MATH60082
Support Class
Examples and Solutions

Contents

0.1 Basic Syntax

0.1.1 Simple Calculation

Write a program to calculate the area and circumference of a circle of radius 5.0. Change the code so that the radius may be specified by the user. For input you will need to use the input `cin` command.

For the first part of this question we will need to do some calculations and get access to the value of the mathematical constant pi. But before we think about that, you should think about what the question asks, and therefore what are the following:-

- Libraries/Methods
- Variables/Parameters/Inputs
- Calculations
- Outputs

In this example we are going to need input output and also a mathematical constant, so it would be a good idea to include the cmath library. Programming is best split up into stages and you should try to compile and run you programs as frequently as possible. The idea is simple, if you have only edited one or two lines and your program breaks, you know where the problem is, if you have added 50+ it is much harder to know where you have gone wrong. As such, you should begin by writing an empty program, and compile and run it to check for errors. Your initial program should look like:

```
// include the input output stream library
// so that you can write to the screen
#include <iostream>
// we will also need the cmath library to
// get access to mathematical functions
#include <cmath>
// use the std namespace
using namespace std;

int main()
{
  // do nothing yet
  return 0;
}
```

download

The output should be blank.

Next we think about the variables and parameters and how to input values. At first it is always better to input the values of variables into the program itself to make it quicker whilst you are debugging, so we will assign the variable inside the code first. Now what do we need? The question asks for the calculation of the area and the circumference of a circle, which indicates that we will need

- radius, a real number
- pi, a real number
- area, a real number
- circumference, a real number

Now we can declare these variables inside the main function, allowing them to be assigned values and involved in calculations. To declare we write double as the type of the variable, here you can think of double as a real number, and then list the variable names separated by commas and finished by a semi colon. You should use as descriptive a name as possible, and those variable names can include letters from the alphabet in upper or lower case and numbers (cannot begin with a number) and underscores.

```
double radius,pi,area,circumference;
```

download

Next assign those variables with a value, remember to finish statements with a semi colon. Multiple statements can be written on the same line as long as they are closed off.

```
radius=5.;pi=4.*atan(1.);
```

download

Next we need to do the calculations. The only thing to note here is that there is no operator for powers, only the pow(x,y) function from the math library, so when calculating a square of a number it is easier to just write number*number. The calculations are:

```
area = radius * radius * pi;
circumference = 2. * pi * radius;
```

download

Finally we can output the solution to screen:

```
cout << " A circle with radius " << radius << " has area ";
cout << area << " and circumference " << circumference << "." << endl;
```

download

At this stage your program should look like:

```
// use the input/output libraries and mathematical functions
#include <iostream>
#include <cmath>
// again use the namespace
using namespace std;

int main()
{
  // use descriptive names for variables
  double radius,pi,area,circumference;
  // assign those variables with a value
  radius=5.;pi=4.*atan(1.);
  // now do the calculations
```

```
  area = radius * radius * pi;
  circumference = 2. * pi * radius;
  // output results to screen
  cout << " A circle with radius " << radius << " has area ";
  cout << area << " and circumference " << circumference << "." << endl;
  return 0;
}
```

download

and the output in the terminal should be:

```
A circle with radius 5 has area 78.5398 and circumference 31.4159.
```

Finally we add in the ability to let the user change the value of the radius. To do this we need to add in two extra lines in between the declaration of radius and the calculations, as well as removing the assignment of value we already have. The two extra lines consist of one prompting the user for input, and one getting a value from the keyboard.

```
cout << " Input the value of the radius " << endl;
cin >> radius;
```

download

Check that you program compiles and runs as expected. Your final program should look like:

```
// use the input/output libraries and mathematical functions
#include <iostream>
#include <cmath>
// again use the namespace
using namespace std;

int main()
{
  // use descriptive names for variables
  double radius,pi,area,circumference;
  // assign those variables with a value
  pi=4.*atan(1.);
  // prompt the user for input
  cout << " Input the value of the radius " << endl;
  // get the value from the keyboard
  cin >> radius;
  // now do the calculations
  area = radius * radius * pi;
  circumference = 2. * pi * radius;
  // output results to screen
  cout << " A circle with radius " << radius << " has area ";
  cout << area << " and circumference " << circumference << "." << endl;
  return 0;
}
```

download

0.2 Data Types

0.2.1 Number Precision

Calculate the value of π using the expression $\pi = 4\tan^{-1}(1)$, using a float, a double and a long double. Print out the value of each result setting the precision of the output with the command:

```
cout.precision(80);
```

You will need the `iomanip` library included at the top of your program.

