Cpp Notes

1 Preamble Code:

```
#include <iostream>
using namespace std;
      datatype of output
int main ( )
{ return 0;   always end
              line with
}             semi-colon
```

iostream → class defined
→ input output stream

2 * Control/Selection Structure

1) Single Selection (if)
2) Double Selection (if/else)
3) Multiple Selection (switch)

* Repition Structure

1) while (<condition>)
```
{
}
```
2) do/while

3) for (intial ; constraint; step)
```
{
}
```

break;
ends loop

continue;
skips condition
but stays in loop

3 Logical Operators

- AND — &&
- OR — ||
- NOT — !

count++ ↔ count = count + 1

```
// program to sum  first 100 natural numbers

#include <iostream>
 using namespace  std;

 int main ( )
{
 int  sum = 0;
 int i;

 for (i=1,  i<101, i++)
    {
     sum = sum +i;
    }

 cout << " The sum of the 100 numbers is "
        <<  sum << endl; }
```

④ Functions/Modules

A function/module is a block of code that only runs when called. It can be reused as many times, no need to write it again and again.

Example:

$* \underbrace{\text{int}}_{\text{return type of the function}} \ \overbrace{\text{cube}}^{\text{function name}} \ \underbrace{(\text{int } n)}_{\text{input parameter(s)}} \ \{ \xrightarrow{\quad} \text{declaration of the function}$

