October 2023

Introduction to C++: Workshop Three

Dr. Alexander Hill

a.d.hill@liverpool.ac.uk

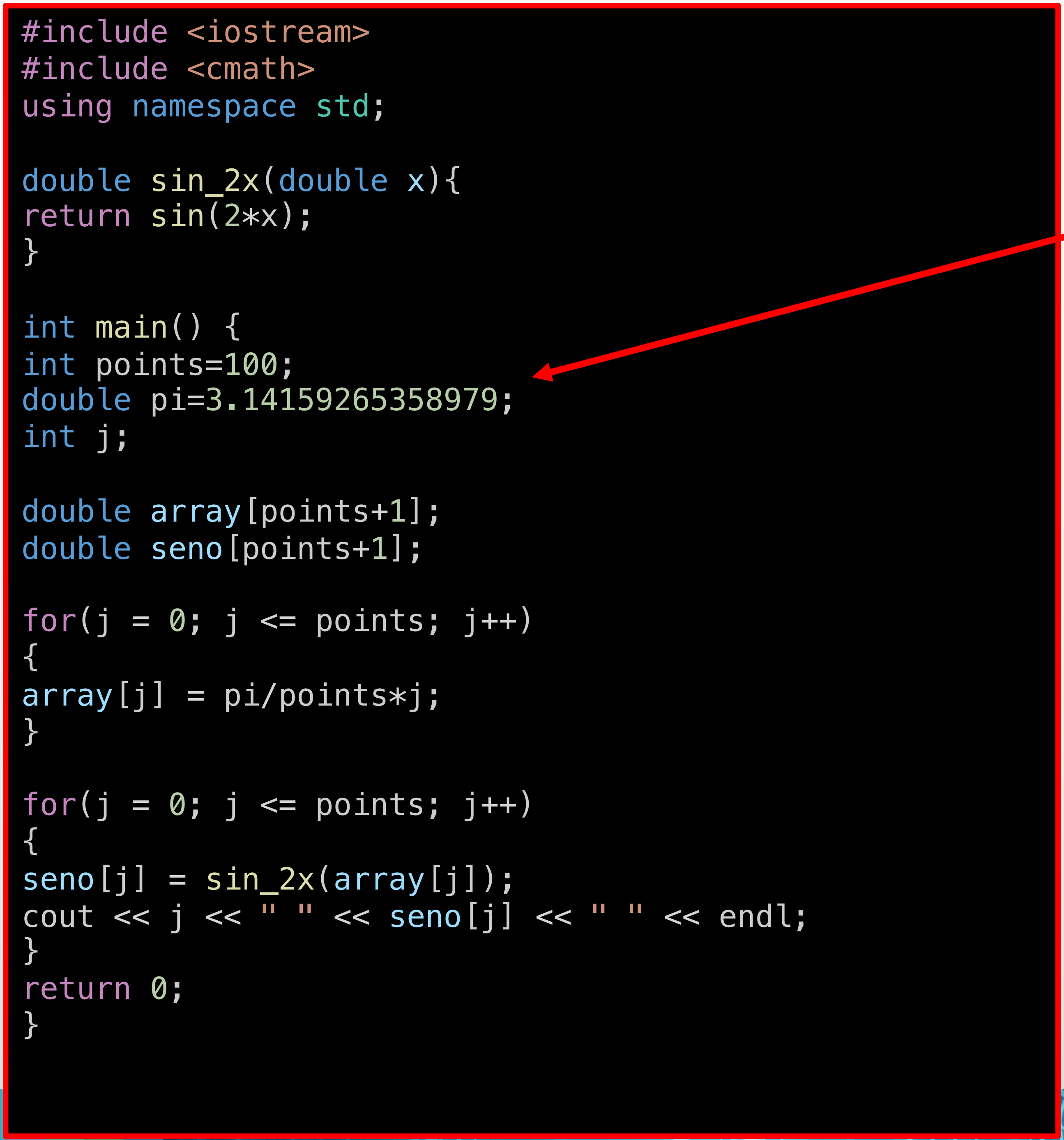
Last Week

- Variables and data types
- Functions
- For loops
- Arrays and vectors

Challenge Four (Homework)

- Create an evenly-space array (or vector) between 0 and π (you'll need to import <cmath>)
- Create a function called sin_2x which returns sin(2x)
- Loop over your array and pass the elements to sin_2x
- Save the results to a new array of the same length
- Send me your scripts by Wednesday evening next week (18/11/23)

LIV.INNO



```
#include <iostream>
#include <cmath>
using namespace std;

double sin_2x(double x){
return sin(2*x);
}

int main() {
int points=100;
double pi=3.14159265358979;
int j;

double array[points+1];
double seno[points+1];

for(j = 0; j <= points; j++)
{
array[j] = pi/points*j;
}

for(j = 0; j <= points; j++)
{
seno[j] = sin_2x(array[j]);
cout << j << " " << seno[j] << " " << endl;
}
return 0;
}
```

Works, but nicer to import it as a constant

Ana

```
$ g++ -o output ana.cpp
(base) alexhill at Alexs-MacBook-Air-2
in
~/Documents/UOL/LIVINNO/Teaching/C++_Wor
kshops/2023/WS3/scripts
$ ./output
0 0
1 0.0627905
2 0.125333
3 0.187381
4 0.24869
5 0.309017
6 0.368125
7 0.425779
8 0.481754
9 0.535827
10 0.587785
11 0.637424
…
```

```cpp
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;
double sin_2x(double x) { //function delcaration;
expects one arg of type double called x
return sin(2 * x); //function definition; uses sin
function from cmath
}
int main() {
const double pi = M_PI; //uses pi constant from cmath
const int numPoints = 9; //number of steps
double step = pi / (numPoints - 1); //calculates step
size
double values[numPoints]; //array to store evenly
spaced points between 0-pi
double results[numPoints]; //new array to store results
of sin_2x
for (int i = 0; i < numPoints; ++i) { //loop
values[i] = i * step; //fills values with points
between 0-pi
results[i] = sin_2x(values[i]); //calculate sin(2x) and
stores it in results
//prints outcome
cout <<"sin(2 * " << values[i] << ") = " << results[i]
<< endl;
}
return 0;}
```

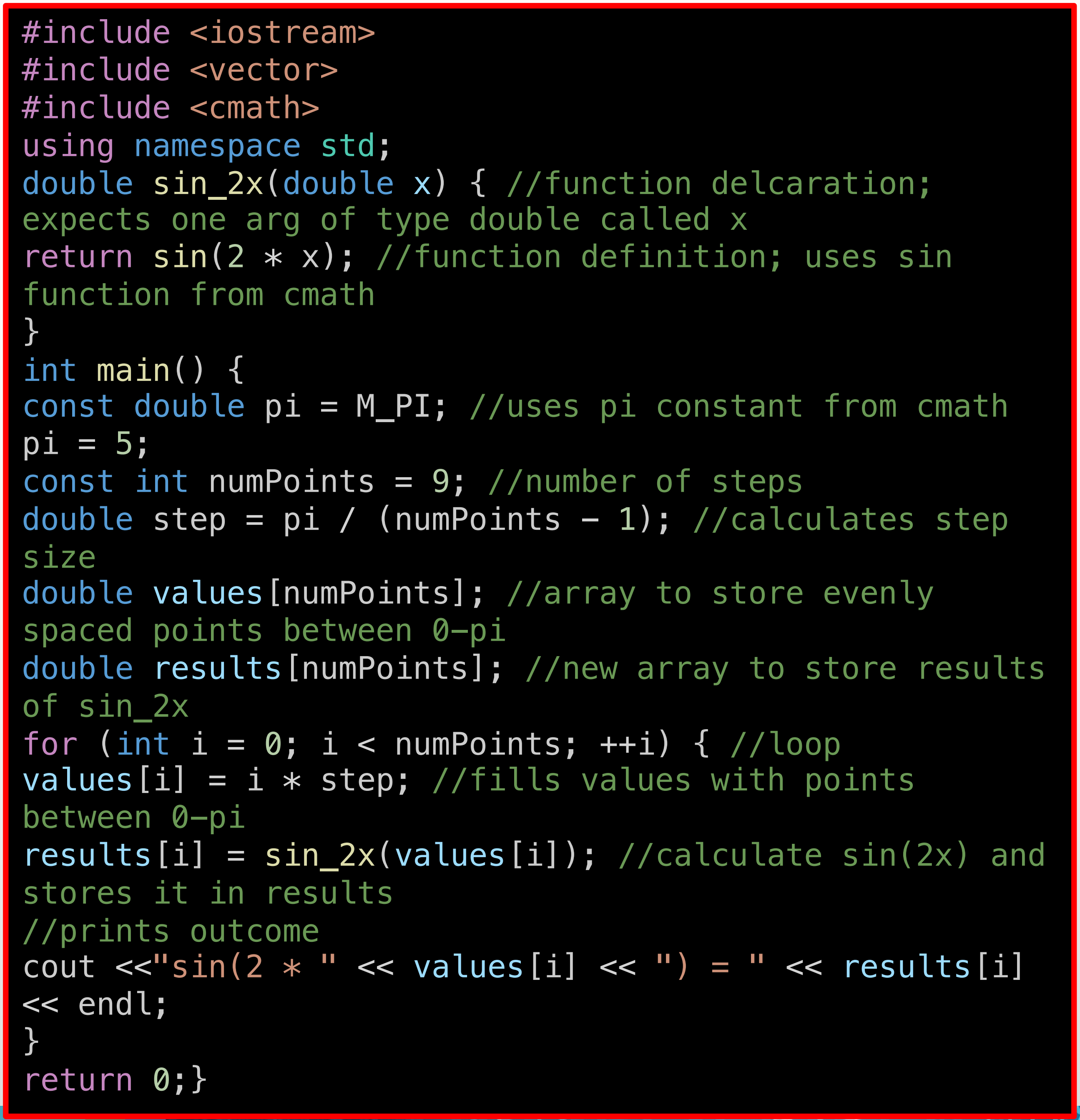
cmath import:
M_PI

const

```
$ g++ -o output emily.cpp
(base) alexhill at Alexs-MacBook-Air-2
in
~/Documents/UOL/LIVINNO/Teaching/C++_Wor
kshops/2023/WS3/scripts
$ ./output
sin(2 * 0) = 0
sin(2 * 0.392699) = 0.707107
sin(2 * 0.785398) = 1
sin(2 * 1.1781) = 0.707107
sin(2 * 1.5708) = 1.22465e-16
sin(2 * 1.9635) = -0.707107
sin(2 * 2.35619) = -1
sin(2 * 2.74889) = -0.707107
sin(2 * 3.14159) = -2.44929e-16
```

