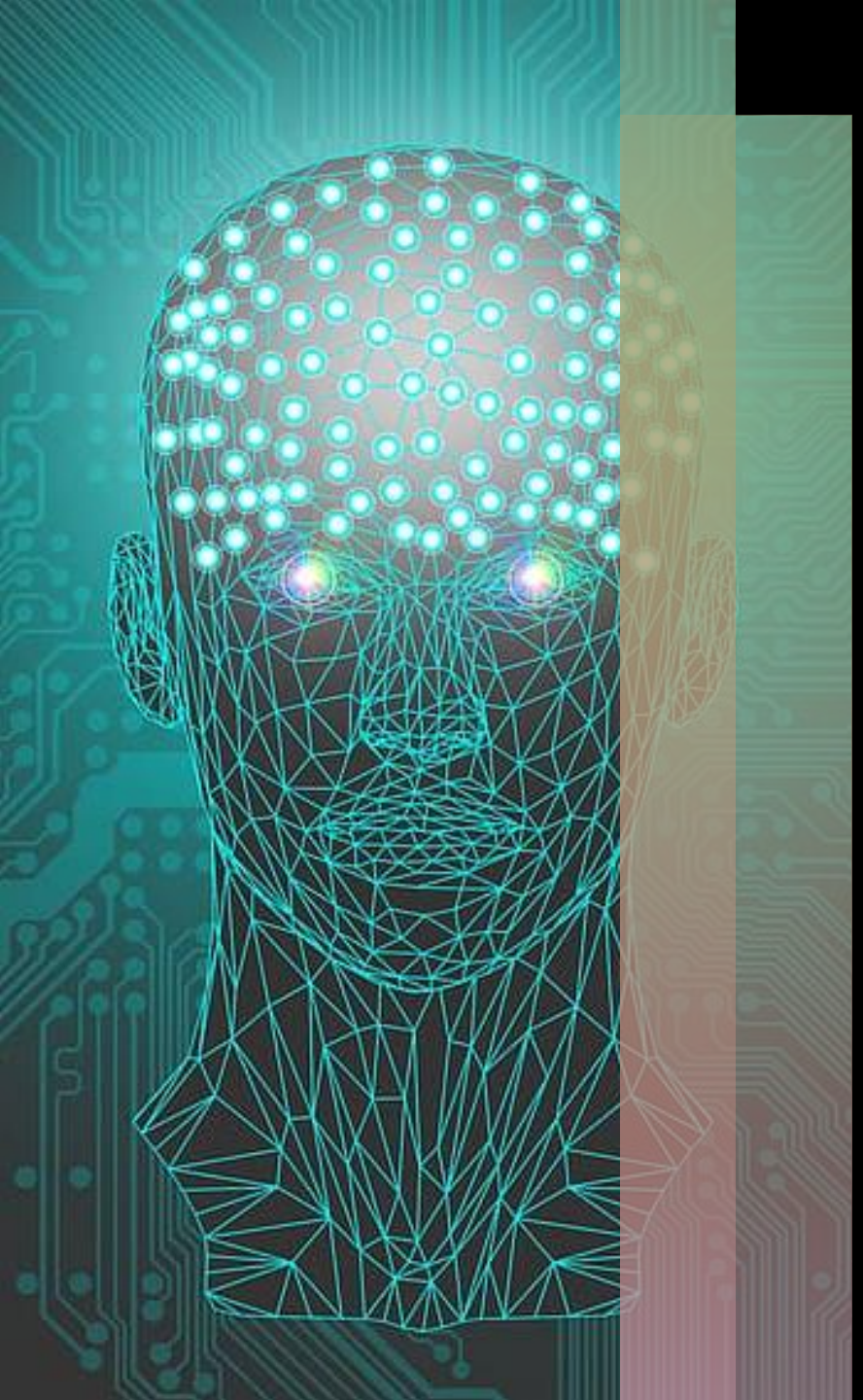
COM4013 Introduction to Software Development