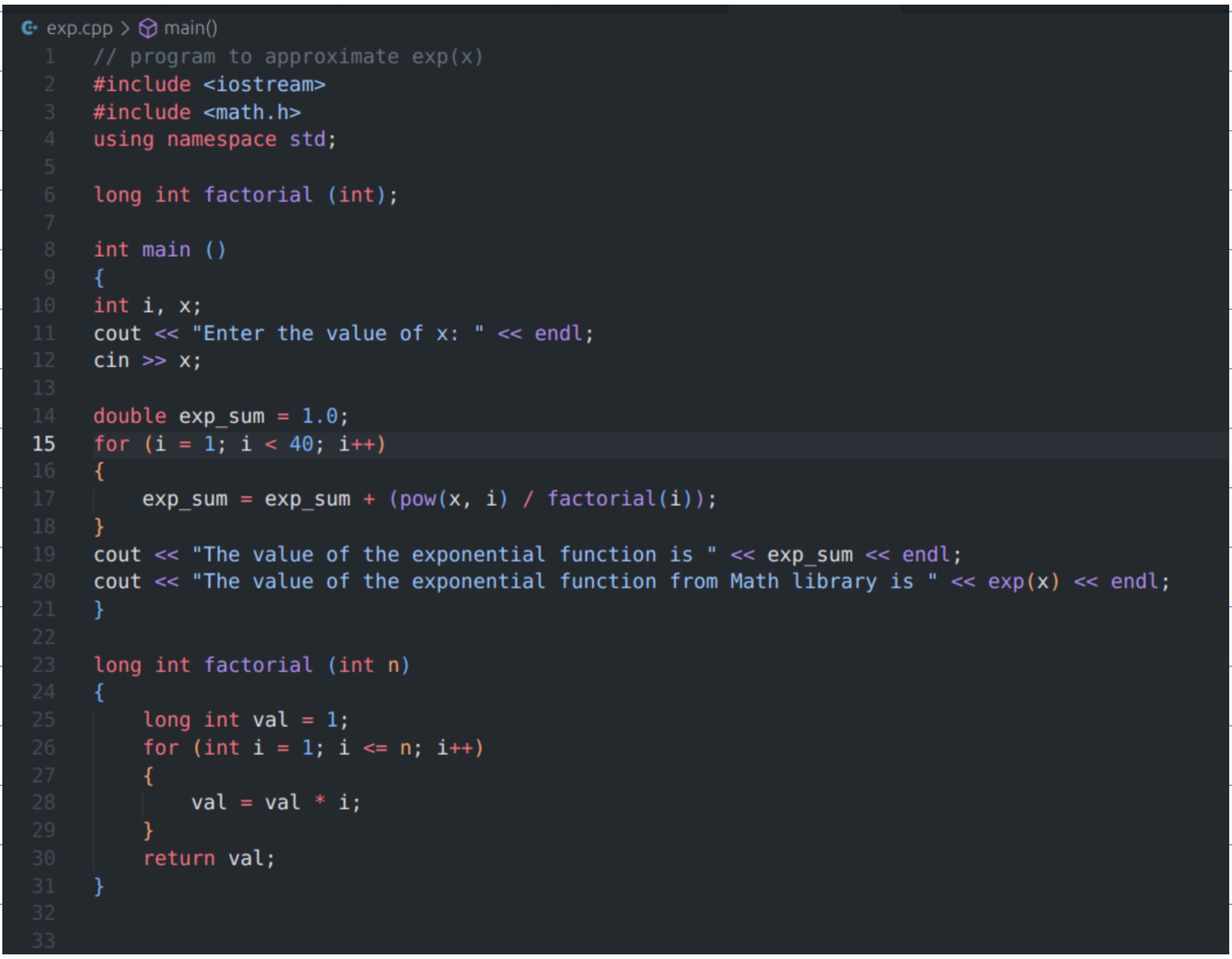
< definition of the function >

}

✳ Note :

Unlike Python, the definition of the function can be written at any point in the program file, even after it has been called. This is because C++ is compiler-based.

(Ex.) Write a function which calculates the exponential function and compare it to that of the math class.


exp.cpp > main()

```cpp
// program to approximate exp(x)
#include <iostream>
#include <math.h>
using namespace std;

long int factorial (int);

int main ()
{
int i, x;
cout << "Enter the value of x: " << endl;
cin >> x;

double exp_sum = 1.0;
for (i = 1; i < 40; i++)
{
    exp_sum = exp_sum + (pow(x, i) / factorial(i));
}
cout << "The value of the exponential function is " << exp_sum << endl;
cout << "The value of the exponential function from Math library is " << exp(x) << endl;
}

long int factorial (int n)
{
    long int val = 1;
    for (int i = 1; i <= n; i++)
    {
        val = val * i;
    }
    return val;
}


```

→ Datatypes Used Above:

1) long int → used to store large range integers
2) int → used to store integers
3) double → used to store large float values.

→ The main() function

This is the entry point of every C++ program. When the program runs, execution starts from the main function. It contains the call to the other functions.

⑤ Arrays

Arrays are used to store multiple values in a single variable. Moreover, arrays store values in a contiguous manner.

int $a_1, a_2, a_3, \dots;$ ⟶ randomly allocated in memory

$\underbrace{\text{int}}_{\text{type of values in array}}\ \underbrace{\text{a}}_{\text{array name}}[\underbrace{10}_{\text{length of the array}}];$ ⟶ consecutively allocated in memory

Example: <Type> <array name>[<array size>];

- int a[3] = {5, 9, 30};

  a[0] → 5

  a[1] → 9

  a[2] → 30

Good Coding Practice:

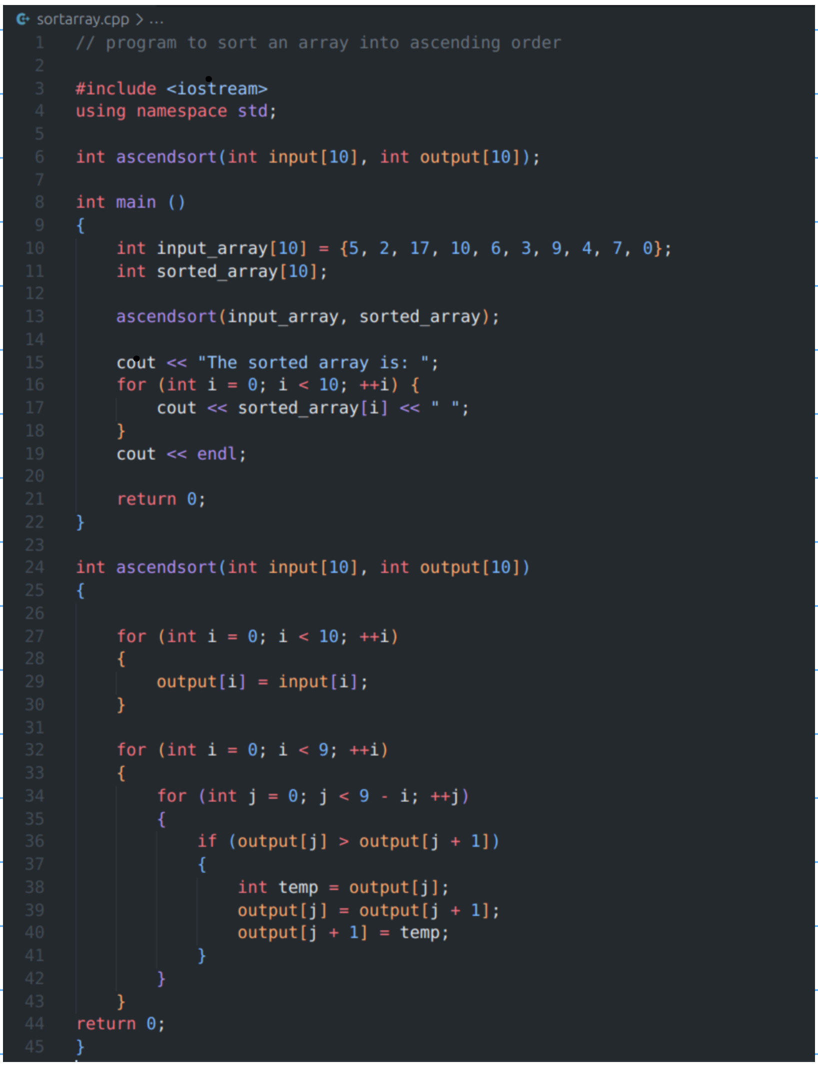
$\text{int a[10]} = \underbrace{\{0\}}_{\text{declare values of the array as 0!}};$

