Module 1:
Inbuilt Data Types
&
Simple functions

In this module we will cover:

- **In-built data types**
- **Operations using in-built data types**
- **Packaging operations inside simple functions**

Aims of this module

In this module we introduce the in-built types (int, float, double, char and bool) with which most students will already be familiar.

We emphasise that it is important to declare the types and names of variable needed in your piece of computer code.

We then introduce the basic operations one can do with in-built data types. This includes arithmetic operations as well as some in-built functions.

Finally we show how repetitive or tedious bits of code can be packaged inside simple user written functions.

1.1 In-built Data Types

Calculations/manipulations are probably at the core of what most people want to do in a scientific computing context

In most computer languages you will find the concept of "data types"

These are the allowed "types" of data with which calculations/manipulations can be performed.

To do these calculations we need our language to allow:

"real variables" for **E**, ϕ

"integer variables" for **S**, **n**

Example:

$$E = E_0 (1 - \cos\phi)^{-1} / 2$$

to calculate energy, E, of an electron scattered through angle ϕ

..or..

$$S = \Sigma n^2$$

to sum a series of integer values

Unlike pen and paper calculations, you are required to DECLARE these variable types and names to the program before you can use them in any program calculations.

```
// A fragment of a C++ program
// where some variables have their
// data types declared, and are
// named


float E ;
float phi ;

int S ;
int n ;
```

Why is this ?

The way a computer stores and does calculations is very different for each data type - so it needs to know in advance in order to:

- allocate the right sort of memory

- invoke the right instructions for a calculation

Also it stops you mis-typing variable names

```
//A fragment of a C++ program
//where some variables have their
//data types declared, and are named


float E ;
float phi ;

int S ;
int n ;
```

This is how you write comments in a C++ program.
Any text following //

Note the syntax:

float and **int** are KEY words in C. They indicate in-built data types.

-after this you write any variable name you want

-terminate with **;**

Variable names can be single letters
or
almost any arbitrary text string

```
// More variable types and ways of naming them

//some integer variables

int sumOfSeries ;
int n ;


// some real variables

float electronEnergy ;
float scattering_angle ;

double electronCharge ;


// Some single character "variables"

char diskDriveLettter ;
char inputCharacter ;

// A boolean variable:

bool isValid ;
```

Private Study:

Familiarise yourself with these in-built data types from the course book

1.2 Simple Operations on in-built types

As example consider managing a bank account. We want do do simple operations like transfer money in and out of it

```
// Lets declare variables for two bank accounts

float myAccount ;
float taxmansAccount ;

// Variables to help with transaction
float taxRebate ;

// Set value of rebate
taxRebate = 500.27 ;

// Adjust accounts

myAccount      = myAccount + taxRebate ;
taxmansAccount = taxmansAccount - taxRebate ;
```

Note the operations of `=, +, -`

Note the common operation of a = a+b

```
myAccount      = myAccount + taxRebate ;
taxmansAccount = taxmansAccount - taxRebate ;
```

Simpler way is to use += and -+ operators

```
myAccount      +=  taxRebate ;
taxmansAccount -=  taxRebate ;
```

As second example consider the electron energy formula used previously

$$E = E_0 (1 - \cos\phi)^{-1} / 2$$

We want to calculate **E** for an input value of ϕ

```
// Declare variables

float electronEnergy ;
float Ezero ;
float phiScatter ;

// Set E0 in keV
Ezero = 560.0 ;

// Scattering angle in radians
phiScatter = 1.5 ;

// Do calculation

electronEnergy
     = Ezero / ( 1. - cos(phiScatter) ) / 2. ;
```

Note that line-breaks are irrelevant. You can format the text however you like

Note the divide operation
/

Note the "function" **cos()** which is built into C++. More of these later

The boolean type:

```
// An example of a boolean operation

bool isValid ;


// set it to be true or false

isValid = true ;

isValid = false ;
```

The char type:

We will not use char variables much (or at all). We will use strings instead which come later.

If you ever need to handle C style character strings directly then
(i) my commiserations
(ii) consult the C reference books

```
// More in-built arithmetic operations
int a,b,c ;
float d,e,f ;

// simple integer and float operations already seen
d = e + f ;
d = e * f ;  // multiply
a = b - c ;
d = e / f ;  // floating point divide
a = b / c ;  // integer divide, truncates  fraction

// New one sets a to remainder after b/c
a = b%c ;

// Compound operations
d *= e  ;   //   means d = d * e
d /= e  ;   //   means d = d / e
a += b  ;
a -= b  ;

// Some unary operators
a++ ;    //   increments a by 1: a += 1
b-- ;    //   decrements by 1
```

Private Study:

Familiarise with these operators

C++ also has a lot of in-built library functions which you will often need to use.