The precision of the numbers you do calculations with must be set when the variable is declared. There are two main types, a float and a double, which are single and double precision storage respectively. In this course we shall only use double precision, which gives around 15-16 digits of accuracy. Now if we are to solve this task, you must first consider what libraries/methods etc you will need. Here we are told that we need the `iomanip` library to specify outputs, but we also need the `cmath` lib for calculating inverse tan. Your initial program should look like:

```
// include the input output stream library
// so that you can write to the screen
#include <iostream>
// and the input/output manipulator library
#include <iomanip>
// we will also need the cmath library to
// get access to mathematical functions
#include <cmath>
// use the std namespace
using namespace std;

int main()
{
  // do nothing yet
  return 0;
}
```

download

Check that the code runs and compiles and does nothing.

Now we want to declare two different versions of pi using float and double.

```
  // store the variable pi using different data types
  float pi_f;
  double pi_d;
```

download

Next we do the calculation, using the formula in the question. We have to be very careful here to specify what type a number is. Simply writing `1` implies an integer, whilst writing `1.` implies a double. To be more specific write `1.f` for a float. You need to specify the correct type for each number in the calculation to avoid conversions in type.

```
  // call the atan function to calculate pi
  pi_f = 4.f*atan(1.f); // calls atan<float>(float x)
  pi_d = 4.*atan(1.); // calls atan<double>(double x)
```

download

Now try to output the numbers,

```
  cout << pi_f << endl;
  cout << pi_d << endl;
```

download

and you will see an output like this

```
3.14159
3.14159
```

so obviously we need to see more digits to tell the difference (6 digits is default). Adjust this with the command

```
   cout.precision(80);
```

download

and you get (depending on your system/compiler)

```
3.1415927410125732421875
3.14159265358979311599796346854418516159057617187 5
```

Even though you see around 24 digits for the float, only around 8 of those are actually accurate, the last 16 digits are created in the conversion from binary to decimal. I have run this on gcc 4.8 which means that you might see slightly different results. Again, any of the digits past where the accurate result for pi ends are created in the conversion from binary to decimal.

The final version of the code is here:

```
// include the input output stream library
// so that you can write to the screen
#include <iostream>
// and the input/output manipulator library
#include <iomanip>
// we will also need the cmath library to
// get access to mathematical functions
#include <cmath>
// use the std namespace
using namespace std;
```

```
int main()
{
  // store the variable pi using different data types
  float pi_f;
  double pi_d;

  // call the atan function, the letter f after a number
  // makes that number a float. The default is a double.
  pi_f = 4.f*atan(1.f);
  pi_d = 4.*atan(1.);
  // must be consistent using long doubles
  // to maintain accuracy

  cout.precision(80);

  cout << pi_f << endl;
  cout << pi_d << endl;

  return 0;
}
```

download

0.2.2 Automatic Conversions

Write a program containing the following commands:

```
double x=1.56,y=2.67;
int a=2,b=5;
cout << " a*x=" << a*x << " x/y=" << x/y << " a/b=" << a/b << endl;
```

Are the results what you would expect? How can we change last expression to get a fraction?

Solution is left as an exercise.

0.2.3 Working with Integers

Try declaring the following variables with these values:

```
int a=3.8;
int b=3.8;
int n=3.8+3.8;
```

Does $n = a + b$? If not why not???

Solution is left as an exercise.

0.3 Doing Calculations

0.3.1 Simple Expressions

Try doing some simple calculations with integers and doubles. Work out

- $$\frac{a}{x} + bx$$
- $$ax^2 + bx + c$$
- $$\frac{a + b * x}{c^4}$$

where

$$a = 1.17745, \quad b = 3, \quad c = -1004.1, \text{ and } x = 24.539 \ .$$

Use brackets to order the calculations. Try declaring the variables a, b, c and x as int, float or double – what difference does it make to the solution?

Solution is left as an exercise.

0.3.2 More Complex Expressions

Now include the `cmath` library in your program. Calculate:-

$$\frac{\sin(ax)}{x}$$

$$\sum_{i=1}^{5}(a + bx)^i$$

$$\sqrt{\sin(e^{x^a})}$$

where

$$a = 1.17745, \quad b = 3, \quad c = -1004.1, \text{ and } x = 24.539 \ .$$

The functions you will need are `sqrt`, `sin`, `cos`, `exp` and , `pow`. Look them up if required. What happens to the result of the top expression if $x = 0$?.