- Nested Array:

  $\text{int a}[\underbrace{n}_{\text{rows}}][\underbrace{m}_{\text{columns}}];$

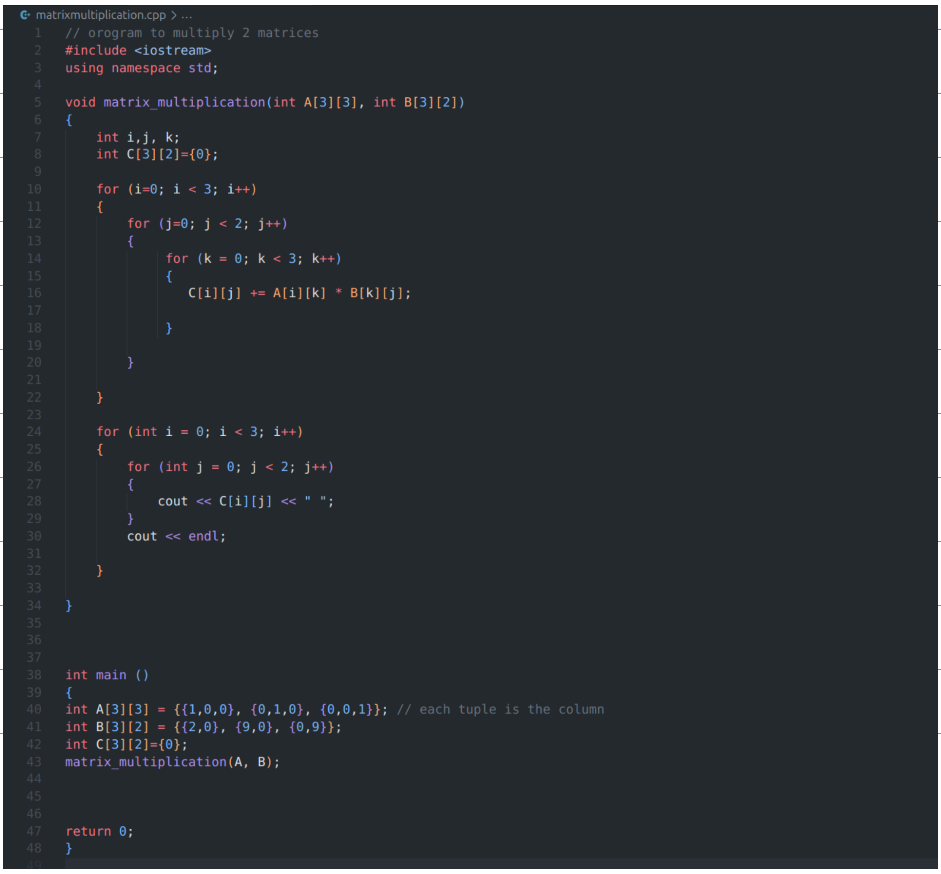
(Ex.) Write a program that takes an array and sorts it in ascending/descending order.

(Ex.) Write a program that multiplies 2 matrices.



sortarray.cpp > ...