We saw the cosine function cos() earlier.
Here are some more useful ones:

See references for a more complete list.

```
// Declare variables
float x, y, z, theta ;

x = cos(theta) ;
x = sin(theta) ;
x = tan(theta) ;

theta = asin(x) ;
theta = acos(x) ;
theta = atan(x) ;

y = exp(x) ;
y = log(x) ;    // Natural logarithm
y = log10(x)    // log base 10
y = abs(x)  ;   // absolute value of x

z = pow(x,y) ;  // x raised to the power of y
z = sqrt(x) ;
```

Student exercise

Write some code to:

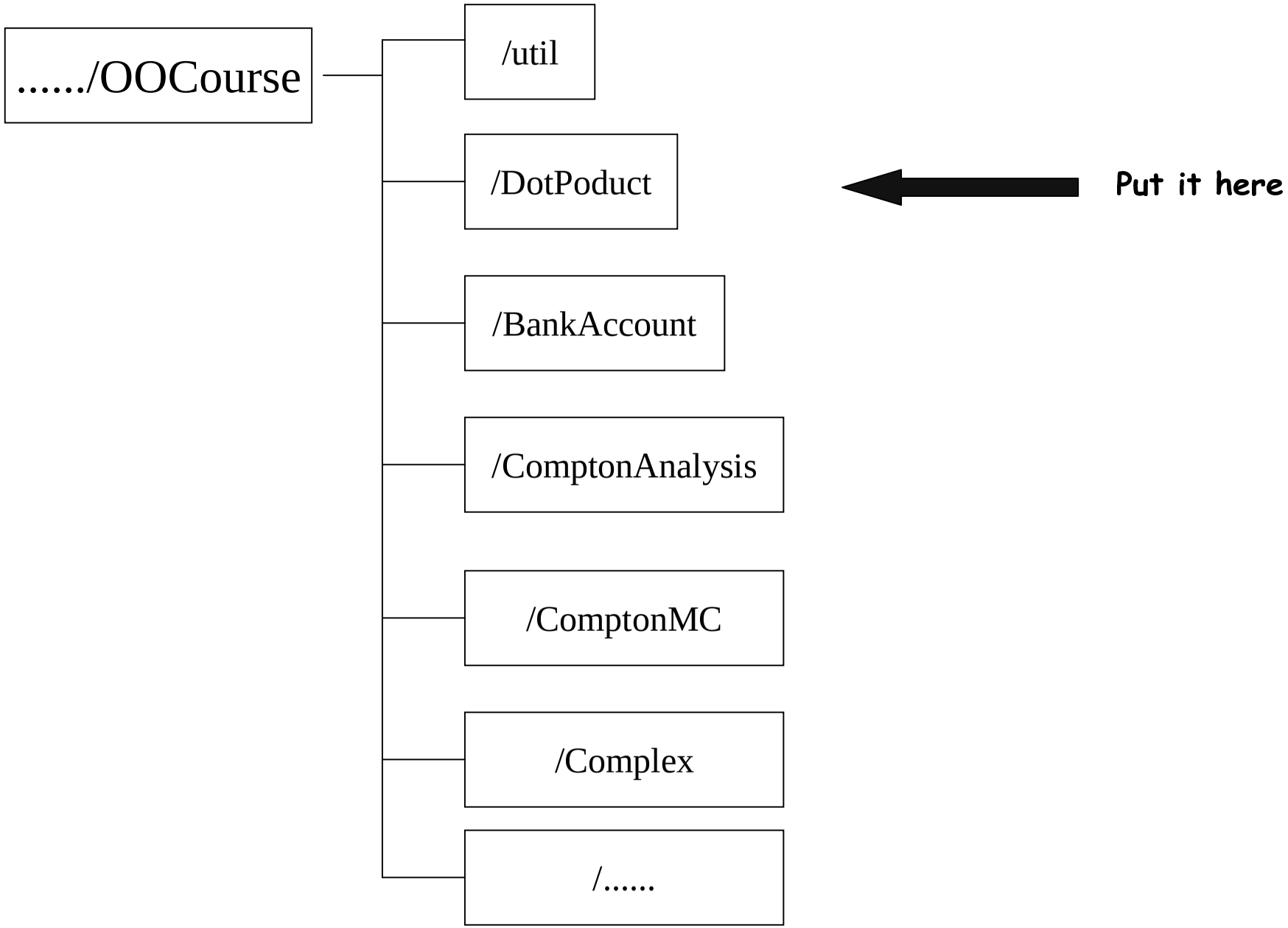
-declare variables to represent two vectors

-find the angle between them

-print out the angle

This is also an opportunity to try out the C++ environment with some simple code.