Emily

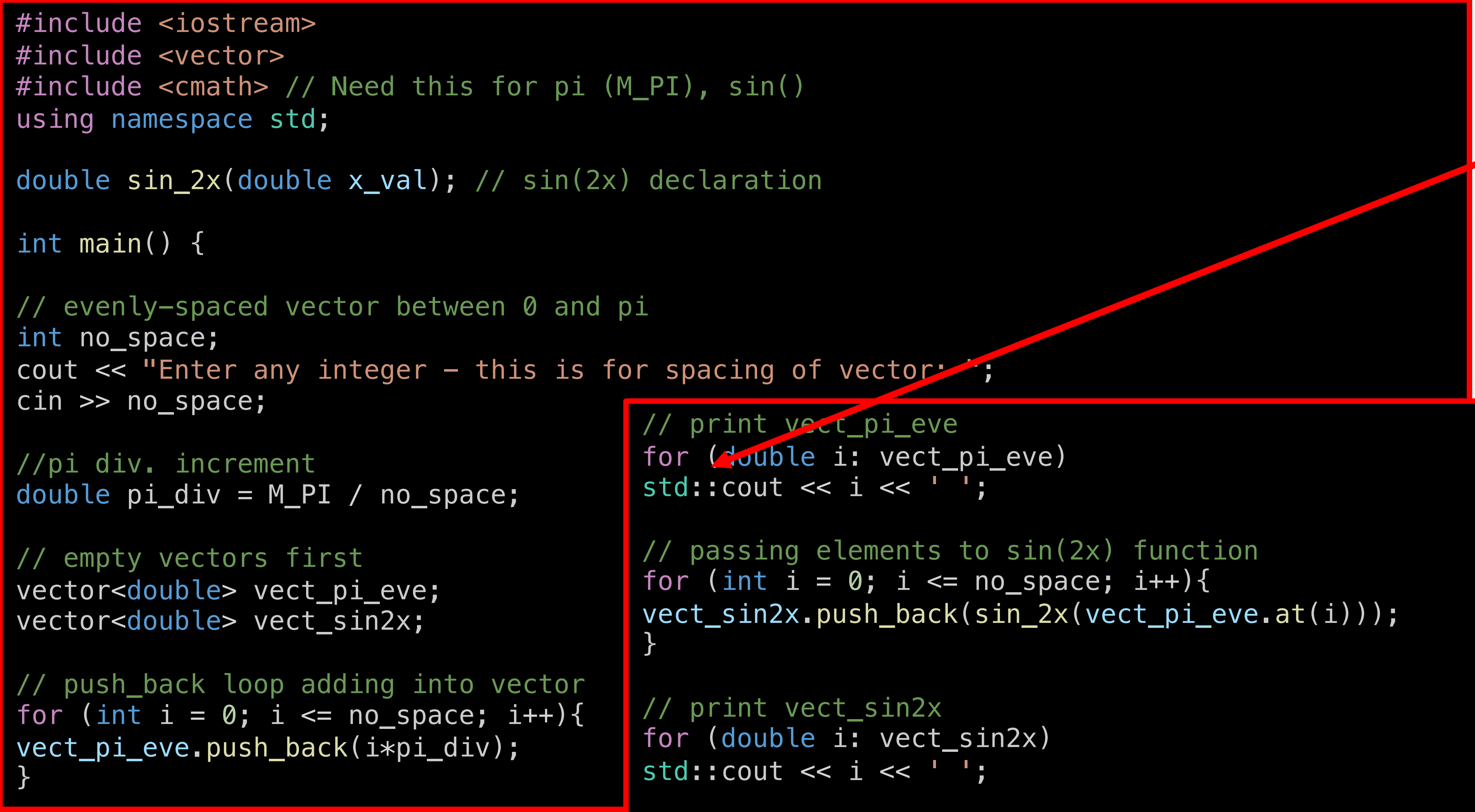
LIV.INNO



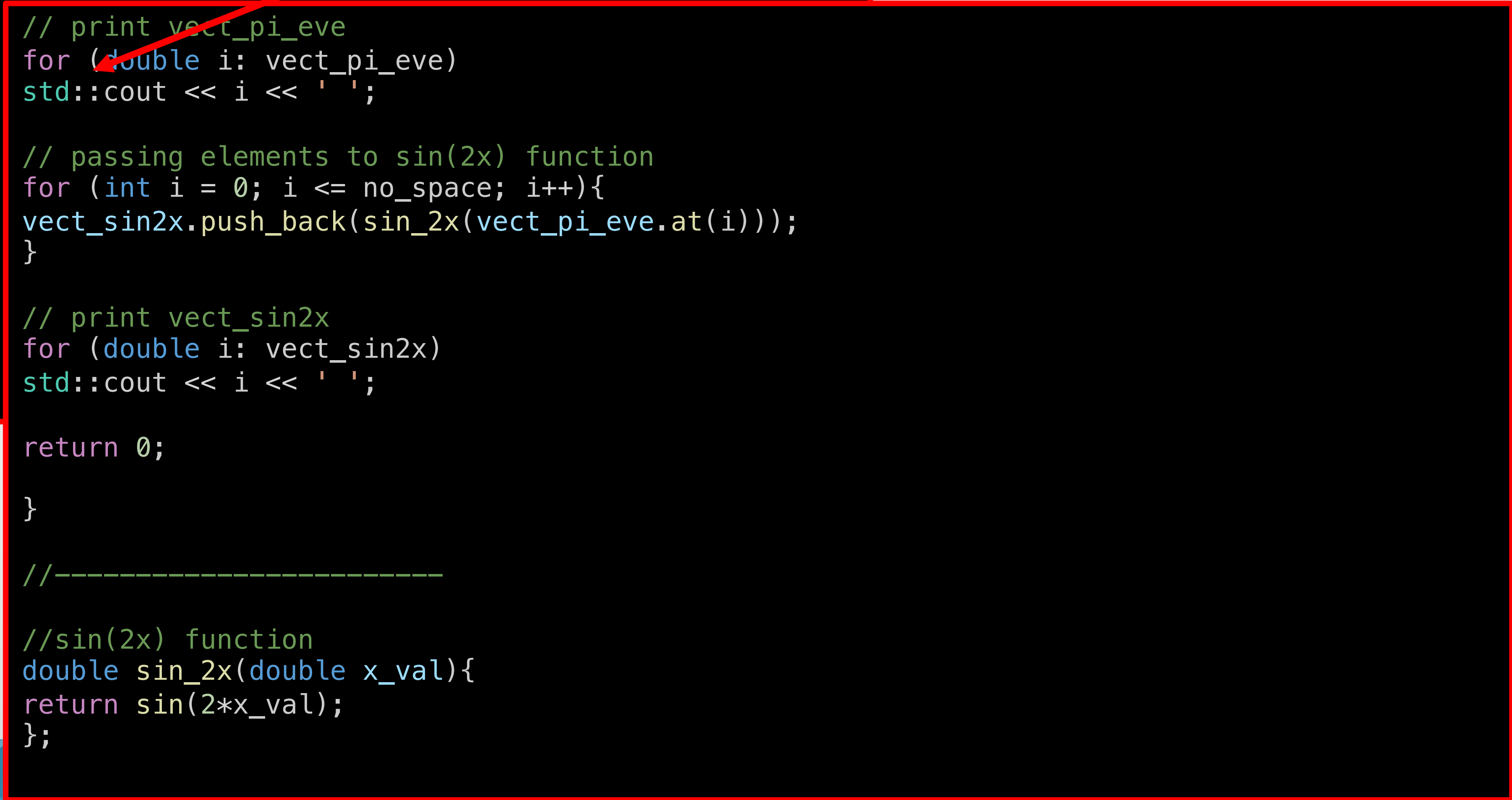
```cpp
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;
double sin_2x(double x) { //function delcaration;
expects one arg of type double called x
return sin(2 * x); //function definition; uses sin
function from cmath
}
int main() {
const double pi = M_PI; //uses pi constant from cmath
pi = 5;
const int numPoints = 9; //number of steps
double step = pi / (numPoints - 1); //calculates step
size
double values[numPoints]; //array to store evenly
spaced points between 0-pi
double results[numPoints]; //new array to store results
of sin_2x
for (int i = 0; i < numPoints; ++i) { //loop
values[i] = i * step; //fills values with points
between 0-pi
results[i] = sin_2x(values[i]); //calculate sin(2x) and
stores it in results
//prints outcome
cout <<"sin(2 * " << values[i] << ") = " << results[i]
<< endl;
}
return 0;}
```

```
$ g++ -o output emily.cpp
emily.cpp:10:8: error: cannot assign to
variable 'pi' with const-qualified type
'const double'
    pi = 5;
    ~~ ^
emily.cpp:9:18: note: variable 'pi'
declared const here
    const double pi = M_PI; //uses pi
constant from cmath
    ~~~~~~~~~~~~~^~~~~~~~
1 error generated.
```

Emily



```cpp
#include <iostream>
#include <vector>
#include <cmath> // Need this for pi (M_PI), sin()
using namespace std;

double sin_2x(double x_val); // sin(2x) declaration

int main() {

// evenly-spaced vector between 0 and pi
int no_space;
cout << "Enter any integer - this is for spacing of vector: ";
cin >> no_space;

//pi div. increment
double pi_div = M_PI / no_space;

// empty vectors first
vector<double> vect_pi_eve;
vector<double> vect_sin2x;

// push_back loop adding into vector
for (int i = 0; i <= no_space; i++){
vect_pi_eve.push_back(i*pi_div);
}
```



```cpp
// print vect_pi_eve
for (double i: vect_pi_eve)
std::cout << i << ' ';

// passing elements to sin(2x) function
for (int i = 0; i <= no_space; i++){
vect_sin2x.push_back(sin_2x(vect_pi_eve.at(i)));
}

// print vect_sin2x
for (double i: vect_sin2x)
std::cout << i << ' ';

return 0;

}

//------------------------

//sin(2x) function
double sin_2x(double x_val){
return sin(2*x_val);
};
```

Don't need
std here

Khang

```
$ g++ -std=c++11 -o output khang.cpp
(base) alexhill at Alexs-MacBook-Air-2 in
~/Documents/UOL/LIVINNO/Teaching/C++_Works
hops/2023/WS3/scripts
$ ./output
Enter any integer - this is for spacing of
vector: 6
0 0.523599 1.0472 1.5708 2.0944 2.61799
3.14159 0 0.866025 0.866025 1.22465e-16 -
0.866025 -0.866025 -2.44929e-16 (base)
alexhill at Alexs-MacBook-Air-2 in
~/Documents/UOL/LIVINNO/Teaching/C++_Works
hops/2023/WS3/scripts
```

```cpp
// print vect_sin2x
for (double i: vect_pi_eve)
std::cout << i << endl;

for (double i: vect_sin2x)
std::cout << i << endl;
```

Luke

```cpp
double sin2x(double a){

return sin(2 * a);

}

int main() {

int n = 8;

vector<double> x_array(n);
vector<double> y_array(n);

for (int i=0; i<n-1; i++){
x_array[i] = M_PI * i / n;
}

for (int i=0; i<n-1; i++){
double b = sin2x(x_array[i]);
y_array[i] = b;
}
```

```
$ g++ -std=c++11 -o output luke.cpp
(base) alexhill at Alexs-MacBook-Air-2 in
~/Documents/UOL/LIVINNO/Teaching/C++_Workshops/2023/WS3/
scripts
$ ./output
X values
0
0.392699
0.785398
1.1781
1.5708
1.9635
2.35619
0
Sin 2X values
0
0.707107
1
0.707107
1.22465e-16
-0.707107
-1
0
```