Solution is left as an exercise.

0.4 Logic

0.4.1 Conditional Statements.

Write a code that takes the arguments a, b and c, computes and then displays the roots of the quadratic equation

$$ax^2 + bx + c = 0$$

You will need to identify whether the roots are real or complex. If the roots are complex, display the results in the form $A + Bi$. Include the `<complex>` library, see how using this could make the previous problem easier.

The idea of this task is to create a simple logical step in which we define what the program should do in different circumstances. In the previous examples there was a linear flow in the logic of the program, which was

- Declare variables
- Assign values
- Calculate
- Output results

where as now we must have something like

- Declare variables
- Assign values
- Calculate
  - **If real::** Output real roots
  - **If complex::** Output complex roots

In order to access this functionality we use the `if` condition to evaluate whether or not something is true and then execute an action depending on the result.

Again when writing a program you should always be thinking about the structure and try to stage your program so that you can run and test as you go along. Those stages in this program might be:-

1. Include all libraries
2. Declare all variables and assign with default values
3. Calculate discriminant and output to screen
4. Use if condition to evaluate if real or not and output roots

5. Lift your code into a function and call it from main

Give the only libraries you should need are the input/output and mathematical functions your initial program should look like:

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

int main()
{
  // do nothing yet
  return 0;
}
```

download

Compile and run and check your program outputs a black terminal.

Now add in the variables we need, for a quadratic we will need to store all of the parameters a, b and c as well as the value of the discriminant.

```
double a,b,c,discriminant;
```

download

and assign default values to the parameters

```
a=1.;b=1.;c=1.;
```

download

Again check your program at every stage that it compiles and runs.

Next we can calculate the discriminant from the formula as

```
discriminant = b * b - 4. * a * c;
cout << " discriminant = " << discriminant << endl;
```

download

Check that the value of the discriminant appear to be correct.

Now use an if condition to do something different depending on whether the discriminant is positive or negative

```
if(discriminant < 0.)
{
  cout << " The roots are complex " << endl;
}
else
{
  cout << " The roots are real " << endl;
}
```

download

Now compile and run the program and check that the output is given as expected. Change the values of the parameters a, b and c and check that the program still works.

You may add an else if statement in the middle for the case when the discriminant is zero. Add some extra code inside the if blocks to output the values of the roots. Again check and compile that the code is working. At this stage your program should look something like:

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

int main()
{
  // declare all variables
  double a,b,c,discriminant;
  // assign default values
  a=1.;b=1.;c=1.;
  // calculate the discriminant
  discriminant = b * b - 4. * a * c;
  cout << " discriminant = " << discriminant << endl;
  // use if statement to change output
  if(discriminant < 0.)
  {
    cout << " The roots are complex and are " << endl;
    cout << " x+ = " << -b/2./a << "+" << sqrt(-discriminant)/2./a << "i" <<
        endl;
    cout << " x- = " << -b/2./a << "-" << sqrt(-discriminant)/2./a << "i" <<
        endl;
  }
  else
  {
    cout << " The roots are real and are" << endl;
    cout << " x+ = " << (-b + sqrt(discriminant))/2./a << endl;
    cout << " x- = " << (-b - sqrt(discriminant))/2./a << endl;
  }
  return 0;
}
```

download

Once your program has been thoroughly checked, we need to create a function above the main function that does everything in the main function. Now we need to think about what are the inputs and outputs from the function. The input will clearly be a, b and c, but what about outputs? There are only printed statements to screen so in fact the returned value is nil or void. As such we can write the definition as

```
void solveQuadratic(double a,double b,double c)
{

}
```

download

Enter this function definition into your code above the main function and below the libraries.

Now we can copy/paste all statements from the main into solveQuadratic (but not the return statement since there is no return value in this function). The only things that need to be changed are to remove a, b, and c from being redeclared and reassigned. The function can be called from main and you final program should look like:-

