October 2024

Introduction to C++: Workshop Three

Dr. Alexander Hill

a.d.hill@liverpool.ac.uk

LIV.INNO

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

```cpp
#include <cmath>
--||--
int main() {
int double start = 0;
double end = M_PI;
double step = M_PI/4;

// Create the evenly spaced array between 0 and pi
vector<double> x_values;
for (double x = start; x <= end; x +=step) {
x_values.push_back(x);
}

// Print the evenly spaced array just to check
cout << "Evenly spaced array between 0 and pi in fractions of pi:" << endl;
for (double x : x_values) {
cout << x << " ";
}
cout << endl;
cout << endl;

// Array to store results of sin(2x)
vector<double> sin_values;
for (double x : x_values) {
sin_values.push_back(sin_2x(x));
}
--||--
```

Nice use of the for loop

Jak

Evenly spaced array between 0 and pi in fractions of pi:
0 0.785398 1.5708 2.35619 3.14159

sin(2(0.0000)) = 0.0000
sin(2(0.7854)) = 1.0000
sin(2(1.5708)) = 0.0000
sin(2(2.3562)) = -1.0000
sin(2(3.1416)) = -0.0000

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

double sin_2x(double theta);

int main() {
vector<double> angles = {0, M_PI/4, M_PI/2, 3*(M_PI/4), M_PI};

cout << "Angles and sin(2x) values:\n";
cout << '\n';

for (double x: angles) {
cout << "Angle: " << x << '\n';
cout << "sin(2x): " << sin_2x(2*x) << '\n';
cout << '\n';
}

return 0;
}

double sin_2x(double theta){
return sin(2*theta);
}
```

Works nicely, but maybe not so easily adaptable

Liam

```
$ ./run
Angles and sin(2x) values:

Angle: 0
sin(2x): 0

Angle: 0.785398
sin(2x): 1.22465e-16

Angle: 1.5708
sin(2x): -2.44929e-16

Angle: 2.35619
sin(2x): 3.67394e-16

Angle: 3.14159
sin(2x): -4.89859e-16
```

Not an array, just printing the values

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

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

int main() {

vector<double> PiVector = {};
vector<double> Sin2piVector = {};

for (double i = 0; i <= 10; ++i) {
PiVector.push_back((i/10)*M_PI);
}

for (int i = 0; i <= (PiVector.size() - 1); ++i) {
Sin2piVector.push_back(sin_2x(PiVector.at(i)));
}

for (int i = 0; i <= (PiVector.size() - 1); ++i) {
cout << "Pi = " << PiVector.at(i)<< " : Sin2Pi = " << Sin2piVector.at(i) << endl;
}
return 0;
}
```

Better practice to avoid repetition if possible, i.e. int n = 10

Good use of vector functions

```
$ ./run
Pi = 0 : Sin2Pi = 0
Pi = 0.314159 : Sin2Pi = 0.587785
Pi = 0.628319 : Sin2Pi = 0.951057
Pi = 0.942478 : Sin2Pi = 0.951057
Pi = 1.25664 : Sin2Pi = 0.587785
Pi = 1.5708 : Sin2Pi = 1.22465e-16
Pi = 1.88496 : Sin2Pi = -0.587785
Pi = 2.19911 : Sin2Pi = -0.951057
Pi = 2.51327 : Sin2Pi = -0.951057
Pi = 2.82743 : Sin2Pi = -0.587785
Pi = 3.14159 : Sin2Pi = -2.44929e-16
```

Rosie

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

double sin2x(double x);
int a,i ;
double pi;

int main() {
a = 10;
pi = M_PI;
std:: vector<double> values(a);
for(i=0; i<a;i++){
values[i] = i*pi /(a-1);
}
std:: vector<double>op(a);
for (i=0; i<a;i++){
op[i] = sin2x(values[i]);
}
for(int i=0; i<a;i++){
std::cout <<"x :"<<values[i]<<" // sin(2x) = "<< op[i] <<std::endl;
}

return 0;
}

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

Declaration outside main!