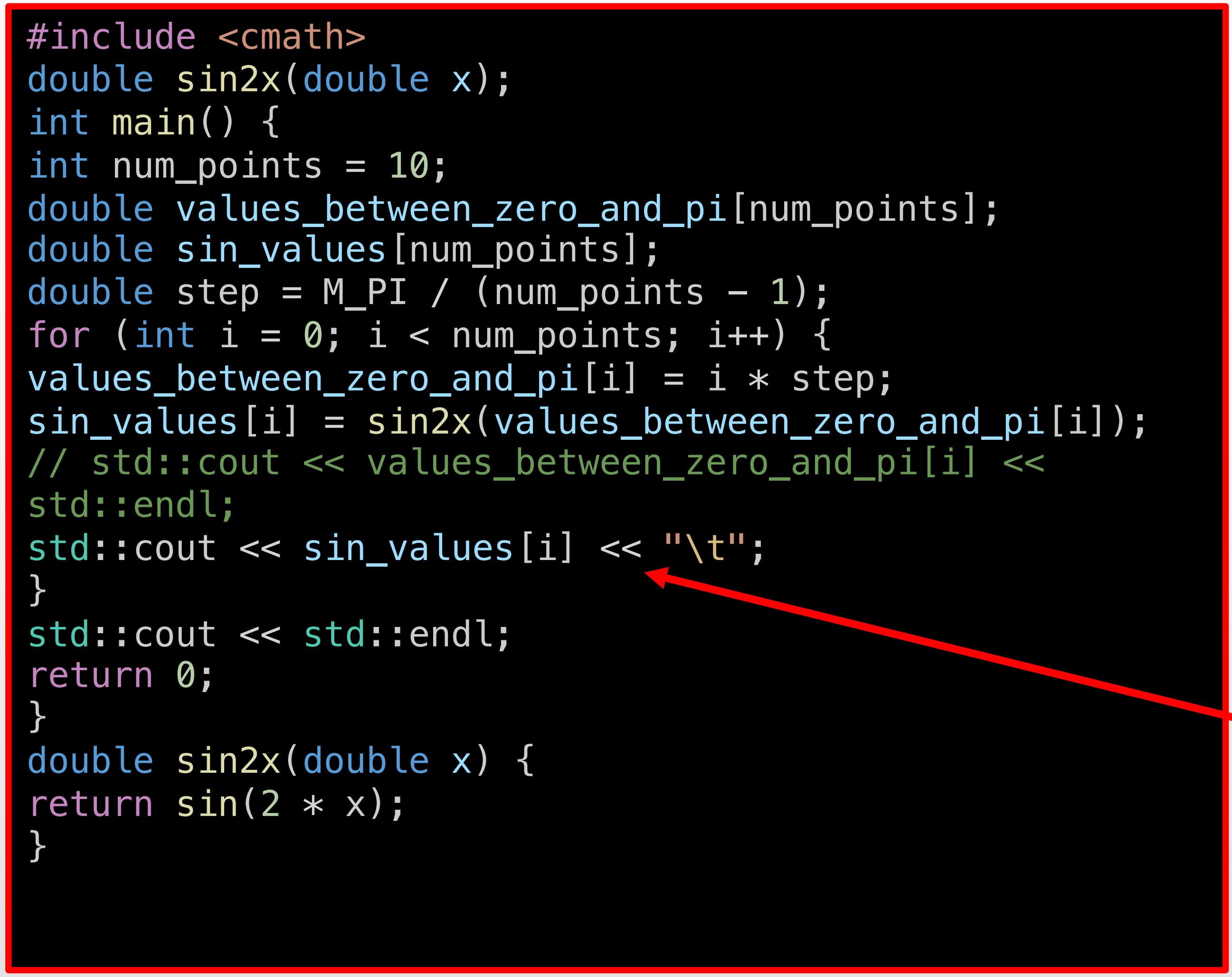
We aren't going to the edge of the array here

Sam

```cpp
#include <cmath>
double sin2x(double x);
int main() {
int num_points = 10;
double values_between_zero_and_pi[num_points];
double sin_values[num_points];
double step = M_PI / (num_points - 1);
for (int i = 0; i < num_points; i++) {
values_between_zero_and_pi[i] = i * step;
sin_values[i] = sin2x(values_between_zero_and_pi[i]);
// std::cout << values_between_zero_and_pi[i] <<
std::endl;
std::cout << sin_values[i] << "\t";
}
std::cout << std::endl;
return 0;
}
double sin2x(double x) {
return sin(2 * x);
}
```

```
$ g++ -std=c++11 -o output sam.cpp
sam.cpp:12:14: error: no member named 'cout' in
namespace 'std'
        std::cout << sin_values[i] << "\t";
        ~~~~~^
sam.cpp:14:10: error: no member named 'cout' in
namespace 'std'
    std::cout << std::endl;
    ~~~~~^
sam.cpp:14:23: error: no member named 'endl' in
namespace 'std'
    std::cout << std::endl;
                 ~~~~~^
3 errors generated.
```

Sam



```
#include <cmath>
double sin2x(double x);
int main() {
int num_points = 10;
double values_between_zero_and_pi[num_points];
double sin_values[num_points];
double step = M_PI / (num_points - 1);
for (int i = 0; i < num_points; i++) {
values_between_zero_and_pi[i] = i * step;
sin_values[i] = sin2x(values_between_zero_and_pi[i]);
// std::cout << values_between_zero_and_pi[i] <<
std::endl;
std::cout << sin_values[i] << "\t";
}
std::cout << std::endl;
return 0;
}
double sin2x(double x) {
return sin(2 * x);
}
```

```
$ ./output
0 0.642788 0.984808 0.866025 0.34202 -
0.34202 -0.866025 -0.984808 -0.642788 -
2.44929e-16
```

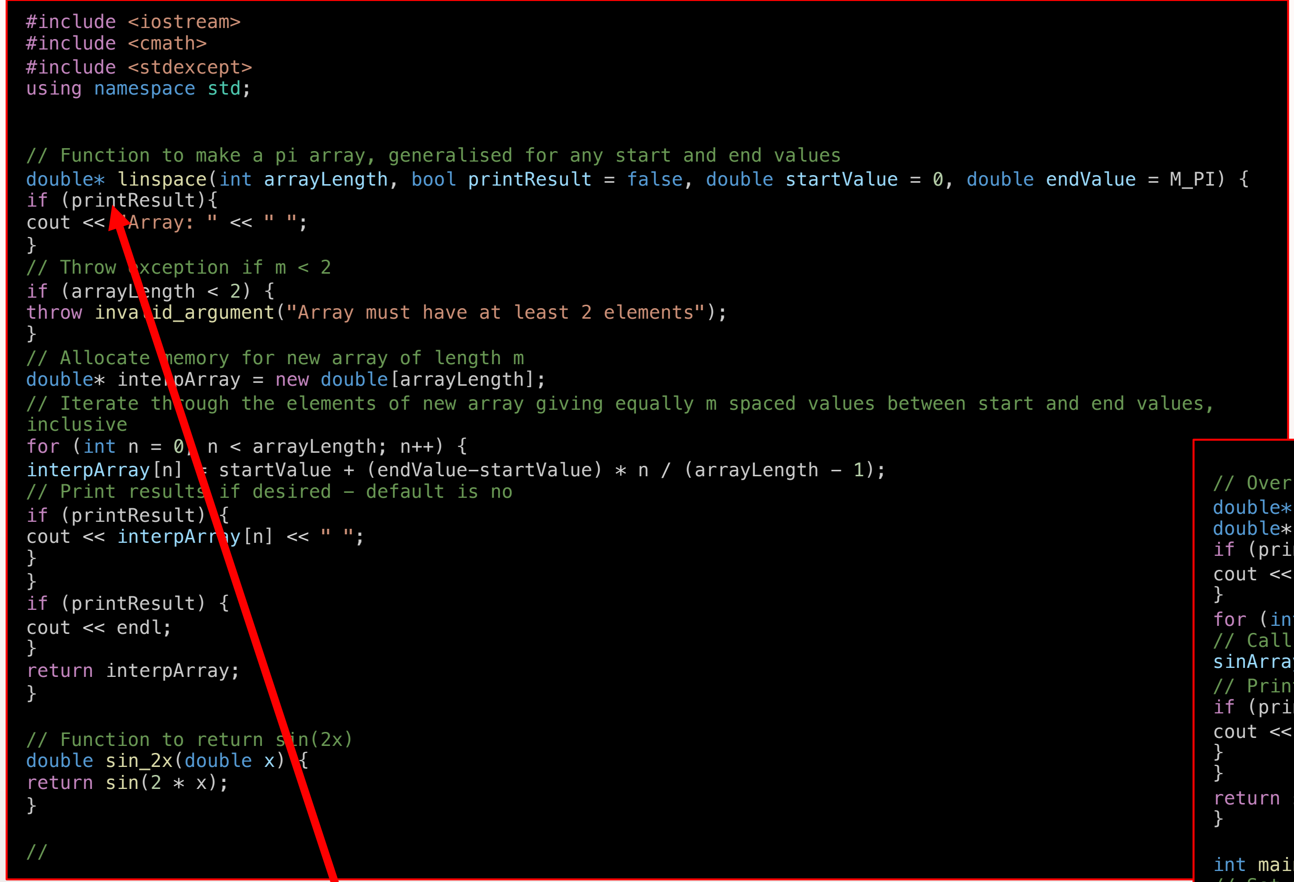
Consider some easier to read formatting, e.g. line breaks and associated string

Joe

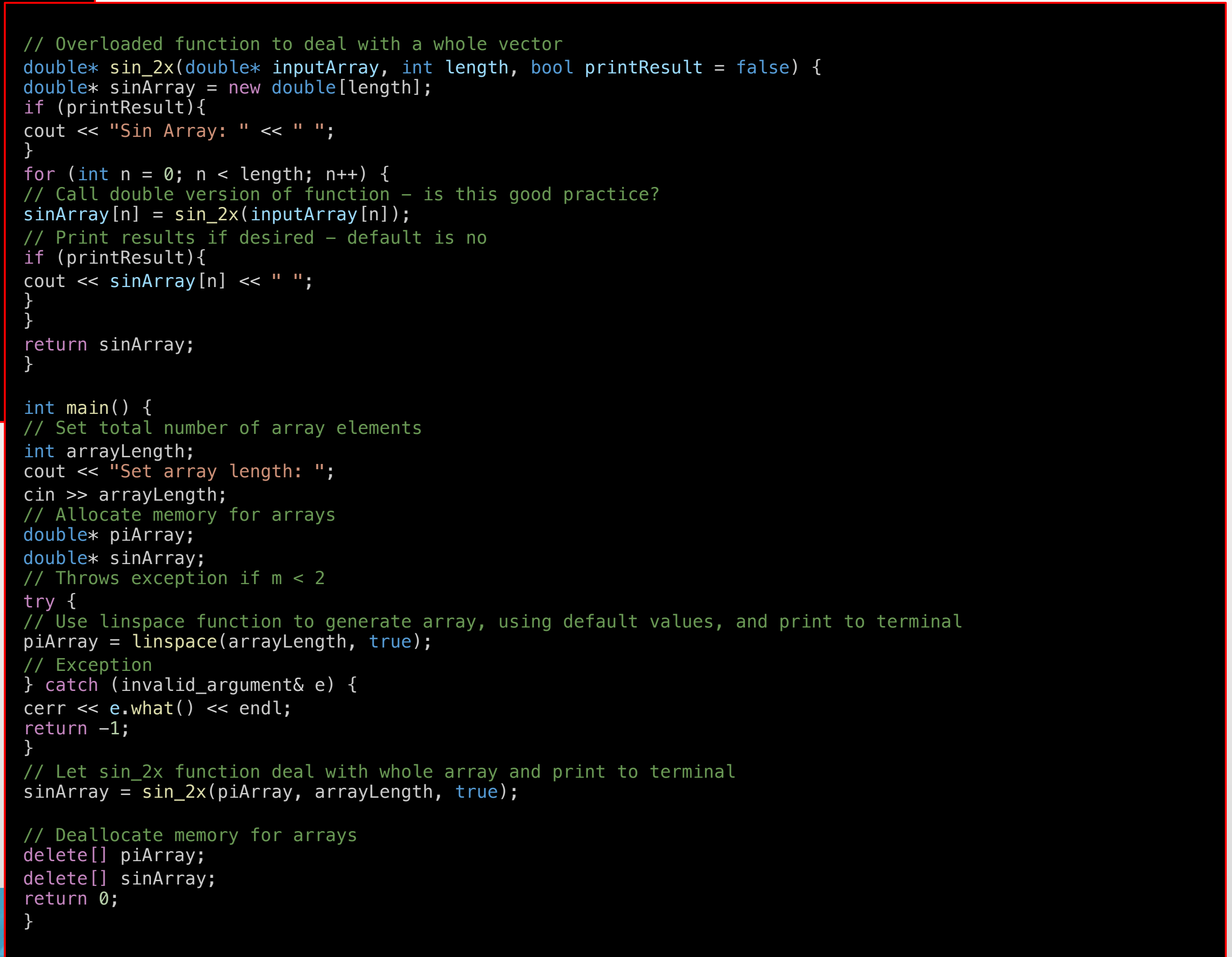
```cpp
#include <iostream>
#include <cmath>
using namespace std;


// Function that just returns sin(2x)
double sin_2x(double x) {
return sin(2 * x);
}


int main() {
// Set total number of array elements
int arrayLength;
cout << "Set array length: ";
cin >> arrayLength;
// Declare pi array and sin array
double piArray[arrayLength];
double sinArray[arrayLength];
// Print to terminal
cout << "Pi Array: ";
// For loop to build pi array and print to terminal:
for (int n = 0; n < arrayLength; n++) {
piArray[n] = M_PI * n / (arrayLength - 1);
cout << piArray[n] << " ";
}

cout << endl << "Sin Array: ";

// For loop to call the sin_2x function and print to terminal:
for (int n = 0; n < arrayLength; n++) {
sinArray[n] = sin_2x(piArray[n]);
cout << sinArray[n] << " ";
}

return 0;
}
```

```
$ g++ -std=c++11 -o output Joe\ Hadley\ Workshop\ 2\
Homework.cpp
(base) alexhill at Alexs-MacBook-Air-2 in
~/Documents/UOL/LIVINNO/Teaching/C++_Workshops/2023/WS3
/scripts
$ ./output
Set array length: 10
Pi Array: 0 0.349066 0.698132 1.0472 1.39626 1.74533
2.0944 2.44346 2.79253 3.14159
Sin Array: 0 0.642788 0.984808 0.866025 0.34202 -
0.34202 -0.866025 -0.984808 -0.642788 -2.44929e-16
(base) alexhill at Alexs-MacBook-Air-2 in
~/Documents/UOL/LIVINNO/Teaching/C++_Workshops/2023/WS3
/scripts
```