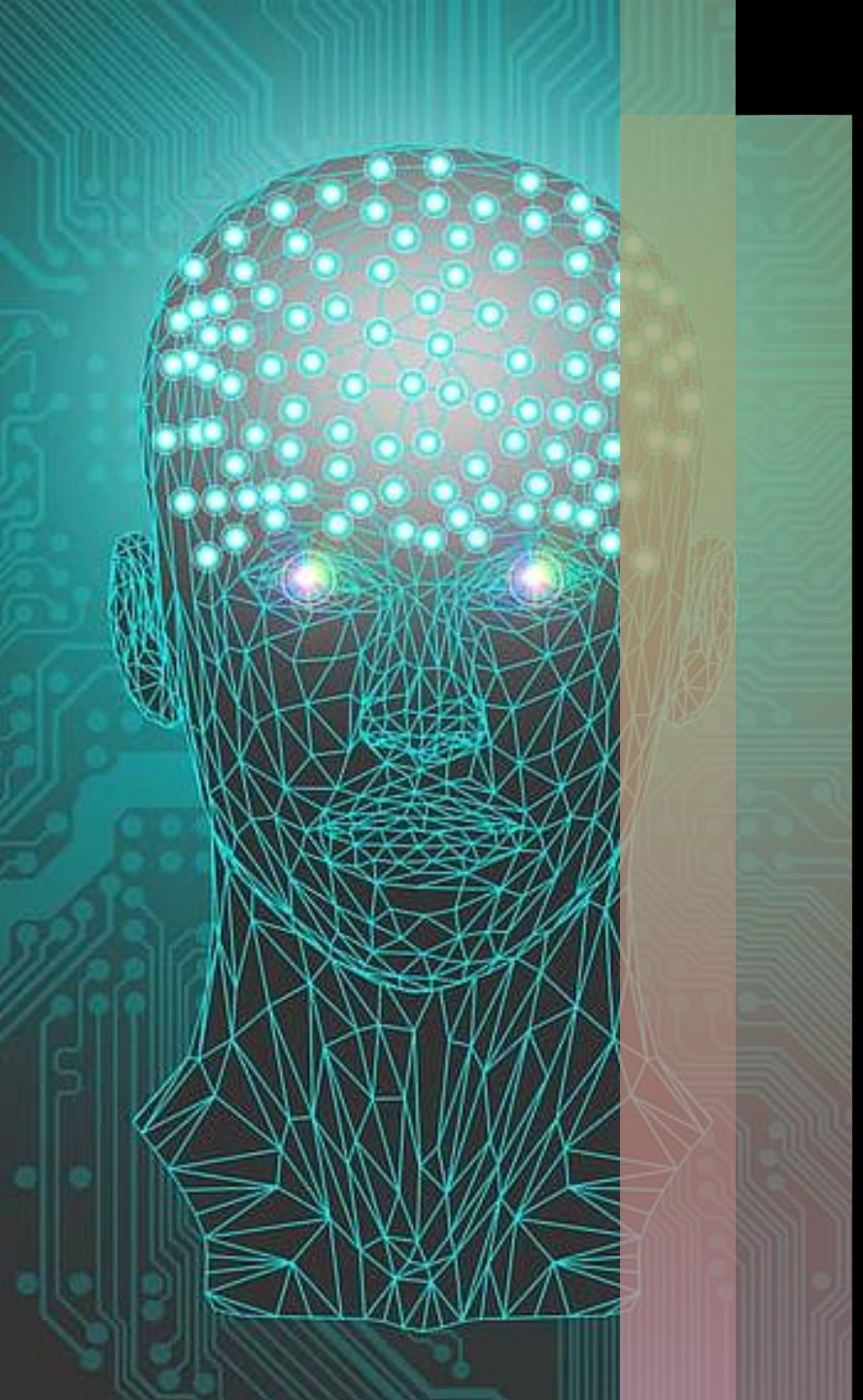
- Week 12
- Introduction to Algorithms and Pseudocode

- *Umar Arif – u.arif@leedstrinity.ac.uk*



Introduction to File Streams

- Files in C++ are handled through file streams, similar to console I/O but requiring the declaration of new streams.
- Three types of file streams:
  - ifstream: Input stream (similar to cin for reading).
  - ofstream: Output stream (similar to cout for writing).
  - fstream: Input/output stream.
- Header inclusion is necessary: #include <fstream>.