std:: needed here as vector is defined within the std namespace

```
$ ./run
x :0 // sin(2x) = 0
x :0.349066 // sin(2x) = 0.642788
x :0.698132 // sin(2x) = 0.984808
x :1.0472 // sin(2x) = 0.866025
x :1.39626 // sin(2x) = 0.34202
x :1.74533 // sin(2x) = -0.34202
x :2.0944 // sin(2x) = -0.866025
x :2.44346 // sin(2x) = -0.984808
x :2.79253 // sin(2x) = -0.642788
x :3.14159 // sin(2x) = -2.44929e-16
```

Archie

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

vector<double> starting_vect(int num_of_vals);
vector<double> pi_vect(vector<double>starting_vector);
vector<double> sin_vect(vector<double>pi_vector);
int select_vals();
void print_vals(vector<double> sin_vector);


int main() {
int num_of_vals = select_vals();
vector<double> v1 = starting_vect(num_of_vals);
vector<double> v2 = pi_vect(v1);
vector<double> v3 = sin_vect(v2);
print_vals(v1);
print_vals(v2);
print_vals(v3);
cout << endl;
return 0;
}
// More below
```

```
$ ./run
enter number of sin values: 4
values for starting_vect: 4 3 2 1
values for pi_vect: 0.785398 1.0472 1.5708 3.14159
values for sin2x: 1 0.866025 1.22465e-16 -2.44929e-16
```

Really nice use of main()

Archie

```
vector<double> starting_vect(int num_of_vals) {
vector<double> vector1;
for (int i = 0; i < num_of_vals; i++) {
vector1.push_back(num_of_vals - i);
}
return vector1;
}


vector<double> pi_vect(vector<double>starting_vector){
vector<double> pi_vector;
for(double k: starting_vector){
pi_vector.push_back(M_PI/k);
}
return pi_vector;
}
```

```
$ ./run
enter number of sin values: 4
values for starting_vect: 4 3 2 1
values for pi_vect: 0.785398 1.0472 1.5708 3.14159
values for sin2x: 1 0.866025 1.22465e-16 -2.44929e-16
```

pi_vector not evenly space spaced, goes pi/4, pi/3, pi/2, pi/1 etc

LIV.INNO

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

using namespace std;

double sin_2x(double z)
{
const double t = 1e-10;
double y;
y = sin(z);
if(abs(y)<t) y=0.0;
return y;
}

int main()
{
double theta, x[13], d[13], ans[13];
for(int i=0;i<13;i++)
{
x[i]=(i*M_PI/6);
theta=2*x[i];
d[i]=(theta*180/M_PI);//converting to degree
ans[i]=sin_2x(theta);
cout<<"The value of sin("<<d[i]<<"') = "<<ans[i]<<endl;
}
return 0;
}
```

Shirsendu

Good use of const, setting sensitivity limit manually

Personally would have sin_2x(double x). Also – you are import 2x really, which is potentially confusing

13 and 6 work, but not adaptable and open to human error

```
$ ./run
The value of sin(0') = 0
The value of sin(60') = 0.866025
The value of sin(120') = 0.866025
The value of sin(180') = 0
The value of sin(240') = -0.866025
The value of sin(300') = -0.866025
The value of sin(360') = 0
The value of sin(420') = 0.866025
The value of sin(480') = 0.866025
The value of sin(540') = 0
The value of sin(600') = -0.866025
The value of sin(660') = -0.866025
The value of sin(720') = 0
```

```cpp
// Online C++ compiler to run C++ program online
#include <iostream>
#include <iomanip>
using namespace std;
#include <cmath>
#include <vector>
// PART 1: To generate a evenly space vector between 0 to PI
double sin_2(double ang);
int main() {
const double PI = 3.141592653589793;
double x,h, thetao, theta;
int n=10; // Total entries between 0 to PI
thetao=0; // Inital angle in radian
h=PI/n; // step size
vector<double>angle={};
vector<double>sin_2x={};
cout << "Anlges"<<'\n';
for (int i=0; i<n+1; i++) {
angle. push_back(thetao);
cout << setprecision(3);
cout<< angle[i] << '\n'; // display vector elements
theta=thetao+h;
thetao=theta;
}
// Part 2: Generate a function which calculate Sin2x and save entries in a new vector
cout << "Sin2x"<< '\n';
for ( int j=0;j<n+1; j++){
sin_2x. push_back(sin_2(angle[j]));
cout << setprecision(3);
cout <<sin_2x[j] << '\n';
}
return 0;
}
double sin_2(double ang) {
return 2*cos(ang)*sin(ang);
}
```

```
$ ./run
Anlges
0
0.314
0.628
0.942
1.26
1.57
1.88
2.2
2.51
2.83
3.14
Sin2x
0
0.588
0.951
0.951
0.588
1.22e-16
-0.588
-0.951
-0.951
-0.588
-2.45e-16
```