Joe



```cpp
#include <iostream>
#include <cmath>
#include <stdexcept>
using namespace std;


// Function to make a pi array, generalised for any start and end values
double* linspace(int arrayLength, bool printResult = false, double startValue = 0, double endValue = M_PI) {
if (printResult){
cout << "Array: " << " ";
}
// Throw exception if m < 2
if (arrayLength < 2) {
throw invalid_argument("Array must have at least 2 elements");
}
// Allocate memory for new array of length m
double* interpArray = new double[arrayLength];
// Iterate through the elements of new array giving equally m spaced values between start and end values,
inclusive
for (int n = 0; n < arrayLength; n++) {
interpArray[n] = startValue + (endValue-startValue) * n / (arrayLength - 1);
// Print results if desired - default is no
if (printResult) {
cout << interpArray[n] << " ";
}
}
if (printResult) {
cout << endl;
}
return interpArray;
}

// Function to return sin(2x)
double sin_2x(double x) {
return sin(2 * x);
}

//
```



```cpp
// Overloaded function to deal with a whole vector
double* sin_2x(double* inputArray, int length, bool printResult = false) {
double* sinArray = new double[length];
if (printResult){
cout << "Sin Array: " << " ";
}
for (int n = 0; n < length; n++) {
// Call double version of function - is this good practice?
sinArray[n] = sin_2x(inputArray[n]);
// Print results if desired - default is no
if (printResult){
cout << sinArray[n] << " ";
}
}
return sinArray;
}

int main() {
// Set total number of array elements
int arrayLength;
cout << "Set array length: ";
cin >> arrayLength;
// Allocate memory for arrays
double* piArray;
double* sinArray;
// Throws exception if m < 2
try {
// Use linspace function to generate array, using default values, and print to terminal
piArray = linspace(arrayLength, true);
// Exception
} catch (invalid_argument& e) {
cerr << e.what() << endl;
return -1;
}
// Let sin_2x function deal with whole array and print to terminal
sinArray = sin_2x(piArray, arrayLength, true);

// Deallocate memory for arrays
delete[] piArray;
delete[] sinArray;
return 0;
}
```

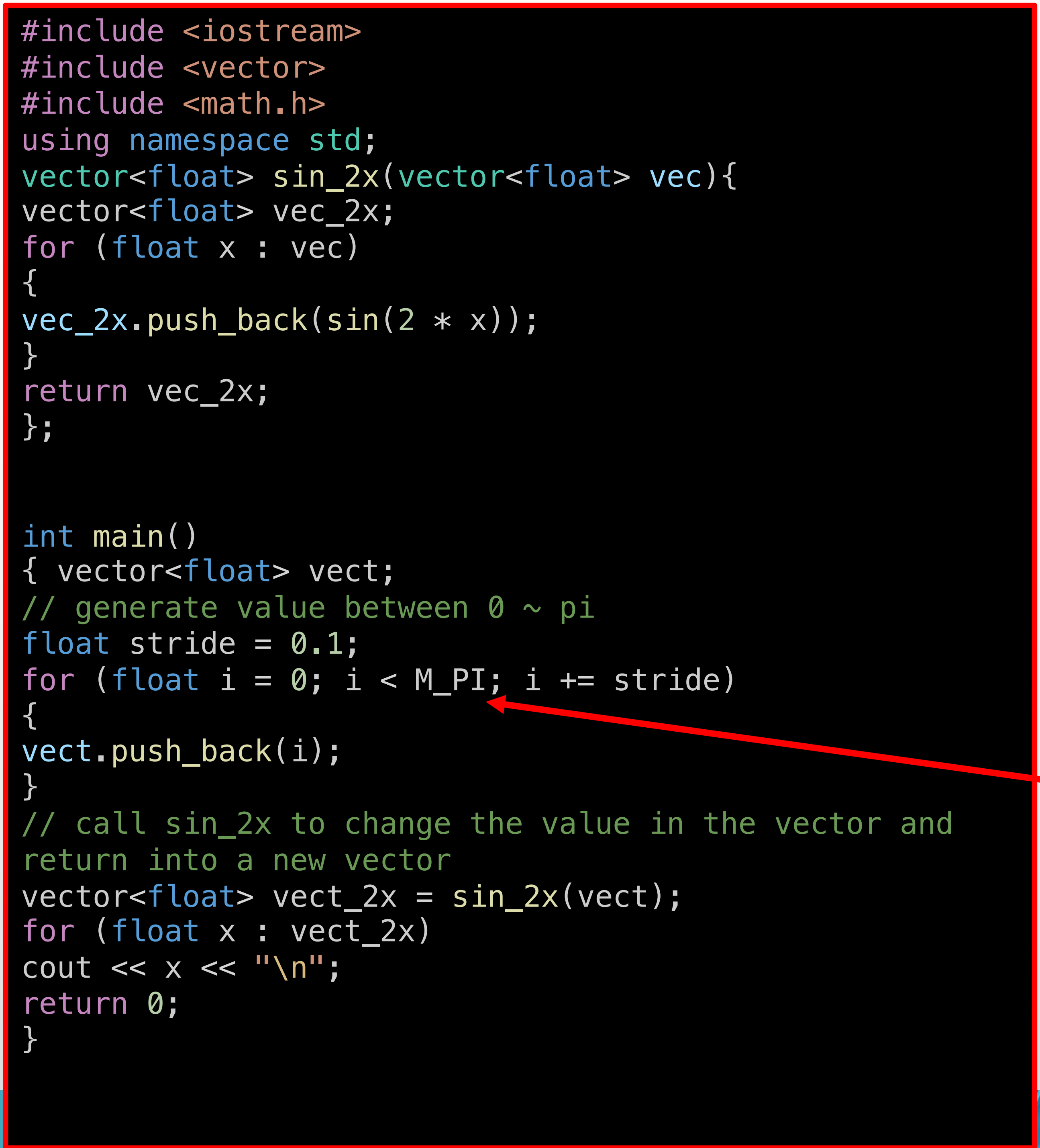
Use of pointers and creation of python-like functions 👍

```cpp
#include <cmath>
#include <vector>
#include <algorithm>
#include <iostream>

using namespace std;

double sin_2x(double x);

int main(){
int n = 10;
vector<double> v1(n), v2(n);

// fill with values between 0 and pi
double d = M_PI/(n-1);

std::generate(v1.begin(),v1.end(), [i=0, &d]() mutable {return d*i++;});

std::cout << "x = ";
for (double j: v1) {
std::cout << j << ", ";
}
std::cout << endl;

std::generate(v2.begin(),v2.end(), [i=-1, &v1]() mutable {i++; return sin_2x(v1.at(i));});
std::cout << "sin(2x) = ";
for (double j: v2) {
std::cout << j << ", ";
}
std::cout << endl;

return 0;
}

double sin_2x(double x){
auto sin2x = sin(2*x);
return sin2x;}
```

```
$ g++ -std=c++14 -o output marina.cpp
(base) alexhill at Alexs-MacBook-Air-2 in
~/Documents/UOL/LIVINNO/Teaching/C++_Workshops/2023/WS3
/scripts
$ ./output
x = 0, 0.349066, 0.698132, 1.0472, 1.39626, 1.74533,
2.0944, 2.44346, 2.79253, 3.14159,
sin(2x) = 0, 0.642788, 0.984808, 0.866025, 0.34202, -
0.34202, -0.866025, -0.984808, -0.642788, -2.44929e-
16,
```

Nice split declaration and definition

Marina



```cpp
#include <iostream>
#include <vector>
#include <math.h>
using namespace std;
vector<float> sin_2x(vector<float> vec){
vector<float> vec_2x;
for (float x : vec)
{
vec_2x.push_back(sin(2 * x));
}
return vec_2x;
};


int main()
{ vector<float> vect;
// generate value between 0 ~ pi
float stride = 0.1;
for (float i = 0; i < M_PI; i += stride)
{
vect.push_back(i);
}
// call sin_2x to change the value in the vector and
return into a new vector
vector<float> vect_2x = sin_2x(vect);
for (float x : vect_2x)
cout << x << "\n";
return 0;
}
```

```
$ g++ -std=c++11 -o output qiyuan.cpp
(base) alexhill at Alexs-MacBook-Air-2 in
~/Documents/UOL/LIVINNO/Teaching/C++_Workshops/2023/WS3/scripts
$ ./output
0
0.198669
0.389418
0.564642
0.717356
0.841471
0.932039
0.98545
0.999574
0.973848
0.909297
0.808496
0.675463
0.515501
0.334988
0.14112
-0.0583747
-0.255542
-0.442521
-0.611858
-0.756803
-0.871576
-0.951602
-0.993691
-0.996165
-0.958924
-0.883455
-0.772765
-0.631267
-0.464603
-0.279417
-0.083091
```

Qiyuan

You're not getting quite up to pi with this array

Sinead

```
#include <iostream>
#include <cmath>
using namespace std;

int main() {

double num_array[5] = {0, M_PI_4 , M_PI_2
, 3*M_PI_4 , M_PI};
double result[5];
double sin_2x;
int n;

for (int i=0; i<5; i++){
//cout << num_array[i] << endl;
sin_2x = sin(2*num_array[i]);
result[i]= sin_2x;
cout << "sin(2x)=" << result[i]<< endl;
}

return 0;
}
```

```
$ g++ -std=c++11 -o output
sinead.cpp
(base) alexhill at Alexs-MacBook-
Air-2 in
~/Documents/UOL/LIVINNO/Teaching/C++
_Workshops/2023/WS3/scripts
$ ./output
sin(2x)=0
sin(2x)=1
sin(2x)=1.22465e-16
sin(2x)=-1
sin(2x)=-2.44929e-16
```

