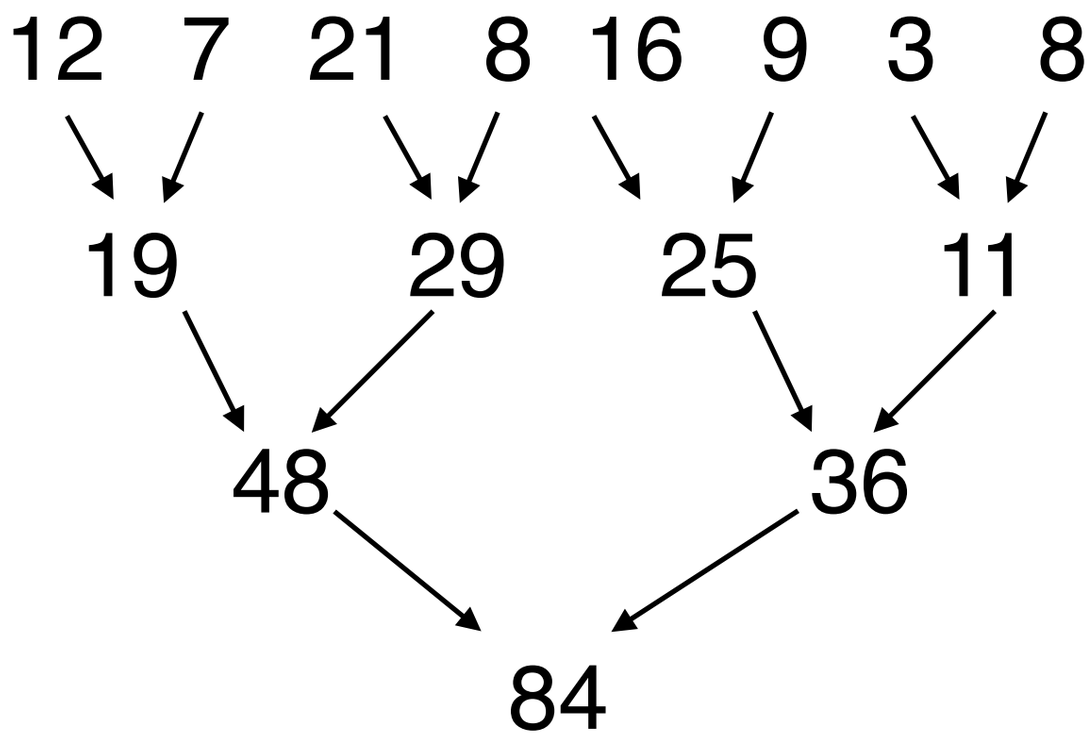
```
for (initialization; condition; increment)
      { statements}

do {statements} while (condition);

while (condition)
      {statements}
```

An example of parallel computing

Summing numbers



Serial sum: 7 steps; Parallel : 3 steps

Summation Algorithm

Assume p processors, 2p numbers

Step 1: 2p numbers -> p numbers

Step 2: p numbers -> $p/2 = p/2^1$

Step 3: $p/2^2$

.....

Step j: $p/2^{(J-1)}$

if, at step j, we have 2 elements left, then $p = 2^j$

or the number of steps $j = \log_2 (p)$

e.g. 2048 numbers in 10 steps

So for numbers n greater than 2p

Assume n/p is an integer for simplicity

If each processor adds n/p numbers, then only p left. These can be done in $\log_2 p$ steps

Total number of steps = $n/p + \log_2 p$

This works for multiply and divide as well

Functions

Functions are used to carry out some task (but conveniently located outside of the main program). They must be declared or defined prior to the main program.

Functions

```
// function example
#include <iostream>
using namespace std;

int addition (int a, int b)                    // this function returns an integer
{
  int r;
  r=a+b;                                       //r is a local variable in function
  return r;
}

int main ()
{
  int z;
  z = addition (5,3);
  cout << "The result is " << z << endl;
  return 0;
}
```

http://www.cplusplus.com/doc/tutorial/functions/

Function prototyping

```
// function example
#include <iostream>
using namespace std;

int addition( int, int);

int main ()
{
  int z;
  z = addition (5,3);
  cout << "The result is " << z;
  return 0;
}

int addition (int a, int b)
{
  int r;
  r=a+b;
  return r;
}
```