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

// function to solve a quadratic equation
//     a x^2 + b x + c = 0
// takes a, b, c as real number inputs
// output roots to the screen
void solveQuadratic(double a,double b,double c)
{
  // declare local variables
  double discriminant;
  // calculate the discriminant
  discriminant = b * b - 4. * a * c;
  cout << " discriminant = " << discriminant << endl;
  // use if statement to change output
  if(discriminant < 0.)
  {
    cout << " The roots are complex and are " << endl;
    cout << " x+ = " << -b/2./a << "+" << sqrt(-discriminant)/2./a << "i" <<
        endl;
    cout << " x- = " << -b/2./a << "-" << sqrt(-discriminant)/2./a << "i" <<
        endl;
  }
  else
  {
    cout << " The roots are real and are" << endl;
    cout << " x+ = " << (-b + sqrt(discriminant))/2./a << endl;
    cout << " x- = " << (-b - sqrt(discriminant))/2./a << endl;
  }
}

int main()
{
  solveQuadratic(1.,1.,1.);
  return 0;
}
```

download

0.5 Functions and Loops

0.5.1 Loops (i)

Write a loop to calculate and output the first n fibonacci numbers.

0.5.2 Loops (i)

Write a loop to calculate and output the binomial coefficients for $0 \leq k \leq n$:

$$\begin{pmatrix} n \\ k \end{pmatrix}$$

0.5.3 Approximating sin(x)

Write a function to compute the value of $\sin(x)$ using the first N terms of the power series expansion (N should be an argument to the function):

$$\sin(x) \approx \sum_{k=0}^{N} (-1)^k \frac{x^{2k+1}}{(2k+1)!}$$

How many terms of the series are required to agree with the built-in sin function. Produce output of the two functions side by side for the values 0, $\pi/6$, $\pi/4$, $\pi/2$, $2\pi/3$ and π.

Rewrite the above function for sin so that terms are added until the result is accurate to 10 decimal places. Think about what condition is satisfied for this to happen i.e. what is the truncation error?

The idea of this task is to create a simple function making use of loops and eventually a break command to approximate sin(x) with a power series. Yet again I can't emphasise enough how important it is when writing a program that you should always be thinking about the structure and how you are going to stage your calculations. For this particular example, those stages might be:-

1. Include all libraries
2. Declare all variables and assign with default values
3. Create a loop for summing up values
4. Demonstrate ability to calculate each part of expression on each loop
5. Check results against other solutions
6. Move code into a function

Give the only libraries you should need are the input/output and mathematical functions your initial program should look like:

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

int main()
```

```
{
  return 0;
}
```

download

Compile and run and check your program outputs a blank terminal.

Now add in the variables in the problem, the most obvious variables required are the integers N and k required for counting in the loop, and we obviously need to store x but also need to keep track of the factorial, both of these last 2 variables are double because we are doing calculations with them. Note that even though factorial is always an integer we must use a double to make use of the larger maximum and minimum values that can be stored.

```
int main()
{
  // counting variables
  int N=10,k=0;
  // real variables
  double x=0.5,factorial=1.;

  return 0;
}
```

download

Note that we have chosen sensible starting values for the variables, compile, run and check your program outputs a blank terminal.

Next we add in the loop. Here we are going to start keeping track of the factorial term. By the definition of a factorial, we can find the value of the factorial at the current loop by multiplying the value from the last loop by k.

```
  // for k=1 up to k==n
  for(int k=1;k<=N;k++)
  {
    factorial = factorial * k;
    cout << k << "!=" << factorial << endl;
  }
```

download

This should output the values for k! between 1 and 10. By a similar argument we can get the value of the factorial (2k+1)! by multiplying the previous value by 2k and 2k+1. In the algorithm we may write

```
  // for k=1 up to k==n
  for(int k=1;k<=N;k++)
  {
    factorial = factorial * (2*k) * (2*k+1);
    cout << 2*k+1 << "!=" << factorial << endl;
  }
```

download

This should output the values for (2k+1)! between 3 and 21.

Next add in some code to get alternating plus and minus. The easiest way to do this is to have a variable that we multiply by minus at every step. You just need to be careful that it is minus on the odd loops and plus on the even ones. We might also print out the value of x to the power 2k+1 to check it gives a sensible value.

```
double plusminus=1.;
// for k=1 up to k==n
for(int k=1;k<=N;k++)
{
  factorial = factorial * (2*k) * (2*k+1);
  cout << 2*k+1 << "!=" << factorial << endl;
  plusminus=plusminus*-1.;
  cout << "+-=" << plusminus << endl;
  cout << "x^{2k+1}=" << pow(x,2*k+1) << endl;
}
```

download

Compile and run the code and check that all the values give sensible results.