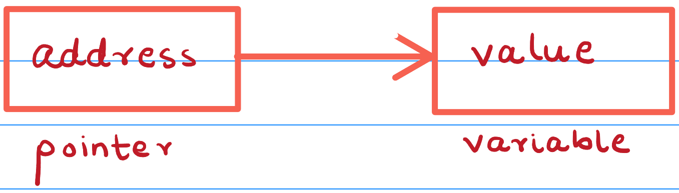
```cpp
// program to sort an array into ascending order

#include <iostream>
using namespace std;

int ascendsort(int input[10], int output[10]);

int main ()
{
    int input_array[10] = {5, 2, 17, 10, 6, 3, 9, 4, 7, 0};
    int sorted_array[10];

    ascendsort(input_array, sorted_array);

    cout << "The sorted array is: ";
    for (int i = 0; i < 10; ++i) {
        cout << sorted_array[i] << " ";
    }
    cout << endl;

    return 0;
}

int ascendsort(int input[10], int output[10])
{

    for (int i = 0; i < 10; ++i)
    {
        output[i] = input[i];
    }

    for (int i = 0; i < 9; ++i)
    {
        for (int j = 0; j < 9 - i; ++j)
        {
            if (output[j] > output[j + 1])
            {
                int temp = output[j];
                output[j] = output[j + 1];
                output[j + 1] = temp;
            }
        }
    }
return 0;
}
```



matrixmultiplication.cpp > ...

```cpp
// orogram to multiply 2 matrices
#include <iostream>
using namespace std;

void matrix_multiplication(int A[3][3], int B[3][2])
{
    int i,j, k;
    int C[3][2]={0};

    for (i=0; i < 3; i++)
    {
        for (j=0; j < 2; j++)
        {
            for (k = 0; k < 3; k++)
            {
                C[i][j] += A[i][k] * B[k][j];

            }

        }

    }

    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 2; j++)
        {
            cout << C[i][j] << " ";
        }
        cout << endl;

    }

}



int main ()
{
int A[3][3] = {{1,0,0}, {0,1,0}, {0,0,1}}; // each tuple is the column
int B[3][2] = {{2,0}, {9,0}, {0,9}};
int C[3][2]={0};
matrix_multiplication(A, B);



return 0;
}
```

⑥ Pointer



A pointer is a variable that stores the memory address as its value.

The way to obtain the memory address of a variable is to add "&" in front of the variable.

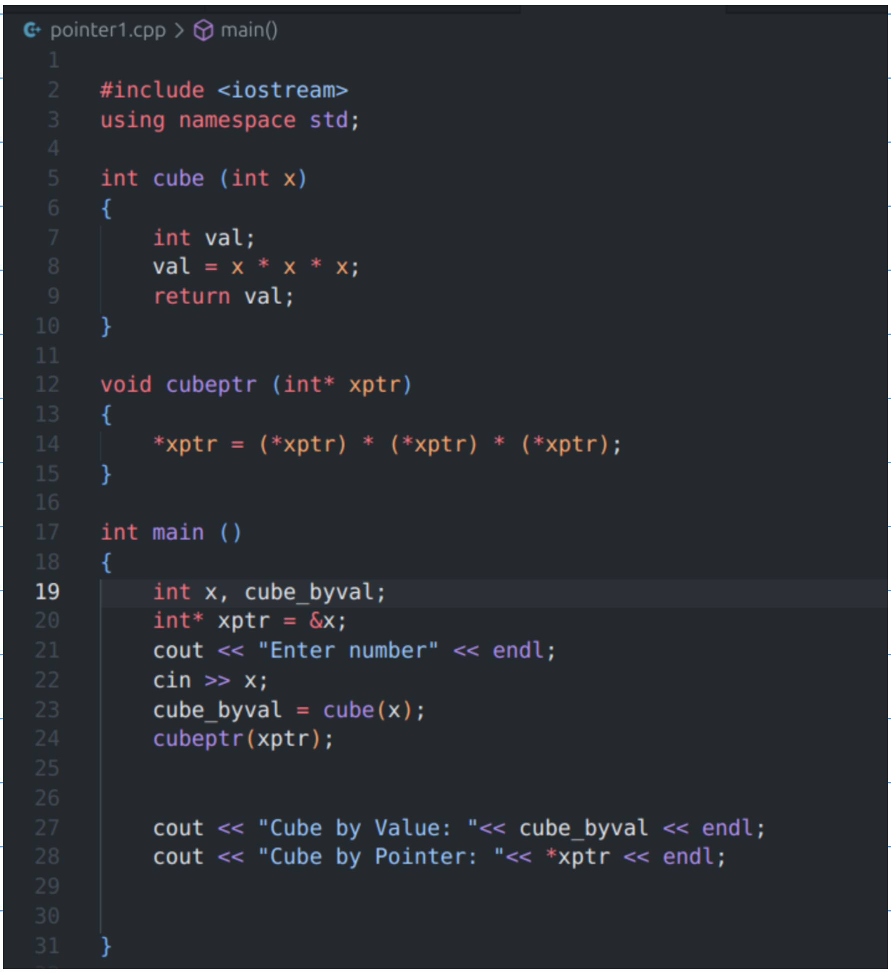
Example:

- int* xptr;
  int x = 5;
  xptr = &x;

$\rightarrow$ <data type>* declares a pointer.