Prototype (occurs before main()

Function

Class Program

Write a program to compute factorial of a number input by user.
The factorial should computed in a function.

$$n! = 1*2*3\ldots..(n-1)*n$$

Factorial

```
/*   Simple function call for factorial
     fails for numbers bigger than 12.  Why?
     int is 32 bits, -2^(31)-1 to 2^(31)-1, or 2,147,483,647.
*/

#include<iostream>

using namespace std;

// Simple factorial Function
int factorial(int var)
{
       int fact=1;
       for(int i = 2; i<= var; i++)
       fact = fact * i;
       return fact;
 }

int main()
{
    int num;
    cout<<"input a number: " << endl;
    cin >> num;
    cout<<"Factorial:"<<factorial(num)<<endl;
    return 0;
}
```

Structures

Structure is an object that can hold different types of data

```
struct type_name {
member_type1 member_name1;
member_type2 member_name2;
member_type3 member_name3;
.
.
} object_names;
```

Example:

```
struct phys_params {
float rho;
float gravity;
float viscosity;
uint gamma_coef;
} Params;
```

structure is type phys_params

only one object of type phys_params;
can define more later: phys_params More_params;

Using structure

```
struct phys_params {
float rho;
float gravity;
float viscosity;
uint gamma_coef;
} Params;

Params.gravity = 9.81;
Params.rho = 1000.;

a = Params.gravity;
```

or

```
force = Params.rho*Params.gravity*vol;
```

```
//: C03:SimpleStruct.cpp
struct Structure1 {
  char c;
  int i;
  float f;
  double d;
} s1;          //s1 declared here as a Structure1

int main() {
  struct Structure1 s2;     //s2 declared here as a Structure1
  s1.c = 'a'; // Select an element using a '.'
  s1.i = 1;
  s1.f = 3.14;
  s1.d = 0.00093;
  s2.c = 'a';
  s2.i = 1;
  s2.f = 3.14;
  s2.d = 0.00093;
} ///:~
```

Type Definition

```
typedef char C;


int main()
{
C achar, your_char; // means that achar, your_char are type char
```

Typedef with Structure

```
//: C03:SimpleStruct2.cpp
// Using typedef with struct
typedef struct {
  char c;
  int i;
  float f;
  double d;
} Structure2;

int main() {
  Structure2 s1, s2;
  s1.c = 'a';
  s1.i = 1;
  s1.f = 3.14;
  s1.d = 0.00093;
  s2.c = 'a';
  s2.i = 1;
  s2.f = 3.14;
  s2.d = 0.00093;
} ///:~
```

Pointers

int k;

Integer k is stored in memory at a given address

To get the address of k, use &k (ampersand == address_of)

<u>A pointer is a variable that stores a memory address.</u>

Uses: if you know an address of a variable, then you can get the value that is stored there

If you have a big piece of data, easier to deal with address of data rather than all the elements of the data

Declare a Pointer

Pointer: <variable_type> *<name>;

```
int *p;   // declare p as a pointer that points to int
int k, j;

k=2;
p= &k;  //pointer is now address_of k
j=*p;  //dereference operator, returns value at
          address in p. So j=2
*p=7;  //what is k now?
```

NOTE: pointer is same type as variable

```
int * p;
int k;
++p;    //increment the pointer address by the
          length of an integer
```

An array:

```
int a[] = {1, 2, 3, 4}; //declare 1-D array and assign elements

int *p;
p = &a[0];  p is pointing to first element in array
cout << “a[1] = “ <<*(p+1) << endl;
```

Pointers to Structures

```
//: C03:SimpleStruct3.cpp
// Using pointers to structs
typedef struct Structure3 {
  char c;
  int i;
  float f;
  double d;
} Structure3;

int main() {
  Structure3 s1, s2;
  Structure3* sp = &s1;  //Note sp is pointer to struct
  sp->c = ‘a';           //Use -> with pointer
  sp->i = 1;
  sp->f = 3.14;
  sp->d = 0.00093;
  sp = &s2; // Point to a different struct object
  sp->c = 'a';
  sp->i = 1;
  sp->f = 3.14;
  sp->d = 0.00093;
} ///:~
```

```
/* Modified from Jensen, T. Tutorial on Pointers*/
#include <iostream>
using namespace std;
int my_array[] = {1,23,17,4,-5,100};
int *ptr;
int main(void)
 {
  int i;
  ptr = &my_array[0];

  for (i = 0; i < 6; i++)
  {
      cout<<"my_array["<<i<<"] = "<< my_array[i]<<endl;
      cout<<"ptr+ "<<i<<" = " << *(ptr+i) <<endl;
   }
  return 0;
}
```