You should write this in a file with a name like:
`dprod1.cc`
The important thing is for it to end in
`.cc`
but the prepend can be anything you like


...../OOCourse
/util
/DotPoduct
Put it here
/BankAccount
/ComptonAnalysis
/ComptonMC
/Complex
/.....

This is a "hands on ASAP course" - that is the only way to learn a language !

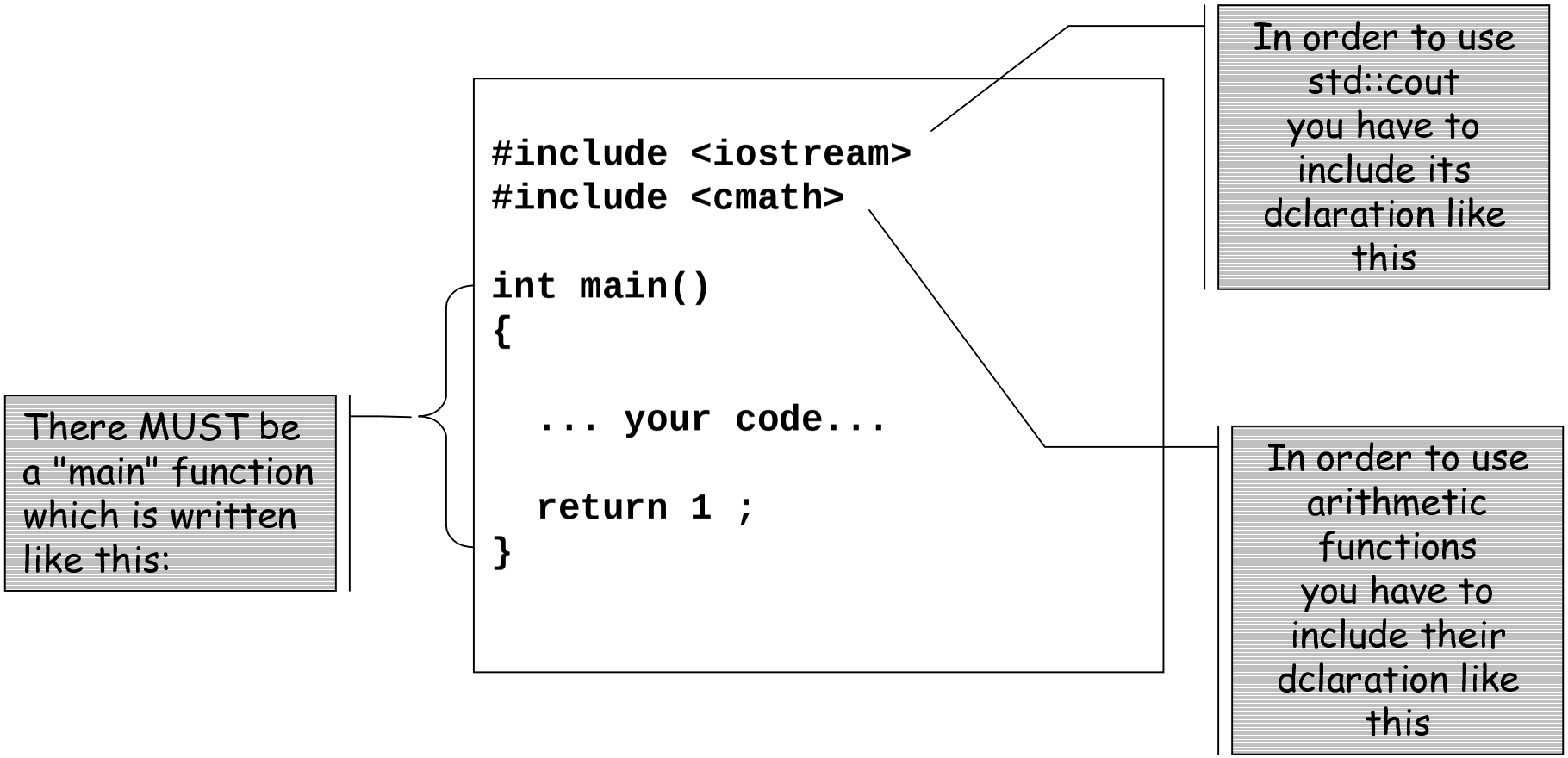
Therefore there are some things you need to know of now which we will only explain fully later