<variable> = &<variable> stores the memory address.

- The following is an example of using pointer:



pointer1.cpp > main()

```cpp

#include <iostream>
using namespace std;

int cube (int x)
{
    int val;
    val = x * x * x;
    return val;
}

void cubeptr (int* xptr)
{
    *xptr = (*xptr) * (*xptr) * (*xptr);
}

int main ()
{
    int x, cube_byval;
    int* xptr = &x;
    cout << "Enter number" << endl;
    cin >> x;
    cube_byval = cube(x);
    cubeptr(xptr);


    cout << "Cube by Value: "<< cube_byval << endl;
    cout << "Cube by Pointer: "<< *xptr << endl;


}
```

☆ The "void" type :

Declaring a function's return type as void implies the function does not to return anything.

⑦ File Handling in Cpp

The class to handle files is called "fstream".

#include <fstream>

↳ ifstream : for reading files

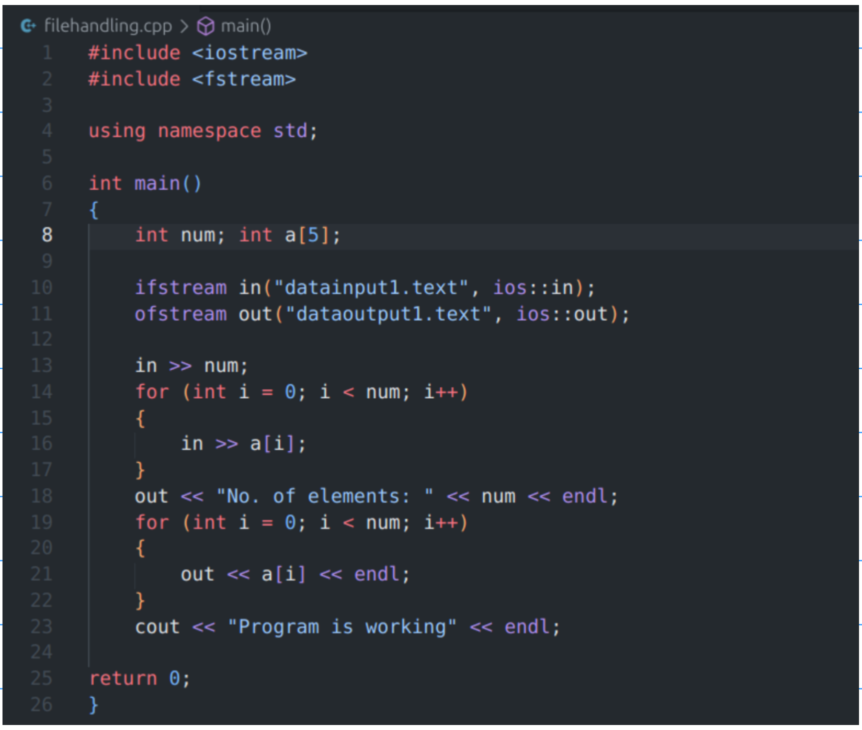
↳ ofstream : for writing files

Example Usage:

☆ The input file : datainput1.text

4 10 1 6 89

☆ The cpp file looks like this ↓


filehandling.cpp > main()

```cpp
#include <iostream>
#include <fstream>

using namespace std;

int main()
{
    int num; int a[5];

    ifstream in("datainput1.text", ios::in);
    ofstream out("dataoutput1.text", ios::out);

    in >> num;
    for (int i = 0; i < num; i++)
    {
        in >> a[i];
    }
    out << "No. of elements: " << num << endl;
    for (int i = 0; i < num; i++)
    {
        out << a[i] << endl;
    }
    cout << "Program is working" << endl;

return 0;
}
```

The first element of the input file is the number of elements other than the first.

The program reads the file and writes those elements in the output file.