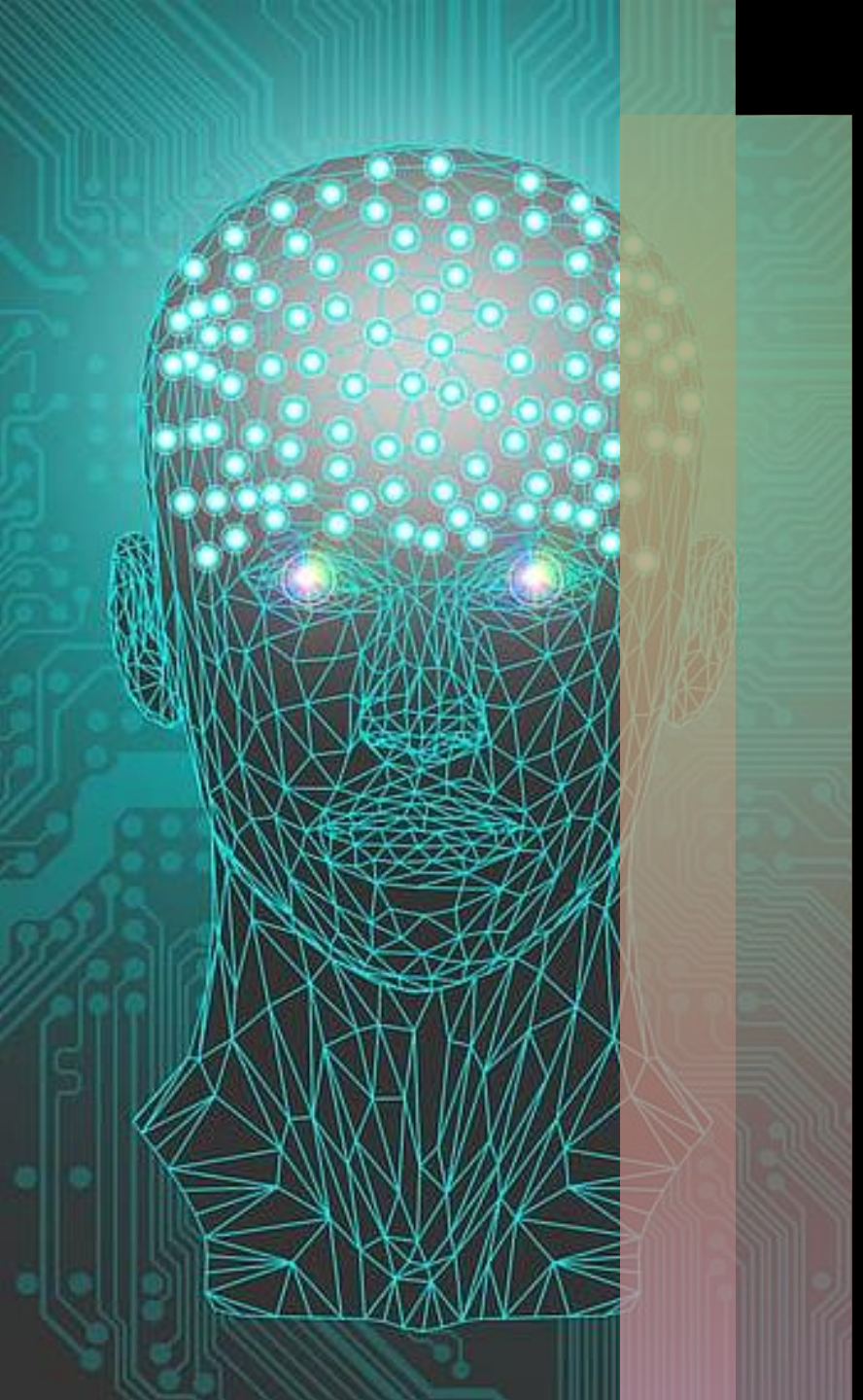


File Input Operations

- File input operations are analogous to standard input
- To read/write from/to a file, use the following sequence:

- Open a file for input:
  - `ifstream infile("filename");`
- Read from an input file:
  - `infile >> data1 >> data2;`
- `Write to an ouput file:`
  - `outfile << data1 << data2 << endl;`
- Close an input file once finished:
  - `infile.close();`

- You need to open the file and declare the file stream
- You need to check that the file has been opened successfully, e.g., that the file exists. An error will occur if you read from a non-existent file. You need to close the file after it's used.