Good use of const, but cleaner to use the M_PI in built into <cmath>

Shaoib

LIV.INNO

```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
- https://youtu.be/LfaMVIDaQ24?t=32542 – Harvard CS50 (Memory)

POINTERS

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

Memory Address

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

Memory Address

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

```
#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

```
#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
```

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
```

LIV.INNO

Pointers

```cpp
#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!

```
#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

```
#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;
cout << 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

```
#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

```
#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
```

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

LIV.INNO

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
```

LIV.INNO

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

```
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 have 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
a = 0x16d8f73b0
sizeof(a) = 8
sizeof(a[0]) = 4
```

Passing arrays into functions

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


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
```

Challenge Six

- 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

```
#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

```
#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

```cpp
#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
```

LIV.INNO

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();
```

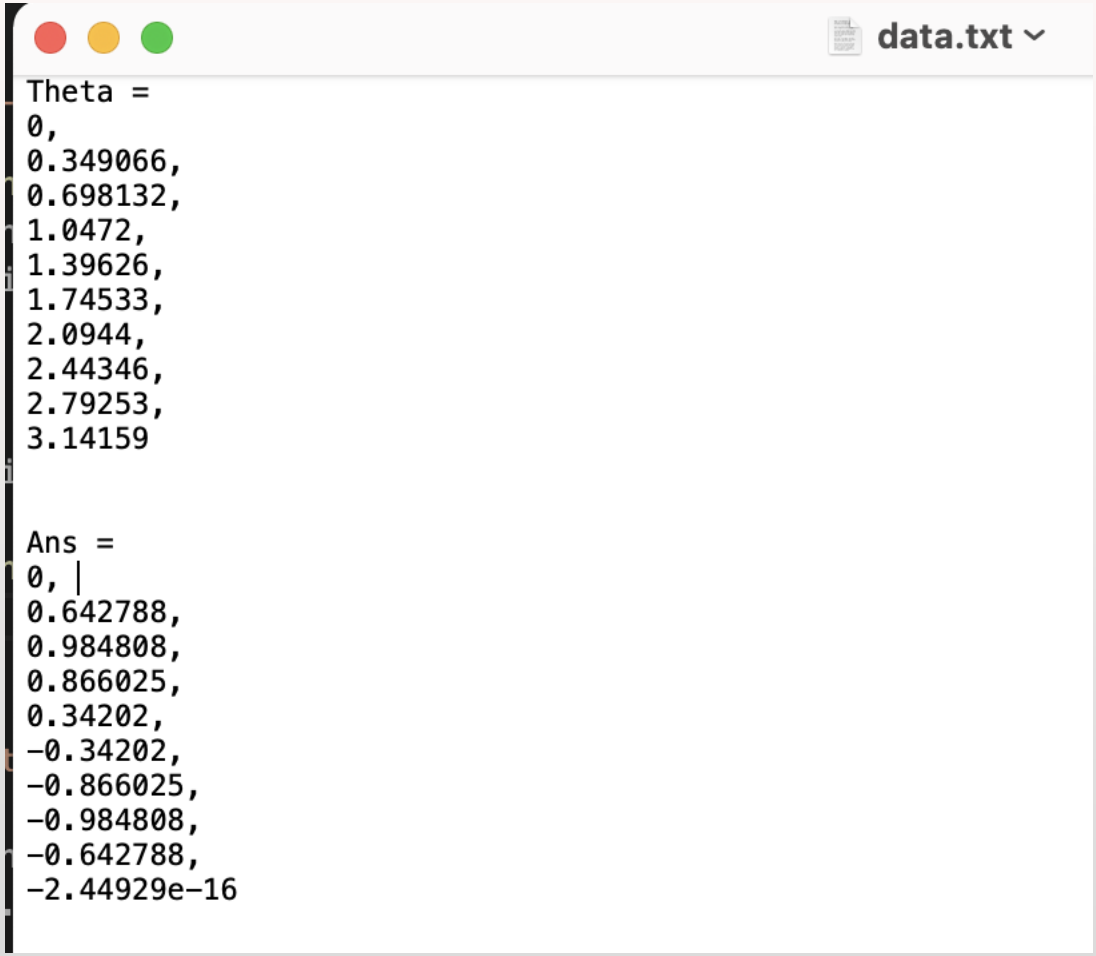
Reading/writing data example

```
#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)));
}
```