Consider generalising this

Sakrican

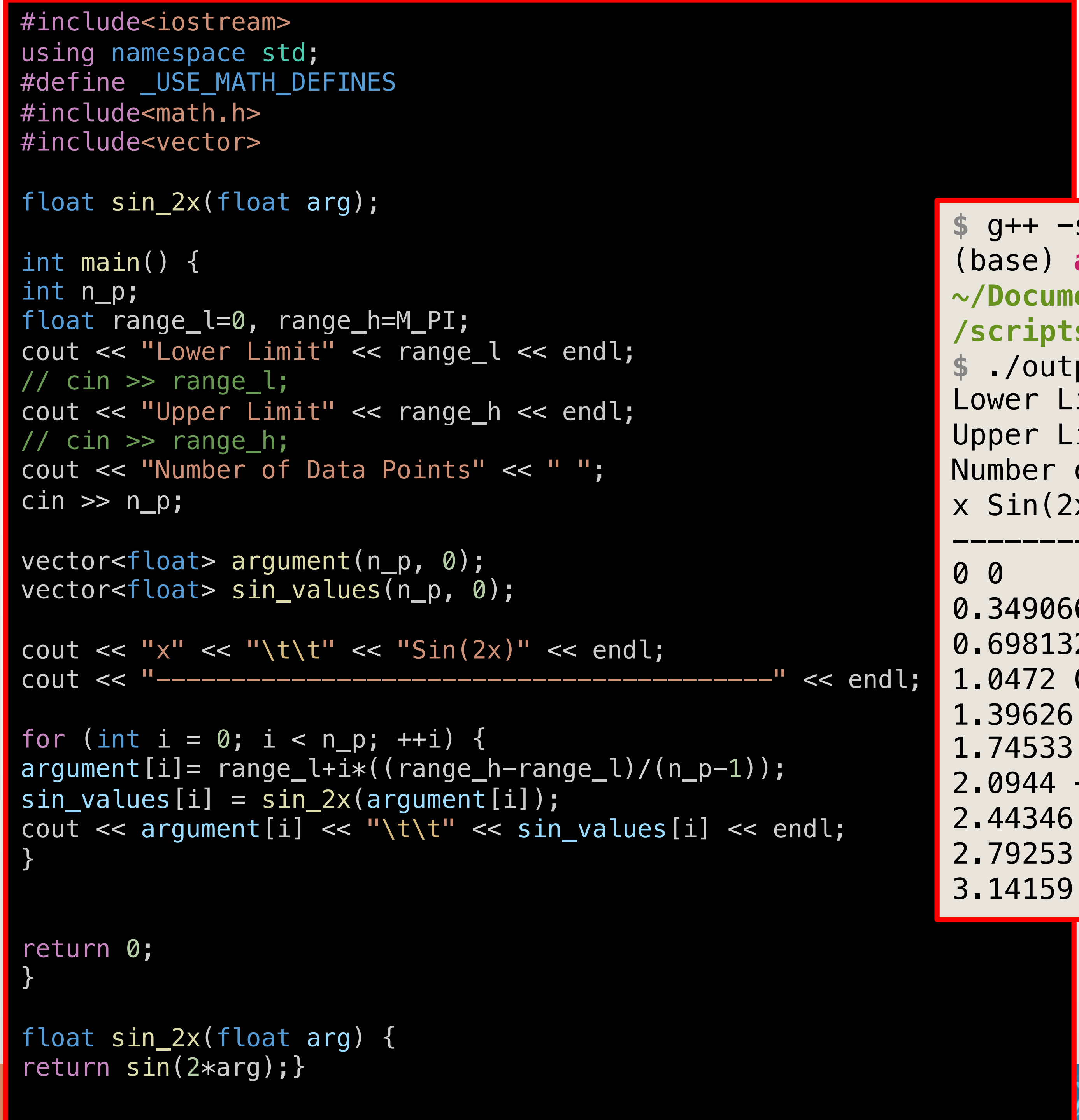
```cpp
#define _USE_MATH_DEFINES

#include <cmath>
#include <iostream>
#include <vector>

using namespace std;


float sin_2x(float val_x);

int main() {


int vector_size;
cout << "Enter the desired length of the array/vector: ";
cin >> vector_size;

float step_size;

step_size = M_PI / (vector_size-1);

vector<float> vector1{0};

cout << "Evenly spaced vector between 0 and pi " << endl;
cout << 0 << " " << vector1.at(0) << endl;
```

Not always needed
(depends on compiler)

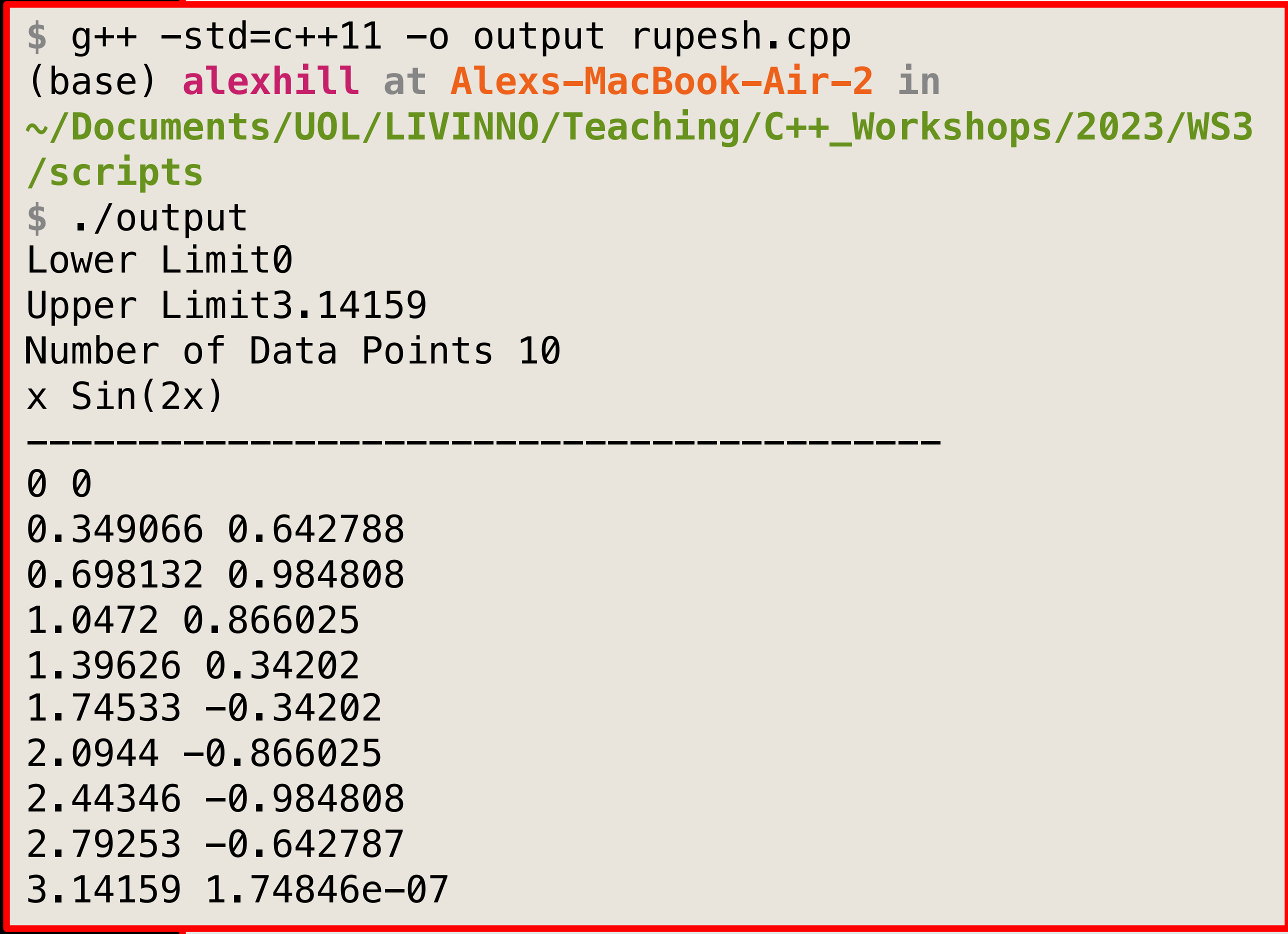
```cpp
for(int i=1; i <= vector_size-1 ; ++i) {
vector1.push_back(step_size*i);
cout << i << " " << vector1.at(i) << endl;
}

vector<float> vector_sin2x{0};

cout << "The vector and the sin2x values " << endl;
cout << 0 << " " << vector1.at(0) << " " << vector_sin2x.at(0) <<
endl;

for(int i=1; i <= vector_size-1 ; ++i) {
vector_sin2x.push_back(sin_2x(vector1.at(i)));
cout << i << " " << vector1.at(i) << " " << vector_sin2x.at(i) <<
endl;
}


return 0;
}

float sin_2x(float val_x){

float val_2x;
val_2x = 2*val_x;

return sin(val_2x);
}
```

Sakrican

```
$ g++ -std=c++11 -o output sakrican.cpp
(base) alexhill at Alexs-MacBook-Air-2 in
~/Documents/UOL/LIVINNO/Teaching/C++_Workshops/2023/WS3/scripts
$ ./output
Enter the desired length of the array/vector: 10
Evenly spaced vector between 0 and pi
0 0
1 0.349066
2 0.698132
3 1.0472
4 1.39626
5 1.74533
6 2.0944
7 2.44346
8 2.79253
9 3.14159
The vector and the sin2x values
0 0 0
1 0.349066 0.642788
2 0.698132 0.984808
3 1.0472 0.866025
4 1.39626 0.34202
5 1.74533 -0.34202
6 2.0944 -0.866025
7 2.44346 -0.984808
8 2.79253 -0.642788
9 3.14159 -3.01992e-07
```



```cpp
#include<iostream>
using namespace std;
#define _USE_MATH_DEFINES
#include<math.h>
#include<vector>

float sin_2x(float arg);

int main() {
int n_p;
float range_l=0, range_h=M_PI;
cout << "Lower Limit" << range_l << endl;
// cin >> range_l;
cout << "Upper Limit" << range_h << endl;
// cin >> range_h;
cout << "Number of Data Points" << " ";
cin >> n_p;

vector<float> argument(n_p, 0);
vector<float> sin_values(n_p, 0);

cout << "x" << "\t\t" << "Sin(2x)" << endl;
cout << "-----------------------------------------" << endl;

for (int i = 0; i < n_p; ++i) {
argument[i]= range_l+i*((range_h-range_l)/(n_p-1));
sin_values[i] = sin_2x(argument[i]);
cout << argument[i] << "\t\t" << sin_values[i] << endl;
}


return 0;
}

float sin_2x(float arg) {
return sin(2*arg);}
```



```
$ g++ -std=c++11 -o output rupesh.cpp
(base) alexhill at Alexs-MacBook-Air-2 in
~/Documents/UOL/LIVINNO/Teaching/C++_Workshops/2023/WS3
/scripts
$ ./output
Lower Limit0
Upper Limit3.14159
Number of Data Points 10
x Sin(2x)
-----------------------------------------
0 0
0.349066 0.642788
0.698132 0.984808
1.0472 0.866025
1.39626 0.34202
1.74533 -0.34202
2.0944 -0.866025
2.44346 -0.984808
2.79253 -0.642787
3.14159 1.74846e-07
```