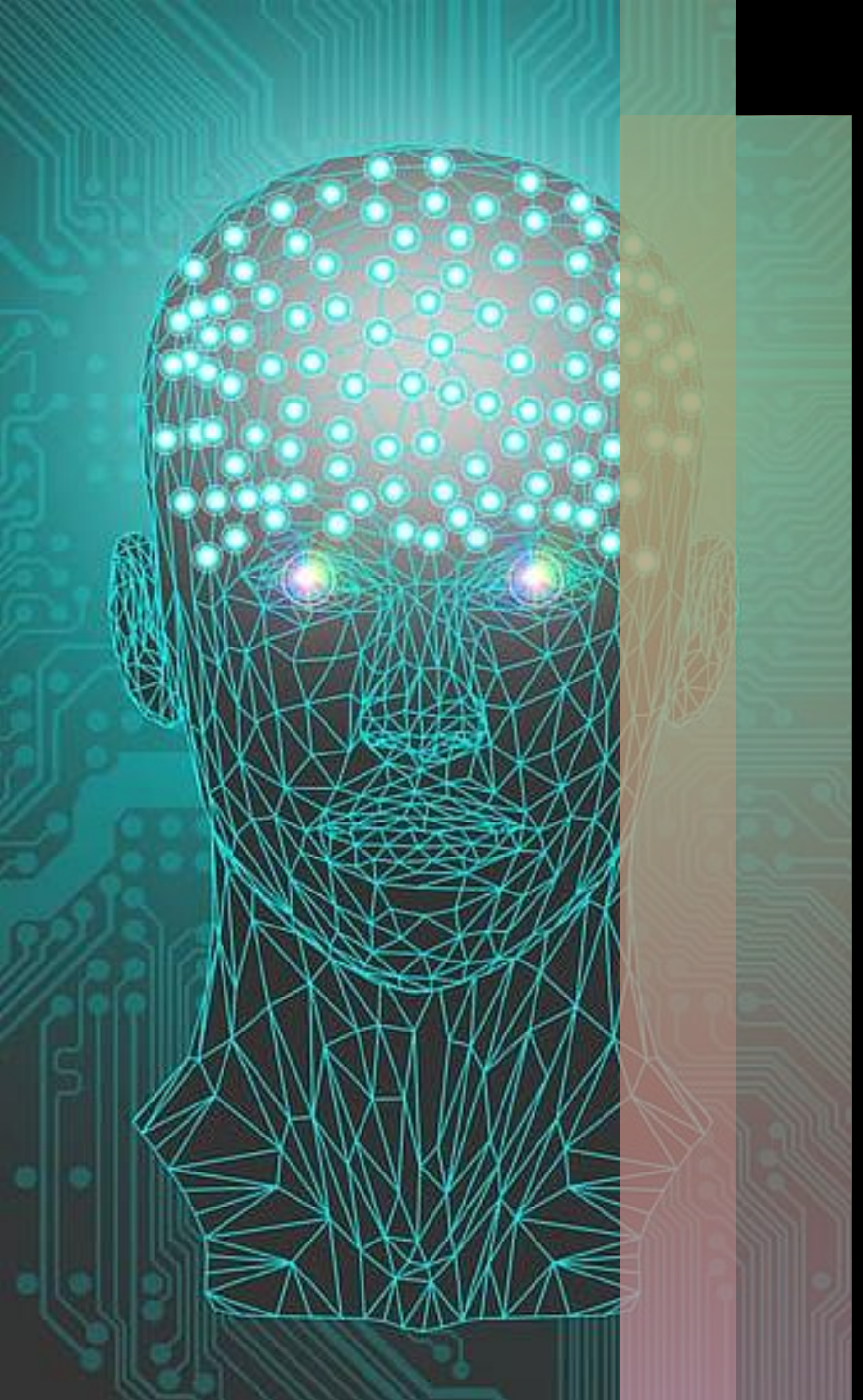


File Existence Checking

- Before performing input operations, check if the file exists:

```
void OpenFile( ifstream& infile )
{
  infile.open( "input.txt" );
  if( !infile )
  {
     cout << "ERROR: Can't open input file\n";
  }
}
```

- Error checking is important to avoid issues with non-existent files
- Note the use of the reference parameter. What is this doing?