☆ The output file : dataoutput1.text

No. of elements : 4

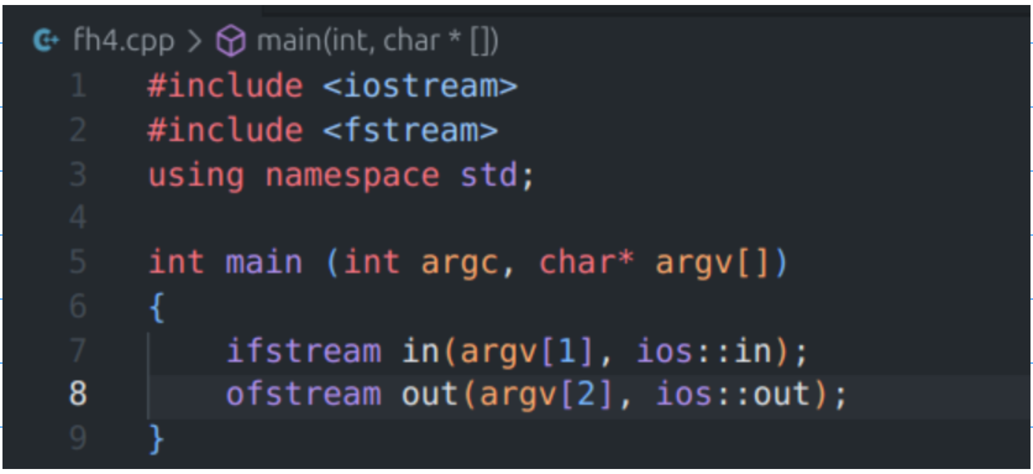
10

1

6

89

☆ This is another example of taking input files and outputting files :



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

int main (int argc, char* argv[])
{
    ifstream in(argv[1], ios::in);
    ofstream out(argv[2], ios::out);
}
```

- $\text{int main (int } \underbrace{\text{argc}}_{\text{for input file}} \text{, char* } \underbrace{\text{argv[]}}_{\text{for output file.}} \text{)}$

- {

  ```
      ifstream in (argv[1], ios :: in);
      ofstream out (argv [2], ios :: out);
  }
  ```

- Now, let's see what this means:

  Let us look at what commands we write, on the terminal.

  ```
  $ g++ <filename>.cpp -o <executible>
  $ ./<executible> <input file> <output file>
  ```

  $\underbrace{\text{./<executible>}}_{\text{argv[0]}} \quad \underbrace{\text{<input file>}}_{\text{argv[1]}} \quad \underbrace{\text{<output file>}}_{\text{argv[2]}}$

  $\boxed{\text{This way you can input/output files at the command line.}}$

8 Class

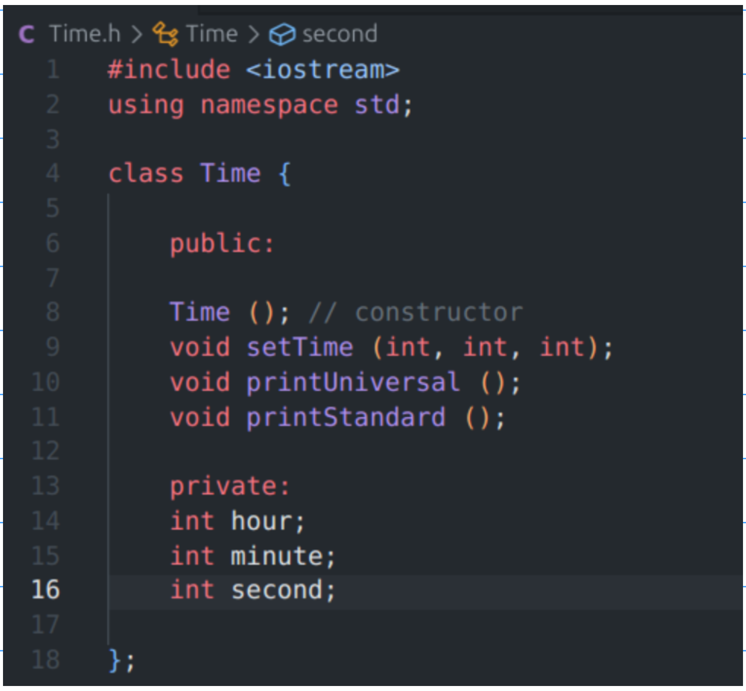
A class is a user defined data type, which holds its own data members and member functions that can be accessed and used by creating an instance of that class.

How to define a class ?