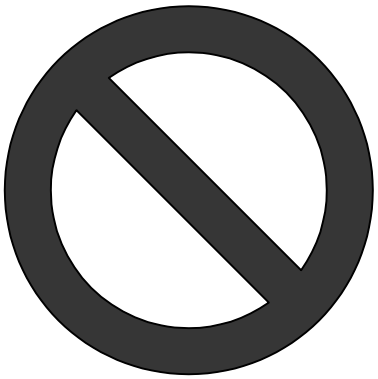
You will need to know
how to print something out
using std::cout

```
float x = 120.37;

std::cout << " The value of x is " << x << std::endl;
```

You also need to know some incantations needed to make the code into a program which can be compiled/linked/run.





Annoyingly C and C++ can be different regarding inbuilt function names

If you use (new way)

#include <cmath>

it should work as I say

If you use (old way)

#include <math.h>

you may get strange results with at least the abs(..) function

```
#include <cmath>

int main()
{

  .......

}
```

1.3 Packaging operations inside simple functions

You can write repetitive and uninteresting bits of code in

Functions

This allows you to encapsulate things you might want to do often in one place.

Also, you often want to modularise the code so its not all in one file (and hence unreadable).

Lets show how you would write a simple "function" to do an
$y = ax^2 + bx + c$ operation

```
float myFunction( float xValue )
{
  // This is a simple function

  float a = 3.0 ;
  float b = 1.5 ;
  float c = 0.7 ;

  float yValue = 0 ;

  yValue = a*xValue*xValue + b*xValue + c ;

  return yValue ;

}
```

This is how you declare a complete function signature

```
float myFunction( float xValue )
{
  // This is a simple function

  float a = 3.0 ;
  float b = 1.5 ;
  float c = 0.7 ;

  float yValue = 0 ;

  yValue = a*xValue*xValue + b*xValue + c ;

  return yValue ;

}
```

This is how you declare the name of the function

```
float myFunction( float xValue )
{
  // This is a simple function

  float a = 3.0 ;
  float b = 1.5 ;
  float c = 0.7 ;

  float yValue = 0 ;

  yValue = a*xValue*xValue + b*xValue + c ;

  return yValue ;

}
```

This is how declare the arguments which it will receive

```
float myFunction( float xValue )
{
  // This is a simple function

  float a = 3.0 ;
  float b = 1.5 ;
  float c = 0.7 ;

  float yValue = 0 ;

  yValue = a*xValue*xValue + b*xValue + c ;

  return yValue ;

}
```

This is how you declare the type of information it is going to return

```
float myFunction( float xValue )
{
  // This is a simple function

  float a = 3.0 ;
  float b = 1.5 ;
  float c = 0.7 ;

  float yValue = 0 ;

  yValue = a*xValue*xValue + b*xValue + c ;

  return yValue ;

}
```

The whole thing must be enclosed by braces {……}

```
float myFunction( float xValue )
{
  // This is a simple function

  float a = 3.0 ;
  float b = 1.5 ;
  float c = 0.7 ;

  float yValue = 0 ;

  yValue = a*xValue*xValue + b*xValue + c ;

  return yValue ;

}
```

```
float myFunction( float xValue )
{
  // This is a simple function

  float a = 3.0 ;
  float b = 1.5 ;
  float c = 0.7 ;

  float yValue = 0 ;

  yValue = a*xValue*xValue + b*xValue + c ;

  return yValue ;

}
```

This is how you return a value from the function to the user (I.e. the person calling it)

Functions can be invoked from anywhere inside another piece of code.

For example we would use the function we just wrote like this from within the main { } block you wrote earlier:

```
...

  // This is a call to the function called
  // "myFunction"

  float x ;
  float y ;

  y = myFunction( x ) ;

...
```

Note: you do not have to use the same variable names to use the function as you used when writing the function. This causes a lot of confusion !!. This will become clear in the exercise.