Reading/writing data

```
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;
}
```

Reading/writing data



Reading/writing data

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

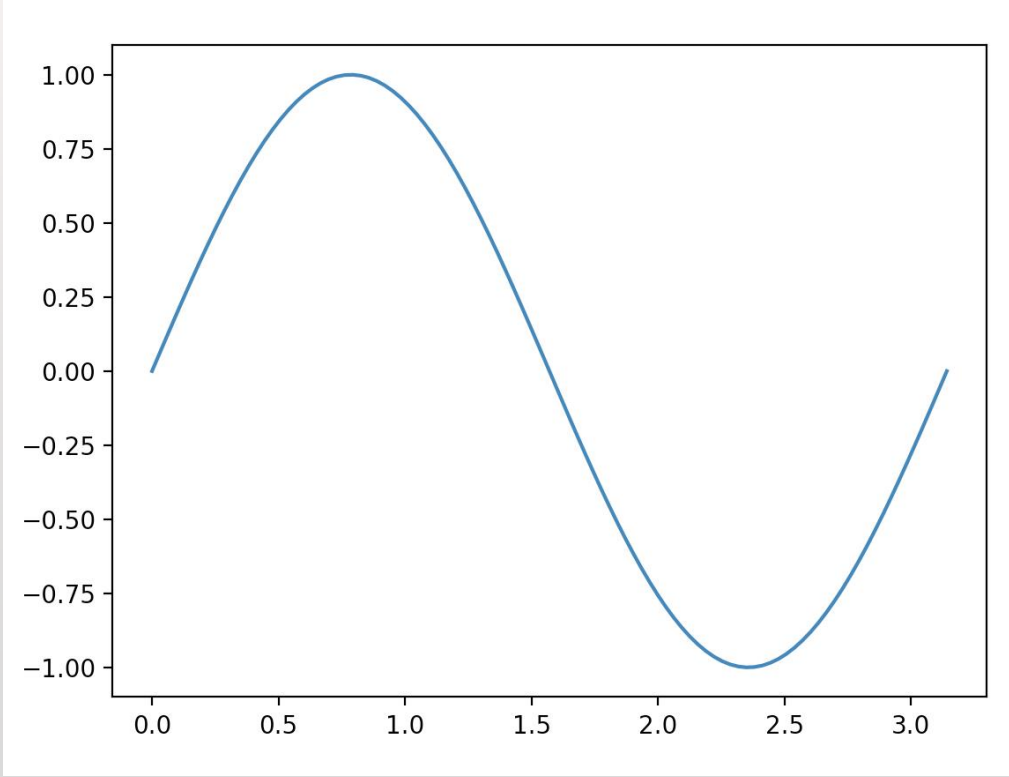
```
import matplotlib.pyplot as plt
from data import *

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…

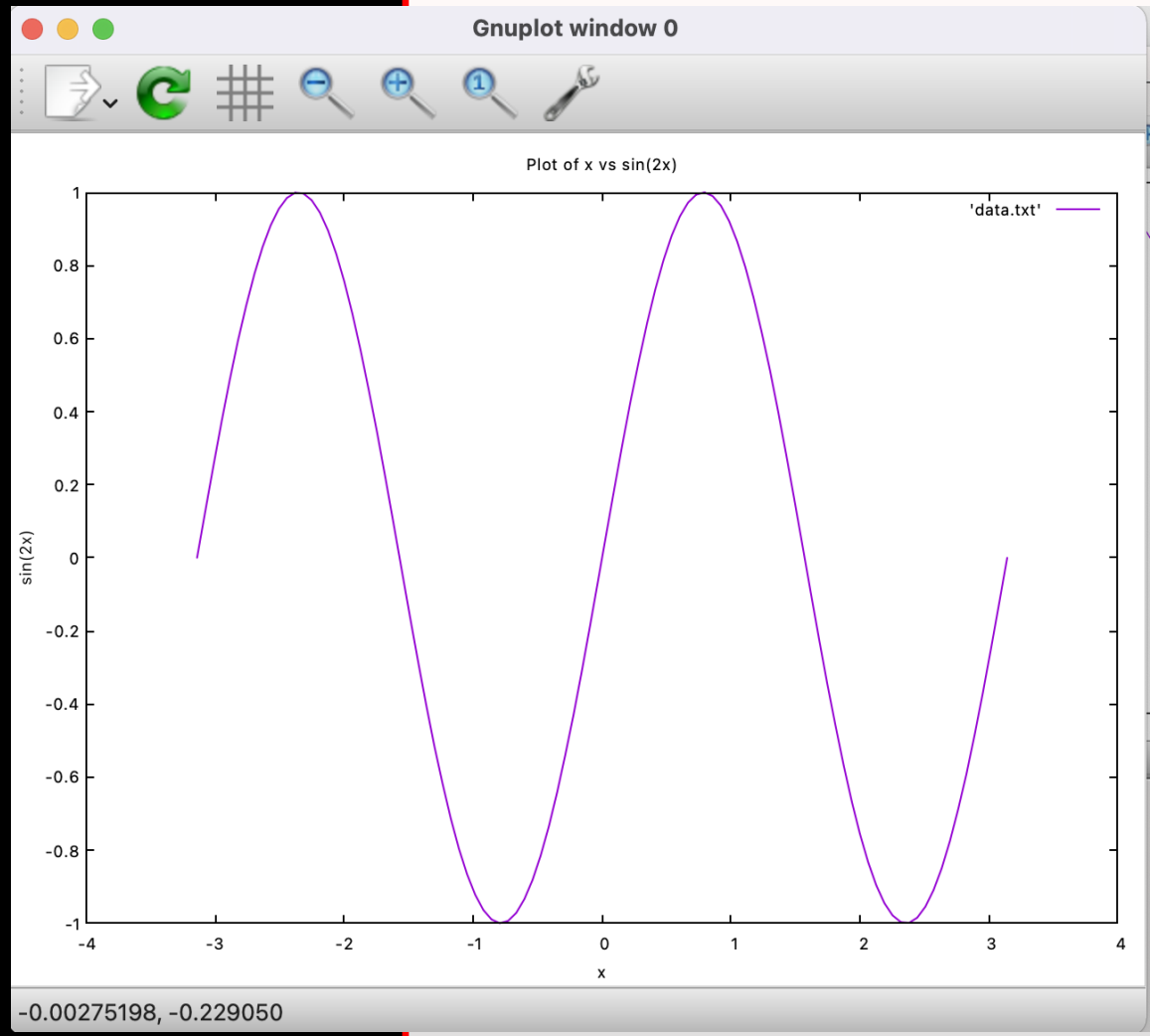
```
$ g++ -std=c++11 -o run lesson_script.cpp

(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
std::vector<double> x_values(num_points);
std::vector<double> y_values(num_points);

// Generate x values and compute y = sin(2 * x)
for (int i = 0; i < num_points; ++i) {
x_values[i] = x_min + i * step;
y_values[i] = sin(2 * x_values[i]);
}