```
class <class name> {
// define data members and member functions
};
```

→ Data Members: These are variables defined inside the class.

→ Member Functions: Functions declared inside a class. Also called a member method.



Time.h > Time > second

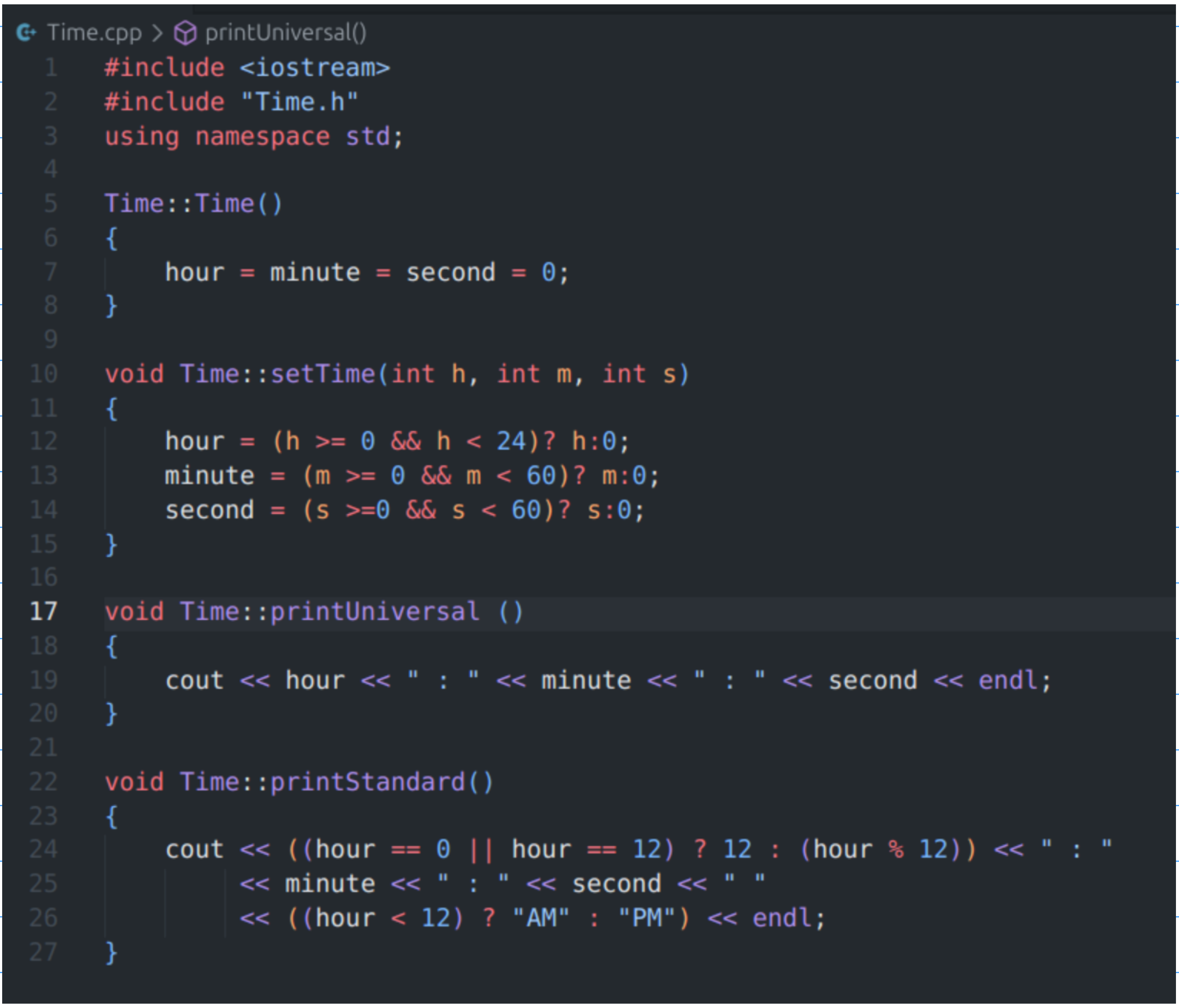
```cpp
#include <iostream>
using namespace std;

class Time {

    public:

    Time (); // constructor
    void setTime (int, int, int);
    void printUniversal ();
    void printStandard ();

    private:
    int hour;
    int minute;
    int second;