Rupesh

```cpp
#include <iostream>
#include <cmath>
#include <vector>
using namespace std;

double sin2x(double theta);

int main() {
double k = 5;
vector<double> v1(k, M_PI);
vector<double> v2 = {};

for (int i = 0; i <= k-1; ++i)
{
v1.at(i) = i * (1.0/(k-1.0)) * v1.at(i);
v2.push_back( sin2x(v1.at(i)) );
}

for (int i = 0; i <= k-1; ++i)
cout << "Theta = " << v1.at(i) << ", sin(2theta) = " << v2.at(i) << endl;



return 0;
}

double sin2x(double theta)
{
return sin(2.0 * theta);
}
```

Alex H

```
$ ./run
Theta = 0, sin(2theta) = 0
Theta = 0.785398, sin(2theta) = 1
Theta = 1.5708, sin(2theta) = 1.22465e-16
Theta = 2.35619, sin(2theta) = -1
Theta = 3.14159, sin(2theta) = -2.44929e-16
```

LIV.INNO

Takeaways

- Use vectors if possible to stop bound issues
- Consider generalising your code early in the process

Hang on…

- Why is sin(2pi) not exactly 0?
- M_PI = 3.141592653589793, not exactly pi!

```cpp
#include <iostream>
#include <cmath>
#include <vector>
#include <iomanip>
using namespace std;


int main() {
float pi_float = M_PI;
double pi_double = M_PI;
float sin_2_pi_float = sin(2*pi_float);
double sin_2_pi_double = sin(2*pi_double);

cout << "pi_float = " << pi_float << ", sin(2.pi_float) = " << sin_2_pi_float << endl;
cout << "pi_double = " << pi_double << ", sin(2.pi_double) = " << sin_2_pi_double << endl;

cout << setprecision(16);
cout << "pi_float = " << pi_float << ", sin(2.pi_float) = " << sin_2_pi_float << endl;
cout << "pi_double = " << pi_double << ", sin(2.pi_double) = " << sin_2_pi_double << endl;

return 0;
}
```

```
$ ./output
pi_float = 3.14159, sin(2.pi_float) = 1.74846e-07
pi_double = 3.14159, sin(2.pi_double) = -2.44929e-16
pi_float = 3.141592741012573, sin(2.pi_float) = 1.748455531469517e-07
pi_double = 3.141592653589793, sin(2.pi_double) = -2.449293598294706e-16
```

Aim of Workshop Three

- Passing vectors into functions (pointers)
- Plotting data (really this time)

Resources

- alex-hill94.github.io/#WS3
- https://www.programiz.com/cpp-programming/online-compiler/?ref=1a2efafc
- https://www.geeksforgeeks.org/pointers-and-references-in-c/
- https://www.w3schools.com/cpp/cpp_pointers.asp

LIV.INNO

POINTERS

- References, memory addresses and pointers
- Returning multiple values from functions
- Passing arrays into functions
- Passing vectors into functions

Memory Address

- The ampersand (&) can be used to get the memory address of a variable
- This is usually in the form of a hexadecimal

```cpp
#include <iostream>
using namespace std;


int main() {
int height = 10; // height variable
cout << "height = " << height << '\n';
cout << "height address = " << &height << '\n';

return 0;
}
```

```
$ ./run
height = 10
height address = 0x16bd133d8
```

References

- You can create a ‘reference variable’ to an existing variable using the ampersand, &
- This is effectively an alias to an already existing variable

```cpp
#include <iostream>

using namespace std;


int main() {
int height = 10; // height variable
int &ref = height; // first reference to height
int ref1 = height; // second reference to height
height = 9;
cout << "height = " << height << '\n';
cout << "ref = " << ref << '\n';
cout << "ref1 = " << ref1 << '\n';

return 0;
}
```

```
$ ./run
height = 9
ref = 9
ref1 = 10
```

LIV.INNO

References

```cpp
#include <iostream>
using namespace std;

int main() {
int height = 10; // height variable
int &ref = height; // reference to height
int copy = height; // copy of height
height = 9;

cout << "height = " << height << '\n';
cout << "&height = " << &height << '\n';
cout << "ref = " << ref << '\n';
cout << "&ref = " << &ref << '\n';
cout << "copy = " << copy << '\n';
return 0;

}
```

The placement of & is important for your chosen purpose

```
$ ./output
height = 9
&height = 0x16d98b3d8
ref = 9
&ref = 0x16d98b3d8
copy = 10
```

Memory address

- Why do we care?

- C++ allows us to manipulate the computer’s memory, which can make code writing and performance more efficient

Pointers

- We can create a variable that saves the memory address of another variable, known as a pointer
- These require the use of an asterisk, *

```
#include <iostream>
using namespace std;


int main() {
int height; // height variable
height = 10;
int* pointer = &height;
cout << "height address = " << &height << '\n';
cout << "pointer = " << pointer << '\n';
cout << "height = " << height << '\n';

return 0;
}
```

```
$ ./run
height address = 0x16dd6f3d8
pointer = 0x16dd6f3d8
height = 10
```

Pointers

```
#include <iostream>
using namespace std;


int main() {
int height; // height variable
height = 10;
int* pointer1 = &height;
int * pointer2 = &height;
int *pointer3 = &height;
cout << "height address = " << &height << '\n';
cout << "pointer1 = " << pointer1 << '\n';
cout << "pointer2 = " << pointer2 << '\n';
cout << "pointer3 = " << pointer3 << '\n';
cout << "height = " << height << '\n';

return 0;
}
```

You can place the asterisk anywhere, but the convention is `int* pointer1`

```
$ ./output
height address = 0x16b4d33d8
pointer1 = 0x16b4d33d8
pointer2 = 0x16b4d33d8
pointer3 = 0x16b4d33d8
height = 10
```

Pointers

- Make sure the data type of the pointer matches the variable!

```cpp
#include <iostream>
using namespace std;


int main() {
int height = 10; // height variable
int* pointer = &height;
string name = "Alex";
int* name_ptr = &name;
cout << "height = " << height << ", height address = " <<
pointer << '\n’;

cout << "name = " << name << ", name address = " << name_ptr
<< '\n';

return 0;
}
```

```
$ g++ -o output test.cpp
test.cpp:8:6: error: cannot initialize a variable of
type 'int *' with an rvalue of type 'std::string *'
(aka 'basic_string<char> *')
int* name_ptr = &name;
     ^          ~~~~~
1 error generated.
```

Deferencing

- We can get the value of the variable that the pointer is pointing at using * again

```cpp
#include <iostream>
using namespace std;


int main() {
int height = 10; // height variable
int* pointer = &height;

cout << "variable = " << height << endl;
cout << "address = " << pointer << endl;
cout << "address value = " << *pointer << endl;
return 0;
}
```

```
$ ./run
variable = 10
address = 0x16dd0f3d8
address value = 10
```

Modifying variables with pointers

```cpp
#include <iostream>
using namespace std;


int main() {
int height = 10; // height variable
int* pointer = &height;
cout << "variable = " << height << endl;
cout << "address = " << pointer << endl;
cout << "address value = " << *pointer << endl;

*pointer = 12;

cout << "variable = " << height << endl;
cout << "address = " << pointer << endl;
cout << "address value = " << *pointer << endl;

return 0;
}
```

```
$ ./run
variable = 10
address = 0x16d8433d8
address value = 10
variable = 12
address = 0x16d8433d8
address value = 12
```

Arrays as Pointers

```cpp
#include <iostream>
using namespace std;

int main()
{
int arr[] = { 1, 2, 3, 4, 5, 6, 7, 8 };
cout << "arr = " << arr << "\n";
cout << "arr[0] = " << arr[0] << "\n";

return 0;
}
```

```
$ ./output
arr = 0x16b2ab3b0
arr[0] = 1
```

Arrays as Pointers

```cpp
#include <iostream>
using namespace std;

int main()
{
int a = 1;
int* b = &a;
cout << "a = " << a << "\n";
cout << "b = " << b << "\n";
cout << "b[0] = " << b[0] << "\n";
cout << "b[1] = " << b[1] << "\n";
cout << "b[2] = " << b[2] << "\n";
cout << "&b[0] = " << &b[0] << "\n";
cout << "&b[1] = " << &b[1] << "\n";


return 0;
}
```

```
$ ./output
a = 1
b = 0x16f6d33d8
b[0] = 1
b[1] = 0
b[2] = 1869428016
&b[0] = 0x16f6d33d8
&b[1] = 0x16f6d33dc
```

LIV.INNO

Challenge five: a few minutes with pointers

- Initialise five variables of type: int, float, double, char, and string
- Create pointer variables of these variables
- Use the pointers to modify the values of the initial variables
- Print the values of the variables and their addresses

Functions and pointers

Pythonic Approach

```cpp
#include <iostream>
using namespace std;
#include <tuple>

tuple <int, int> swap_my_nums(int x, int y)
{
int z = x;
x = y;
y = z;
return make_tuple(x, y);
}

int main() {
int orig_first_number = 10;
int orig_second_number = 20;
int new_first_number;
int new_second_number;

cout << "Before swap: " << "\n";

cout << orig_first_number << " " << orig_second_number << "\n";

// Call the function, which will change the values of first_number and second_number
tie(new_first_number, new_second_number) = swap_my_nums(orig_first_number, orig_second_number);

cout << "After swap: " << "\n";
cout << new_first_number << " " << new_second_number << "\n";

return 0;
}
```

swap_my_nums returns two values

```
$ ./run
Before swap:
10 20
After swap:
20 10
```

LIV.INNO

Passing by Reference

```cpp
#include <iostream>
using namespace std;

void swap_my_nums(int &x, int &y) {
int z = x;
x = y;
y = z;
}

int main() {
int first_number = 10;
int second_number = 20;

cout << "Before swap: " << "\n";
cout << first_number << " " << second_number << "\n";

// Call the function, which will change the values of first_number and
second_number
swap_my_nums(first_number, second_number);

cout << "After swap: " << "\n";
cout << first_number << " " << second_number << "\n";

return 0;
}
```

The function does not make a copy of x and y, it passes the actual variables themselves

```
$ ./run
Before swap:
10 20
After swap:
20 10
```

Passing by pointers

```
#include <iostream>
using namespace std;

void swap_my_nums(int* x, int* y) {
int z = *x;

cout << "x = " << x << endl;
cout << "y = " << y << endl;
cout << "*x = " << *x << endl;
cout << "*y = " << *y << endl;

*x = *y;
*y = z;
}

int main() {
int first_number = 10;
int second_number = 20;

cout << "Before swap: " << "\n";
cout << first_number << " " << second_number << "\n";

// Call the function, which will change the values of first_number and second_number
swap_my_nums(&first_number, &second_number);

cout << "After swap: " << "\n";
cout << first_number << " " << second_number << "\n";

return 0;
}
```

Tells the compiler to expect a pointer to an int variable

Grabs the value at the address the pointer points to

Passes in the memory address

```
$ ./output
Before swap:
10 20
x = 0x16b79f3d8
y = 0x16b79f3d4
*x = 10
*y = 20
After swap:
20 10
```

Passing arrays into functions

```
#include <iostream>
#include <cmath>
using namespace std;

void func(int* a, int* b, int N)
{
int i;
for (i = 0; i < N; i++)
b[i] = a[i] * 2;
}

int main()
{
int arr[] = { 1, 2, 3, 4, 5, 6, 7, 8 };
int arr1[8];
int n = sizeof(arr) / sizeof(arr[0]);
int n1 = sizeof(arr1) / sizeof(arr1[0]);
assert(n==n1);
printf("all good");

func(arr, arr1, n);

for (int i = 0; i < n; i++)
cout << "\n" << arr1[i];

return 0;
}
```

```
$ ./run
all good
2
4
6
8
10
12
14
```