Jensen, A Tutorial on Pointers

Importance of pointers

```
// function example
#include <iostream>
using namespace std;

int addition (int a, int b)
{
  int r;
  r=a+b;
  return r;
}

int main ()
{
  int z;
  z = addition (5,3);
  cout << "The result is " << z << endl;
  return 0;
}
```

Example of “Pass-By-Value”

Pass-by-Value

```
// function example
#include <iostream>
using namespace std;

int addition (int a, int b)
{
  int r;
  a=a+2;
  r=a+b;
  return r;
}

int main ()
{
  int z;
  int a=5;
  z = addition (a,3);
  cout << "The result is " << z << endl;
  cout << “  and a = “ << a << endl;
  return 0;
}
```

If we add a line in addition:

a=a+2;

What is the value of **z** printed out as the program executes?

What is the value of **a** printed out?

C++: Pass-By-Reference

Allows modification of **a**

```
// function example
#include <iostream>
using namespace std;

int addition (int  &a, int b)
{
  int r;
  a=a+2;
  r=a+b;
  return r;
}
int main ()
{
  int z;
  int a=5;
  z = addition (a,3);
  cout << "The result is " << z << endl;
  cout << “  and a = “ << a << endl;
  return 0;
}
```

Uses address of a

Pass-by-Pointer (C, C++)

```
// function example
#include <iostream>
using namespace std;

int addition (int  *a, int b)
{
  int r;
  *a=*a+2;
  r=*a+b;
  return r;
}
int main ()
{
  int z;
  int a=5;
  z = addition (&a,3);
  cout << "The result is " << z << endl;
  cout << “  and a = “ << a << endl;
  return 0;
}
```

Passing Array By Pointer

```
/* Pass by pointer an array
 */

 #include <iostream>
 using namespace std;

 int summer (int * a, int N)
  {

    for (int i =0; i< N; i++)
       {a[i] += i*10;}
    return N;
   }

 int main()
  { int a[10]={1,1,1,1,1,1,1,1,1,1};
    summer (a, 10);
    for (int i = 0; i<10; i++)
      cout <<"a[" <<i<<"] = "<<a[i]<<endl;

    return 0;
 }
```

Important for arrays, as passing by value would mean whole array copied into function local variables, but with pass by reference/pointer only the address of the first member of the array is sent to the function

Pass by Reference

```
/* Pass by reference an array
 */

 #include <iostream>
 using namespace std;

 int summer (int a[ ], int N)    \\use [ ] instead of &
  {

    for (int i =0; i< N; i++)
      {a[i] += i*10;}
   return N;
  }

 int main()
  { int a[10]={1,1,1,1,1,1,1,1,1,1};
   summer (a, 10);
   for (int i = 0; i<10; i++)
     cout <<"a[" <<i<<"] = "<<a[i]<<endl;

   return 0;
 }
```

Class problem

Write a function that swaps two numbers:

In the main, assign a=30, b=45.
Call the function
Print out the results: a=45, b=30
Call similar function
Print out the results: a=30, b=45.

Write one function that is pass-by-reference and another that is pass-by-pointer

Class problem

```
#include <iostream>
using namespace std;
void swap1(…)
{
 …
}
void swap2( ….)
{
 …
}
int main()
{
    int a = 30;
    int b = 45;
    cout<<"Before Swap1\n";
    cout<<"a="<<a<<" b="<<b<<"\n";

    swap1(…);
    cout<<"After Swap1 with pass by reference\n";
    cout<<"a="<<a<<" b="<<b<<"\n";

    swap2(…);
    cout<<"After Swap2 with pass by pointer\n";
    cout<<"a="<<a<<" b="<<b<<"\n";
}
```

Note: return of void

```
void swap1(int &x, int &y)
{
    int z = x;
    x=y;
    y=z;
}
void swap2(int *x, int *y)
{
    int z = *x;
    *x=*y;
    *y=z;
}

int main()

    swap1(a,b);

    swap2(&a, &b);
```