// Save to a file for gnuplot
std::ofstream outfile("data.txt");
for (int i = 0; i < num_points; ++i) {
outfile << x_values[i] << " " << y_values[i] << "\n";
}
outfile.close();

// Plotting with gnuplot
std::ofstream gp("plot.gp");
gp << "set title 'Plot of x vs sin(2x)'\n";
gp << "set xlabel 'x'\n";
gp << "set ylabel 'sin(2x)'\n";
gp << "plot 'data.txt' with lines\n";
gp << "pause -1\n";
gp.close();

// Run gnuplot script
system("gnuplot -persist plot.gp");

return 0;
}
```



LIV.INNO

Challenge Seven: combining what we've learned today (const, &, *) (Homework)

- 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

Seed for random number generator

```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;
}
```

Outputs current calendar time

Modulo: returns the remainder

Returns an integer between 1 and RAND_MAX

Monte Carlo Basics

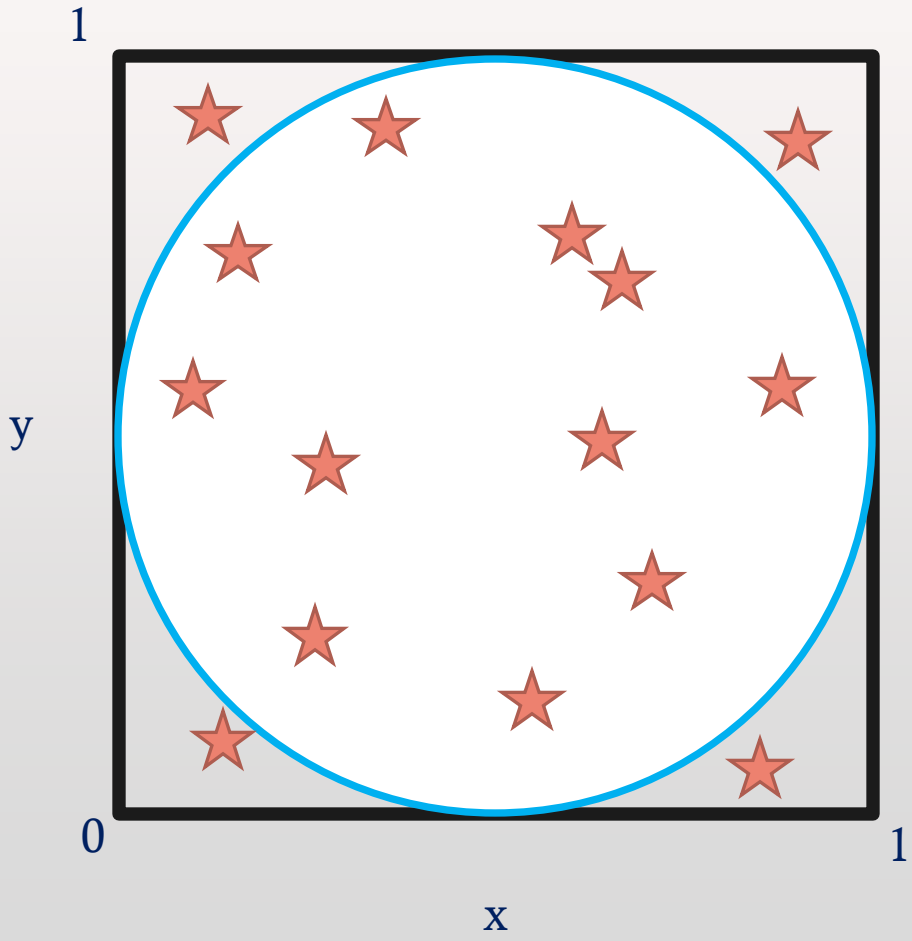
- Monte Carlo methods are a class of computational algorithms that use random sampling to obtain results
- They are often when precise, analytic solutions are impossible
- MC methods are widely used in mathematics and physics
- General idea is to approximate things using samples, e.g. integration, expectations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

LIV.INNO