‘Decay’ of arrays in functions

```cpp
void func(int* a, int* b)
{
cout << "a = " << a << endl;
cout << "sizeof(a) = " << sizeof(a) << endl;
cout << "sizeof(a[0]) = " << sizeof(a[0]) << endl;

int N = sizeof(a);

for (int i = 0; i < N; i++)
b[i] = a[i] * 2;
}

int main()
{
int arr[] = { 1, 2, 3, 4, 5, 6, 7, 8 };
int arr1[8];
int n = sizeof(arr) / sizeof(arr[0]);
int n1 = sizeof(arr1) / sizeof(arr1[0]);

cout << "arr = " << arr << endl;
cout << "sizeof(arr) = " << sizeof(arr) << endl;
cout << "sizeof(arr[0]) = " << sizeof(arr[0]) << endl;

func(arr, arr1);

return 0;}
```

When an array is passed into a function, it effectively becomes a pointer to the first element in the array only, so you <u>have</u> to pass in the array length as a variable into the function ahead of time

```
$ ./output
arr = 0x16d8f73b0
sizeof(arr) = 32
sizeof(arr[0]) = 4
all good
a = 0x16d8f73b0
sizeof(a) = 8
sizeof(a[0]) = 4
```

Passing arrays into functions

```
// CPP Program to demonstrate passing
// an array to a function is always treated
// as a pointer
#include <iostream>
#include <cmath>
using namespace std;
// Note that arr[] for fun is
// just a pointer even if square
// brackets are used

void func(int* a, int N)
{
int i;
for (i = 0; i < N; i++)
++a[i];
}

int main()
{
int arr[] = { 1, 2, 3, 4, 5, 6, 7, 8 };
int n = sizeof(arr) / sizeof(arr[0]);
cout << "Array size inside main() is " << n;
func(arr, n);
int i;
for (i = 0; i < n; i++)
cout << "\n" << arr[i];
return 0;
}
```

```
$ ./run
Array size inside main() is 8
2
3
4
5
6
7
8
9
```

LIV.INNO

Challenge Five

- Create an array called x, with values -5 to 5 in main()
- Pass the array to a function called **quad**, which computes the square of all the values in the array, and save the values to another array called y
- Loop over all x and y and check that things have worked right
- Use pointers to minimise the length of your script

LIV.INNO

Challenge Five

```cpp
#include <cmath>
#include <iostream>
using namespace std;

void quad(int* a, int* b , int N)
{
int i;
for (i = 0; i < N; i++)
b[i] = pow(a[i], 2);
}

int main()
{
int x[] = { -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5};
int n = sizeof(x) / sizeof(x[0]);
int y[n];

quad(x, y, n);

int i;
for (i = 0; i < n1; i++)
{ cout << x[i] << "^2 = " << y[i] << "\n" ;
}
return 0;
}
```

```
$ ./run
-5^2 = 25
-4^2 = 16
-3^2 = 9
-2^2 = 4
-1^2 = 1
0^2 = 0
1^2 = 1
2^2 = 4
3^2 = 9
4^2 = 16
5^2 = 25
```

LIV.INNO

Passing vectors into functions

```cpp
#include <iostream>
#include <vector>
using namespace std;

// The vect here is a copy of vect in main()
void func(vector<int> vect)
{ vect.push_back(3); }


int main()
{
vector<int> vect;
vect.push_back(1);
vect.push_back(2);
func(vect);
// vect remains unchanged after function
// call
for (int i = 0; i < vect.size(); i++)
cout << vect[i] << ”\n";
return 0;
}
```

```
$ ./run
1
2
```

- You can pass a full vector into a function, but a full copy is made, which may take a lot of time to work with
- As the function works with the copy of vect, no change is made to `vect` in `main`()

Passing vectors into functions

```
#include <iostream>
#include <vector>
using namespace std;

// The vect here is the same as the vect in main()
void func(vector<int>& vect)
{ vect.push_back(3); }


int main()
{
vector<int> vect;
vect.push_back(1);
vect.push_back(2);
func(vect);
// vect remains unchanged after function
// call
for (int i = 0; i < vect.size(); i++)
cout << vect[i] << ”\n";
return 0;
}
```

```
$ ./run
1
2
3
```

- Making *vect* a reference stops a copy being made
- Changes made in func() now changes the original *vect* in memory
- If we add const in front of vector, *vect* can no longer be changed by func

Challenge four revisited

```cpp
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;

void sin2x(const vector<double>& input, vector<double>& output) {
int n = input.size();
for (int i = 0; i < n; ++i)
output.push_back(sin(2 * input.at(i)));
}

int main(){
int k;
cout << "Provide n_vals: \n";
cin >> k;

vector<double> thetas(k);
vector<double> ans;

for (int i = 0; i < k; ++i)
{
thetas.at(i) = i * (1./(k-1)) * M_PI;
}
sin2x(thetas, ans);

for (double j: thetas){
cout << "Theta = " << j << "\n";
}

for (double j: ans){
cout << "sin(2theta) = " << j << "\n";
}

return 0;
}
```

```
$ ./output
Provide n_vals:
10
Theta = 0
Theta = 0.349066
Theta = 0.698132
Theta = 1.0472
Theta = 1.39626
Theta = 1.74533
Theta = 2.0944
Theta = 2.44346
Theta = 2.79253
Theta = 3.14159
sin(2theta) = 0
sin(2theta) = 0.642788
sin(2theta) = 0.984808
sin(2theta) = 0.866025
sin(2theta) = 0.34202
sin(2theta) = -0.34202
sin(2theta) = -0.866025
sin(2theta) = -0.984808
sin(2theta) = -0.642788
sin(2theta) = -2.44929e-16
```

PLOTTING DATA

- Reading/writing data basics
- Combining C++ with Python

Reading/writing data

- C++ provides some basic classes for reading/writing data

```
#include <ofstream> // : Stream class to write on files
```
```
#include <ifstream> // : Stream class to read from files
```
```
#include <fstream> // : Stream class to both read and write
from/to files.
```
```
ofstream myfile;
```
```
myfile.open ("example.txt");
```
```
myfile << "Writing this to a file.\n";
```
```
myfile.close();
```

Reading/writing data example

```
#include <iostream>
#include <cmath>
#include <vector>
#include <fstream>

using namespace std;

int main()
{
int k;
cout << "Input n_vals:";
cin >> k;

vector<double> thetas(k);
vector<double> ans;

for (int i = 0; i < k; ++i)
{
thetas.at(i) = i * (1./(k-1)) * M_PI;
}
for (int i = 0; i < k; ++i)
{
ans.push_back(sin(2. * thetas.at(i)));
}
```

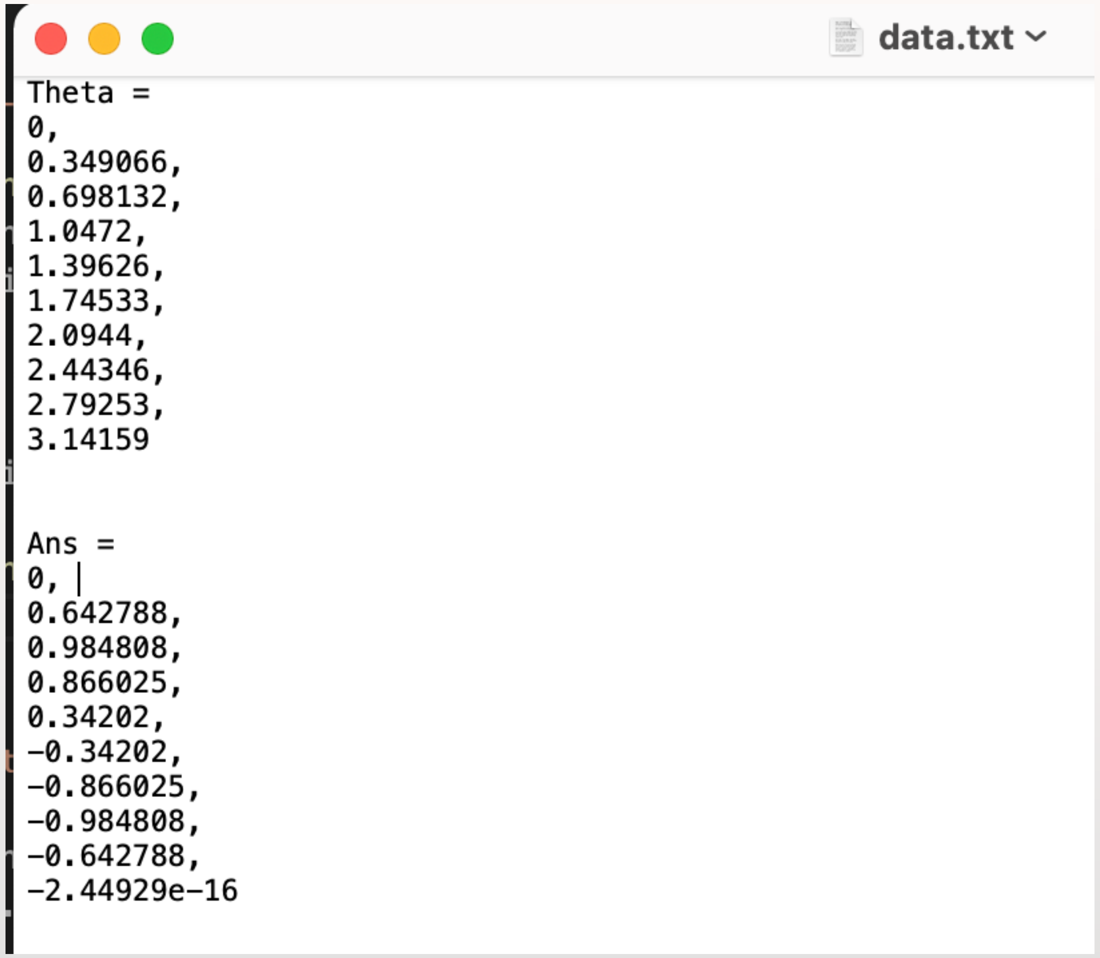
LIV.INNO

Reading/writing data

```cpp
ofstream myfile;
myfile.open ("data.txt");
myfile << "Theta = " << "\n";

for (double j: thetas)
{if (j != thetas.back()){myfile << j << ", " << "\n";}
else{myfile << j << "\n";}}


myfile << ""<< "\n" << "\n";

myfile << "Ans = " << "\n";

for (double j: ans)
{if (j != ans.back()){myfile << j << ", " << "\n";}
else{myfile << j << "\n";}}

myfile << "" << "\n";
myfile.close();

return 0;
}
```

LIV.INNO

Reading/writing data



```
Theta =
0,
0.349066,
0.698132,
1.0472,
1.39626,
1.74533,
2.0944,
2.44346,
2.79253,
3.14159


Ans =
0,
0.642788,
0.984808,
0.866025,
0.34202,
-0.34202,
-0.866025,
-0.984808,
-0.642788,
-2.44929e-16
```