Functions can also be invoked from within other functions:

```
float myFunction1( ..... )
{

  ... lots of code

  // Call a function from within this function

  float answer ;

  answer = myFunction2( .... ) ;

  ...rest of code

}
```

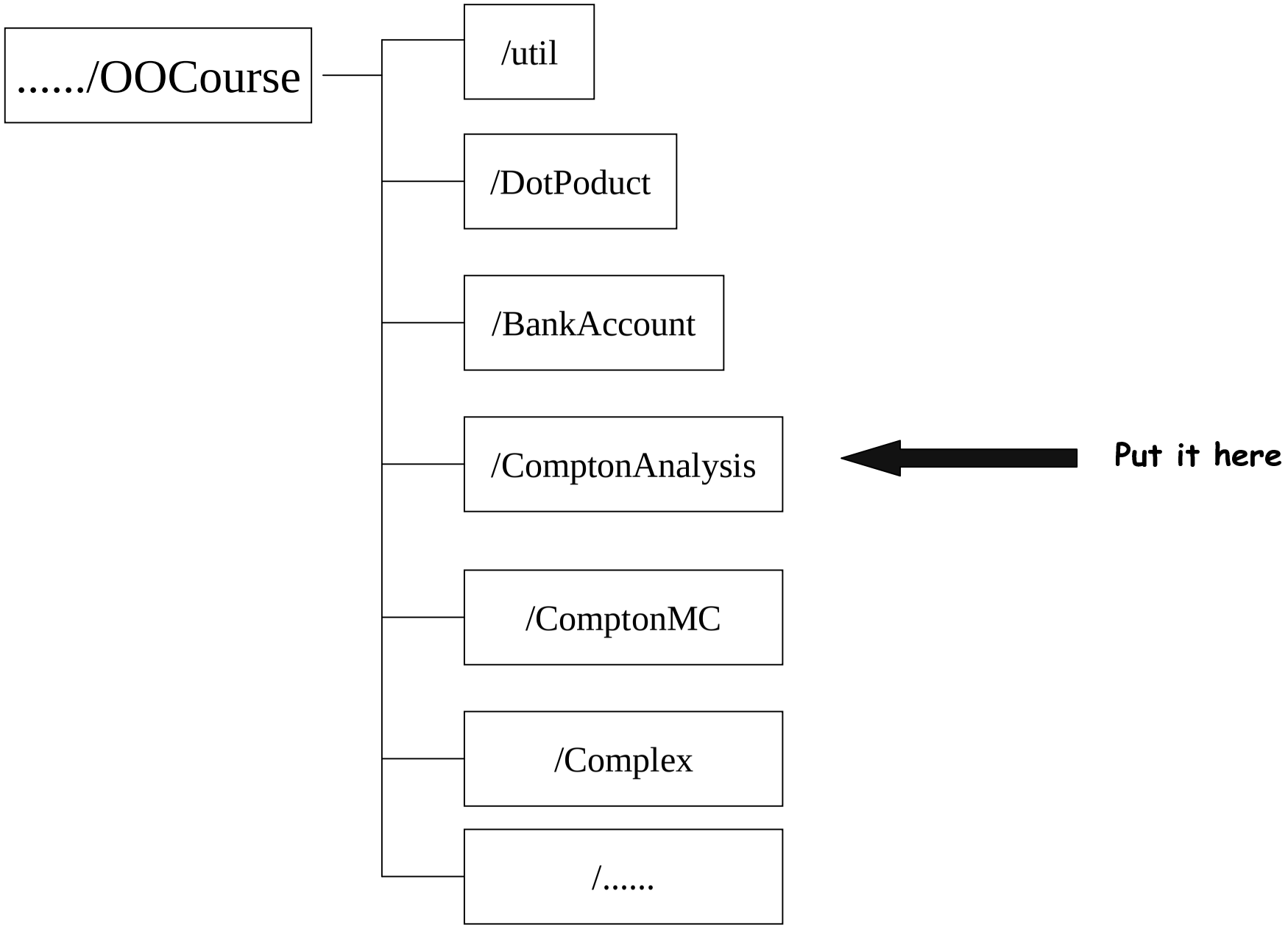
Student exercise

-Write a function to calculate and return the inverse energy (i.e. 1/E) of a Compton scattered electron, given by the relation

$$E = E_0 \left(1 - \cos\phi \right)^{-1} / 2$$

-Write a short `main()` program which uses this function
Use it twice with different sets of input and return variables.

ComptonAnalysis/
invE.cpp


...../OOCourse
/util
/DotPoduct
/BankAccount
/ComptonAnalysis
Put it here
/ComptonMC
/Complex
/.....