};
```

Let's start by building our first class.

Pay attention to the syntax!

This is a ↗ header file. The extension is .h .

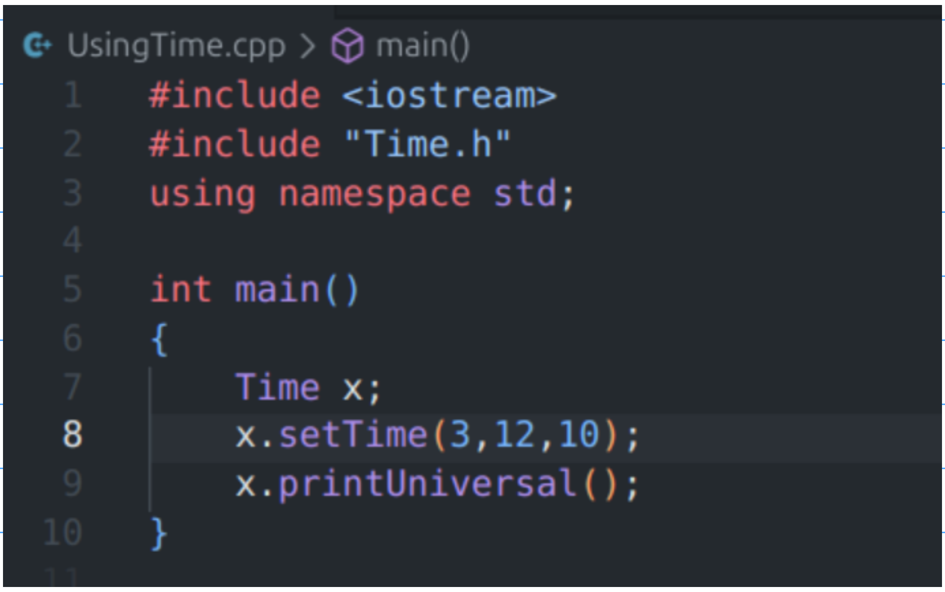
This is the .cpp file where everything gets defined



```cpp
#include <iostream>
#include "Time.h"
using namespace std;

Time::Time()
{
    hour = minute = second = 0;
}

void Time::setTime(int h, int m, int s)
{
    hour = (h >= 0 && h < 24)? h:0;
    minute = (m >= 0 && m < 60)? m:0;
    second = (s >=0 && s < 60)? s:0;
}

void Time::printUniversal ()
{
    cout << hour << " : " << minute << " : " << second << endl;
}

void Time::printStandard()
{
    cout << ((hour == 0 || hour == 12) ? 12 : (hour % 12)) << " : "
         << minute << " : " << second << " "
         << ((hour < 12) ? "AM" : "PM") << endl;
}
```

Now, how do you execute this on the command line?

To do this, we must first create a .cpp file where we will be calling this class and using the member methods or functions defined in the class.


UsingTime.cpp > main()

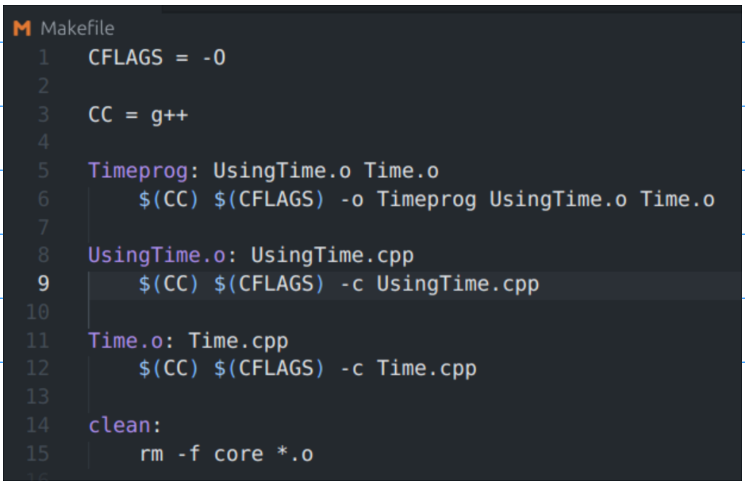
```cpp
#include <iostream>
#include "Time.h"
using namespace std;

int main()
{
    Time x;
    x.setTime(3,12,10);
    x.printUniversal();
}
```

We wrote #include "Time.h" instead of <Time.h> to indicate the class file was in the same directory as the current file.

Now, let's execute this file.

```
$ g++ -c Time.cpp -o Time.o

$ -c UsingTime.cpp -o UsingTime.o

$ Time.o UsingTime.o -o Timeprog
```

But this seems like a lot to execute. So what we do is create something called a Makefile.


Makefile

```makefile
CFLAGS = -O

CC = g++

Timeprog: UsingTime.o Time.o
	$(CC) $(CFLAGS) -o Timeprog UsingTime.o Time.o

UsingTime.o: UsingTime.cpp
	$(CC) $(CFLAGS) -c UsingTime.cpp

Time.o: Time.cpp
	$(CC) $(CFLAGS) -c Time.cpp

clean:
	rm -f core *.o
```

Now, just running the following on the command line will work:

$\$$ make -f $\underbrace{\text{Makefile}}_{\text{name of make file.}}$

⋆ Access Modifiers (forgot to mention):

1) **Public** : members of this class can be accessed from outside the class.

2) **Private**: can only be accessed within the class

3) **Protected**: can be accessed with the class and by derived class

↳ If not specified, Private access is assumed by the computer.

Completed: 30/07/25

Notes by

Anwesha Ghosh