Now all that is left to do is to sum up all of those parts in the algorithm. To do this we use the dummy variable sum to keep track of the summation. Note that we must initialise the value of sum given k=0 since our loop only runs from k=1 up to N.

```
// initialise values for k=0 separately
double plusminus=1.,sum=x;
// for k=1 up to k==n
for(int k=1;k<=N;k++)
{
  factorial = factorial * (2*k) * (2*k+1);
  plusminus=plusminus*-1.;
  sum = sum + plusminus*pow(x,2*k+1)/factorial;
}
cout << " sin("<<x<<";"<<N<<")="<<sum<<endl;
cout << " sin("<<x<<")="<<sin(x)<<endl;
```

download

At this stage you should test the accuracy of the solver using different values of x.

Now we are able to move this piece of code outside of the main function into it's own function. Remember the value we want to return from the function is the sum variable. The definition of the function should look like

```
double mySinFunction(double x,int N)
{
  // paste algorithm in here
  return sum;
}
```

download

Create this function above the main function and copy paste the code from the main function in. You should take out the declarations for x and N. Inside main you can now

run the algorithm by simply calling the function as demonstrated here

```
cout << mySinFunction(0.5,10) << endl;
```

download

The final program should look like:

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

double mySinFunction(double x,int N)
{
  // counting variables
  int k=0;
  // real variables
  double factorial=1.;
  // initialise values for k=0 separately
  double plusminus=1.,sum=x;
  // for k=1 up to k==n
  for(int k=1;k<=N;k++)
  {
    factorial = factorial * (2*k) * (2*k+1);
    plusminus=plusminus*-1.;
    sum = sum + plusminus*pow(x,2*k+1)/factorial;
  }
  return sum;
}

int main()
{
  int N=10;
  double x=0.5;
  cout << " sin("<<x<<");"<<N<<")=";
  cout << mySinFunction(x,N)<<endl;
  cout << " sin("<<x<<")="<<sin(x)<<endl;
  return 0;
}
```

download

0.6 Write to File

Create a simple graph by outputting data to a file

First I will try to explain how to do this with a simple example on different platforms. Unix and Windows have different ways of expressing file paths so you need to make sure you use the correct path depending on the machine that you are using.

Open up a new project and enter the following code as your main file

```
// A program to write to a file which can be opened in a spreadsheet
```

```
#include <iostream>
// use fstream for opening files for input/output
#include <fstream>
using namespace std;

int main()
{
  // open up a file stream to write data
  ofstream output;
  // here we are going to use comma separated variables, so end the filename
      with .csv

  // If I am on a normal windows desktop the open command will be something
      like:
//  output.open("C:/Documents and Settings/Paul Johnson/My Documents/Data/
    test.csv");
  // If I am on a university cluster windows machine the open command will be
       something like:
// output.open("P:/Data/test.csv");
  // If I am on a unix machine the open command will be something like:
// output.open("/home/pjohnson/Data/test.csv");

  // INPUT YOUR OPEN COMMAND HERE
  output.open("MYPATHTO/test.csv");

  if(!output.is_open())
  {
    // NOTE!!!! The file will not open unless the directory exists!!!
    cout << " File not opened \n";
    // stop the program here
    throw;
  }
  // write x vs x^2 to a file
  // each column must be separated by a comma
  // each row goes on a new line
  for(int i=0;i<11;i++)
  {
    double x=i*0.1;
    output << x << " , " << x*x << endl;
  }
  // file write successful then close file
  cout << " File write successful \n";
  output.close();

  return 0;
}
```

download

Now edit the line containing `output.open("MYPATHTO/test.csv");`. The string containing the quotes must give a **valid** path to the directory that you want to open the file in. If you do not specify a path, e.g. `output.open("test.csv");` then the file will be

saved in the directory in which the program is run. You can try it this way and see if you can find the file!!! By a valid path, we mean that the directory must **exist**, i.e. that you must have already created it. So open up My Documents (or your home directory in unix) and create a new folder called "Data". The path to that directory should look something like:

- Normal Windows Desktop:- `C:/Documents and Settings/username/My Documents/Data/`
- University Cluster Windows Desktop:- `P:/Data/`
- Unix (linux):- `/home/username/Data/`
- Unix (Mac OSX):- `/Users/username/Data/`

So now enter (inside quotes) the full path to the directory and a filename to write to. There are some examples that I might use in the code. Obviously you will have to substitute your name or username.

Now run the program. If you see the text `File write successful` everything should have worked. Some points to note:

- Windows **LOCKS** files once they are in use by another program. This means you **CANNOT** write to a file if it is open in excel (or any other program).
- The file ending csv tells windows what sort of file to expect, you could call it anything you want.

Now try plotting the data file using one of the following:

- Gnuplot (unix help windows help)
- Matlab (help)
- Spreadsheet application such as excel or openoffice