This shows the typical structure of a single file containing all the bits you need:

```
#include <.....>


float myFunction( ....)
{
// Code for function in here...
~~~~~~~~
~~~~~~~~
}


int main
{
// Code for main program in here..
// this is where you call myFunction
~~~~~~~~
~~~~~~~~
}
```

the point of using functions

FUNCTIONS - the point of (1)

One “main” file

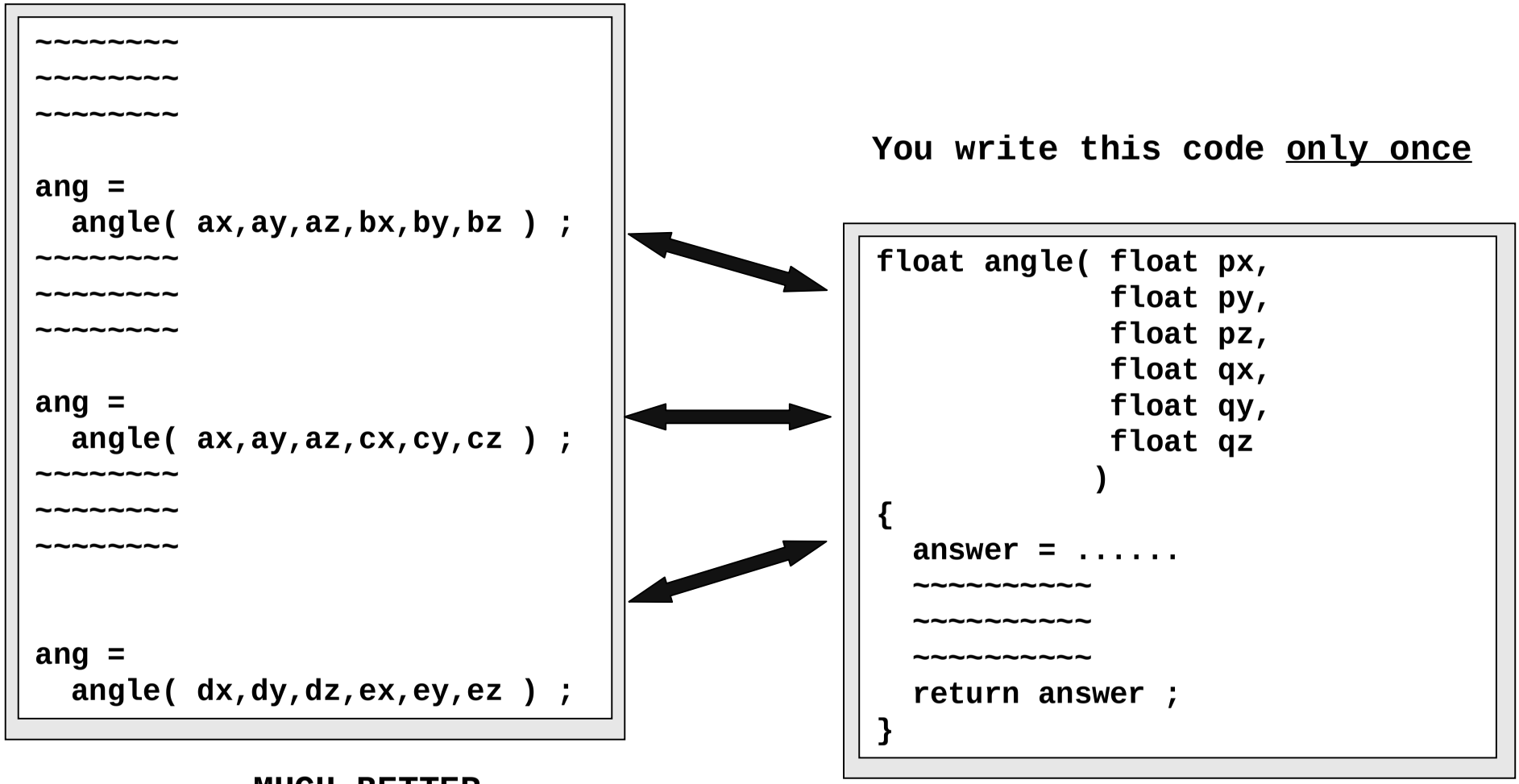
```
~~~~~~~~
~~~~~~~~
~~~~~~~~
dp = ax*bx + ay*by ...
mag = ......
angle = acos( dp / mag ... )
~~~~~~~~
~~~~~~~~
~~~~~~~~
dp = ax*cx + ay*cy ...
mag = ......
angle = acos( dp / mag  ...)
~~~~~~~~
~~~~~~~~
~~~~~~~~
dp = dx*ex + dy*ey ...
mag = ......
angle = acos( dp / mag ... )
```

BAD

⇒ Repeat same long and complicated code many times

⇒ Easy to make mistakes

FUNCTIONS - the point of (2)

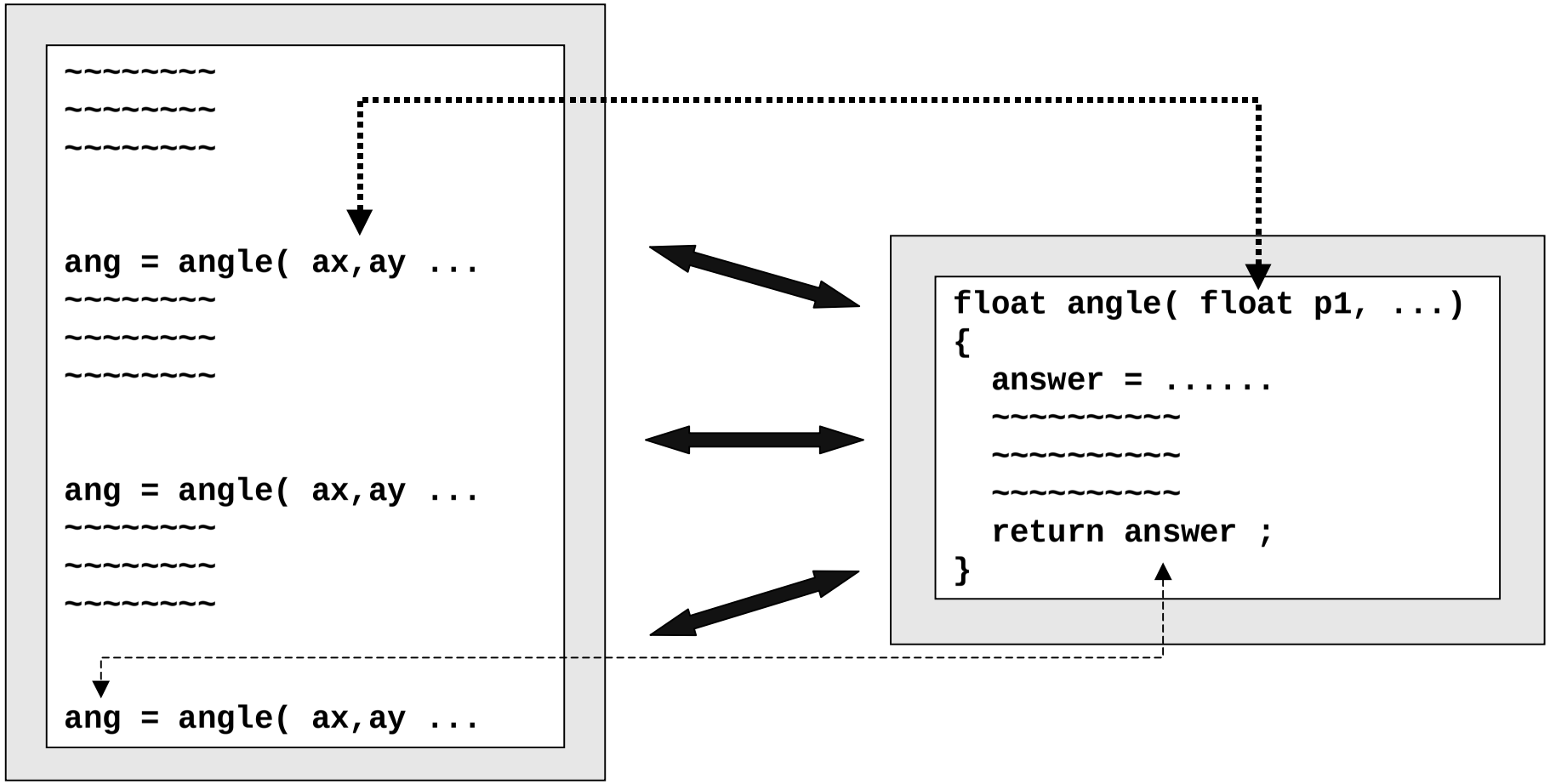