Reading/writing data

- Alternatively, save to a python script…
- Create a plotting code plot.py

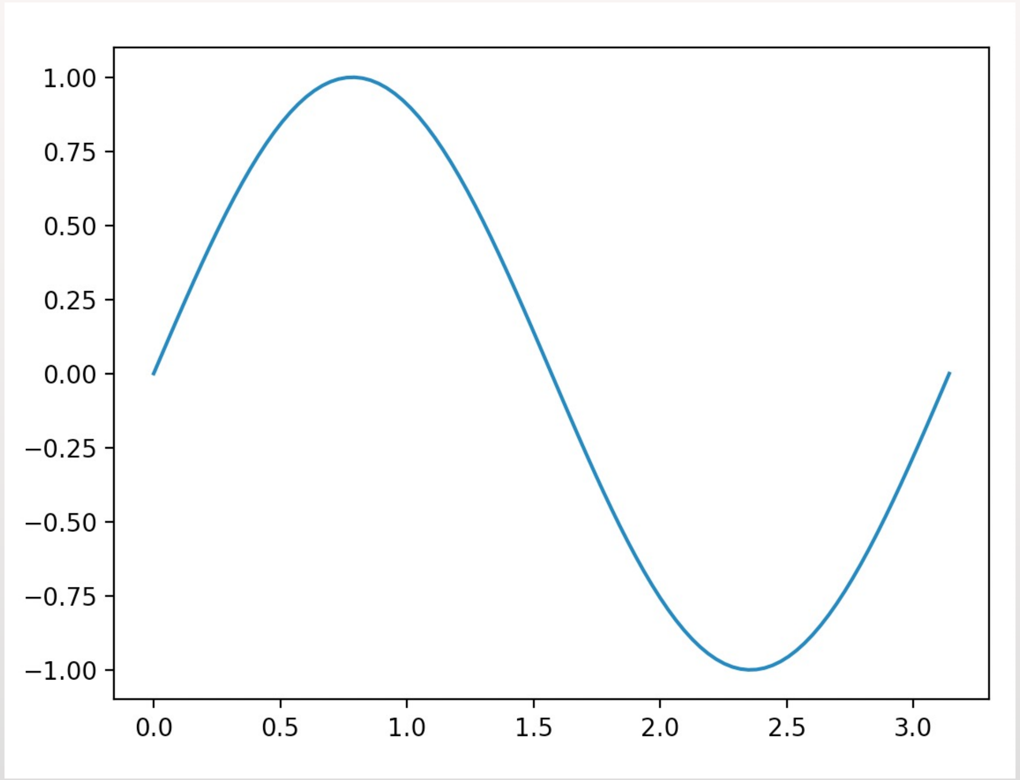
```
import matplotlib.pyplot as plt
from data import *

plt.figure()

plt.plot(theta, ans)

plt.show()
```

```
ofstream myfile;
myfile.open ("data.py");

myfile << "import numpy as np" << "\n” << "\n";
myfile << "theta = np.array((" << "\n";

for (double j: thetas){
if (j != thetas.back()){
myfile << j << ", " << "\n";
}
else{
myfile << j << "\n";
}
}
myfile << "))"<< "\n" << "\n";

myfile << "ans = np.array((" << "\n";

for (double j: ans){
if (j != ans.back()){
myfile << j << ", " << "\n";
}
else{
myfile << j << "\n";}
}
myfile << "))" << "\n";

myfile.close();
return 0;
}
```

Reading/writing data

- Running this in the command line…

```
$ g++ -std=c++11 -o run lesson_script.cpp

(base) alexhill at Alexs-MacBook-Air in
~/Documents/UOL/Teaching/C++_Workshops/Worksho
ps/WS3/scripts
$ ./run
Input n_vals:100

(base) alexhill at Alexs-MacBook-Air in
~/Documents/UOL/Teaching/C++_Workshops/Worksho
ps/WS3/scripts
$ ipython plot.py &
```

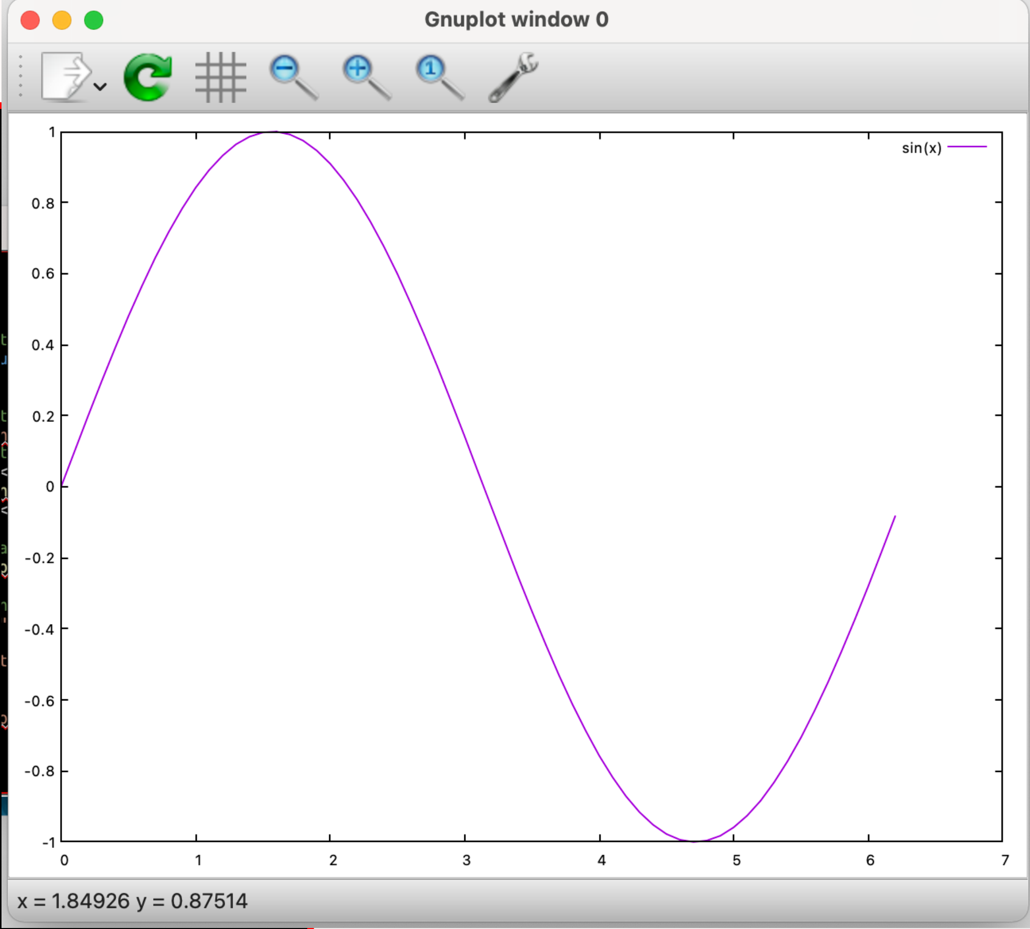


Caveats

- This isn’t the most efficient way of saving data, we want to work with binary files for that

- This requires the use of python

```cpp
#include <iostream>
#include <fstream>
#include <cmath>
using namespace std;
// Function to generate sin(x) for a given x value
double sinFunction(double x) {
return sin(x);}
int main() {
// Create a file to store the data for the plot
ofstream dataFile("sin_data.txt");
// Generate data points for sin(x) and write them to the file
for (double x = 0; x <= 2 * M_PI; x += 0.1) {
double y = sinFunction(x);
dataFile << x << " " << y << "\n";}
dataFile.close();
// Use Gnuplot to create a plot
FILE *gnuplotPipe = popen("gnuplot -persist", "w");
if (gnuplotPipe) {
// Plot sin(x) from the data file
fprintf(gnuplotPipe, "plot 'sin_data.txt' with lines title 'sin(x)'\n");
fflush(gnuplotPipe);
cout << "Press Enter to exit..." << endl;
cin.get();
pclose(gnuplotPipe);} else {
cerr << "Error: Gnuplot not found. Please install Gnuplot to run this script."
<< endl;}
return 0;}
```



LIV.INNO

Challenge six: combining what we've learned today (`const, &, *`)

- Create a function called func() that takes in a vector, and computes:
  - $$f(x) = \begin{cases} e^{-1/x^2} & x \neq 0 \\ 0 & x = 0 \end{cases}$$
- Create a vector with a range -10 to 10 inside main(), and pass it into func()
- Save the input and output to a file 'data.py'. Bonus points if the file writing is done inside a function called write_out(string filename, vector<int>& vect)
- Plot the input and output using a separate python file, 'plot.py'
- Compile, run, and plot this all in the command line

Monte Carlo Methods

- Generating random numbers in C++
- Basics of Monte Carlo methods

```cpp
#include<iostream>
#include<cstdlib>
using namespace std;

int main(){

// Providing a seed value
srand(time(NULL));

// Loop to get 5 random numbers
for(int i=1; i<=5; i++){
// Retrieve a random number between 100
and 200
// Offset = 100
// Range = 101
int random = 100 + (rand() % 101);

// Print the random number
cout<<random<<endl;
}

return 0;
}
```

Seed for random number generator

Outputs current calendar time

Modulo: returns the remainder

Returns an integer between 1 and RAND_MAX

Monte Carlo Basics

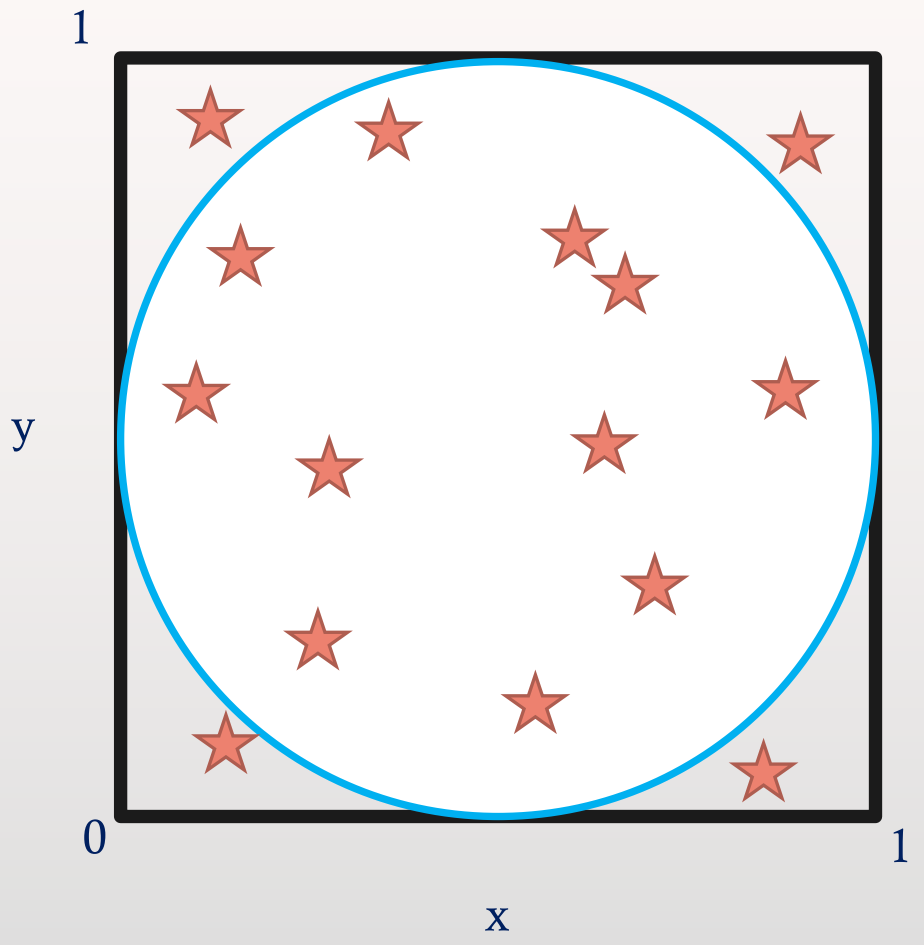
- Monte Carlo methods are a class of computational algorithms that use random sampling to obtain results
- They are often when precise, analytic solutions are impossible
- MC methods are widely used in mathematics and physics
- General idea is to approximate things using samples, e.g. integration, expectations of probabilities

Example: area of a circle

- $0.5^2 = x^2 + y^2$
- Draw random numbers between x = (0,1) and y = (0,1)
- Compute the fraction that satisfies

$$0.5^2 \geq x^2 + y^2$$

- Area of circle = area of square * fraction for $n \to \infty$



Thanks!