MUCH BETTER

⇒ Write code only once, maintain it in one place only

⇒ Call it from main code as often as you need

FUNCTIONS - the point of (3)

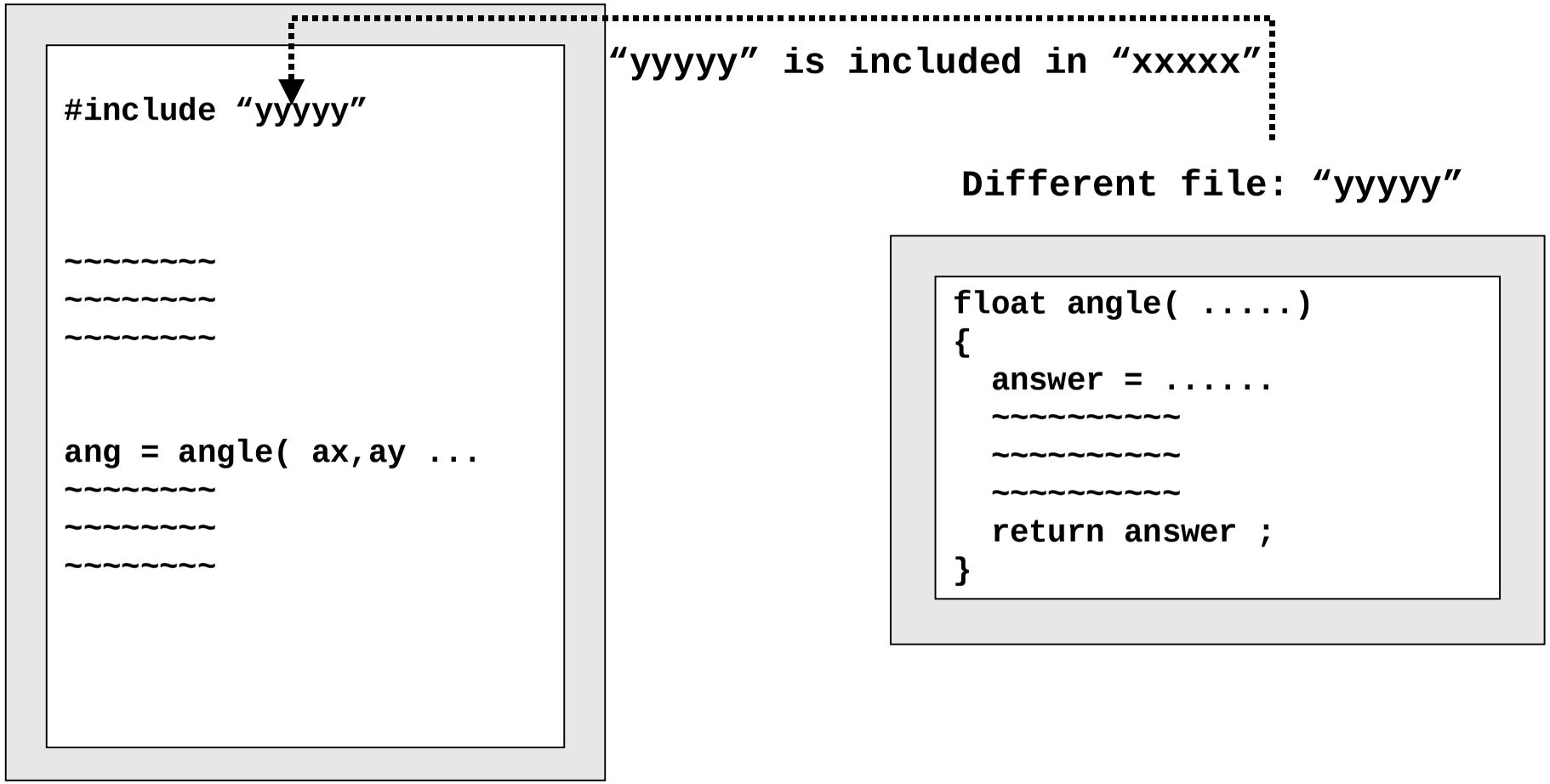


NOTE: NAMES IN FUNCTION ARE “DUMMY” NAMES
It doesn’t matter what they are called in function,
they get replaced with actual variables when called

organisation of functions

FUNCTIONS - organisation of (4)

One “main” file called xxxxx



Function written in separate file and included with “include”
⇒ Means you can include it in different applications later

Summary of Module 1: In-built Data Types & Simple functions

In this module we have covered the following topics. It is assumed that the students will further familiarise themselves with these topics during private study. Questions and problems can be addressed either during the "surgery" time, or students are welcome to contact the course lecturer(s) at any time.

- **In-built data types**
  - **int**
  - **float**
  - **double**
  - **bool**
  - **char**

....contd

·Simple operations on in-built types

arithmetic operators +,-,*,/,=,+=,-=,.. etc..

in-built functions cos, sin .. log,..etc ..

· Packaging operations inside simple functions

declaration of function name

declaration of arguments

return type

body of the function

· "Hands on" basics of how to make a program run

compile link and run

include other files containing functions you need

A last word: Philosophy !

In this course we are showing you

1) How to Program

as well as

2) How to write C/C++

- OO philosophy
- organisation
- modularisation

⇒ all language independent

- technical syntax of C++

⇒ nothing to do with how to write a good program

The two are different !