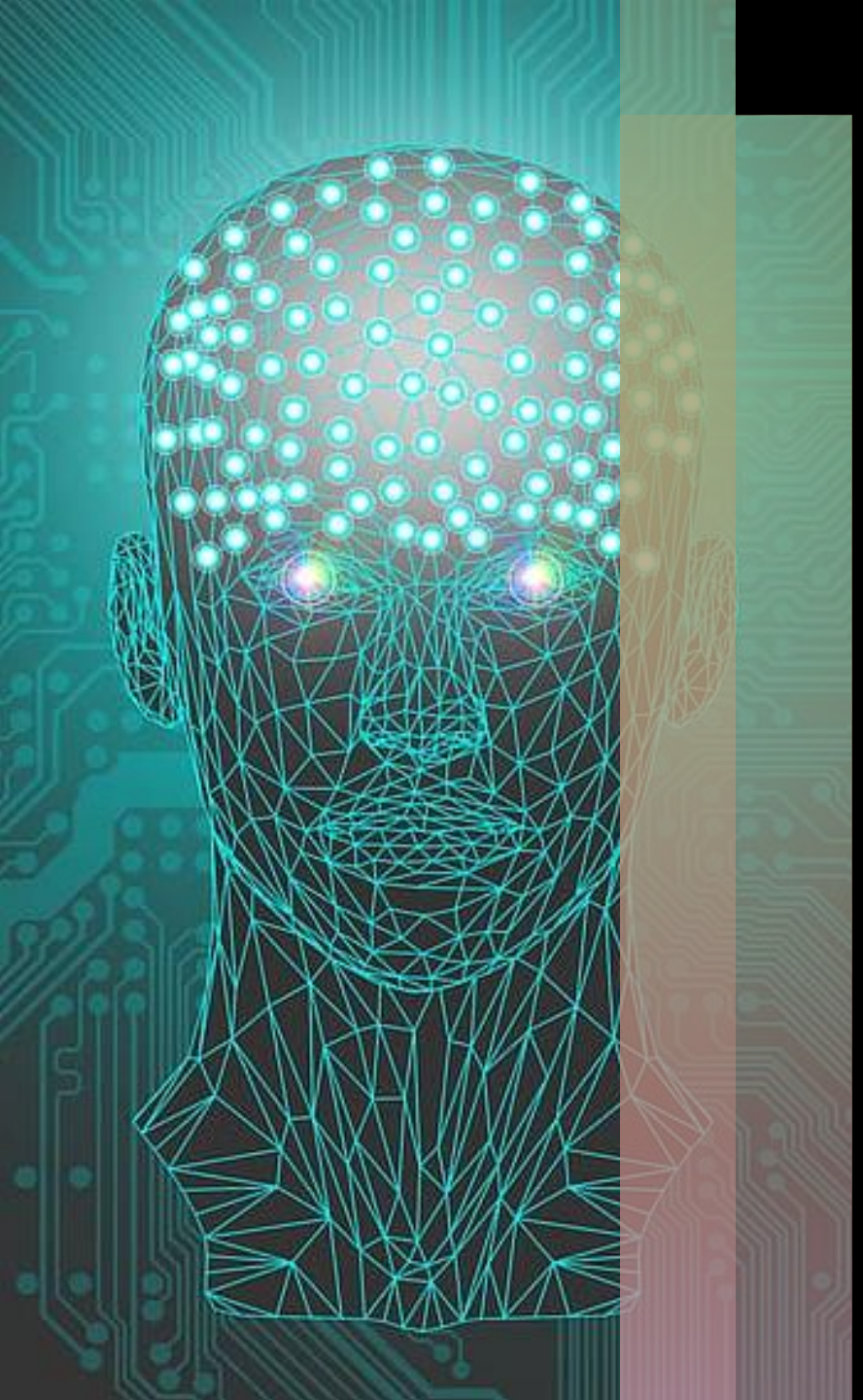


Handling End-of-File

- File input operations require detecting the end of the file:

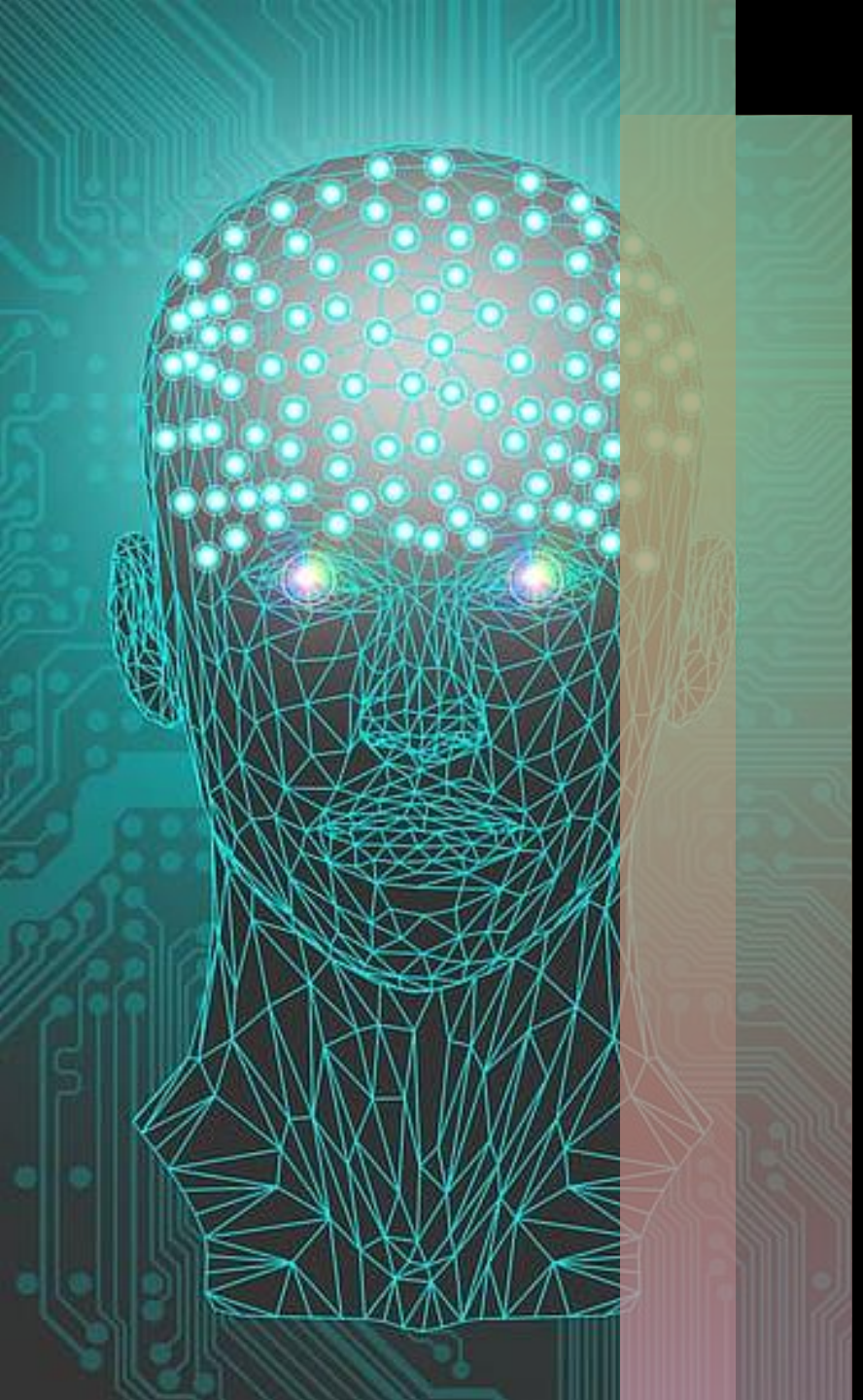
```
while( !infile.eof() )
{
  char ch;
  infile >> ch;
  if( !infile.eof() )
  {
     cout << ch;
  }
}
```

- eof() doesn't return true until it read after the end of the file. You will get a spurious extra character at the end without the test.



Handling End-of-File Function

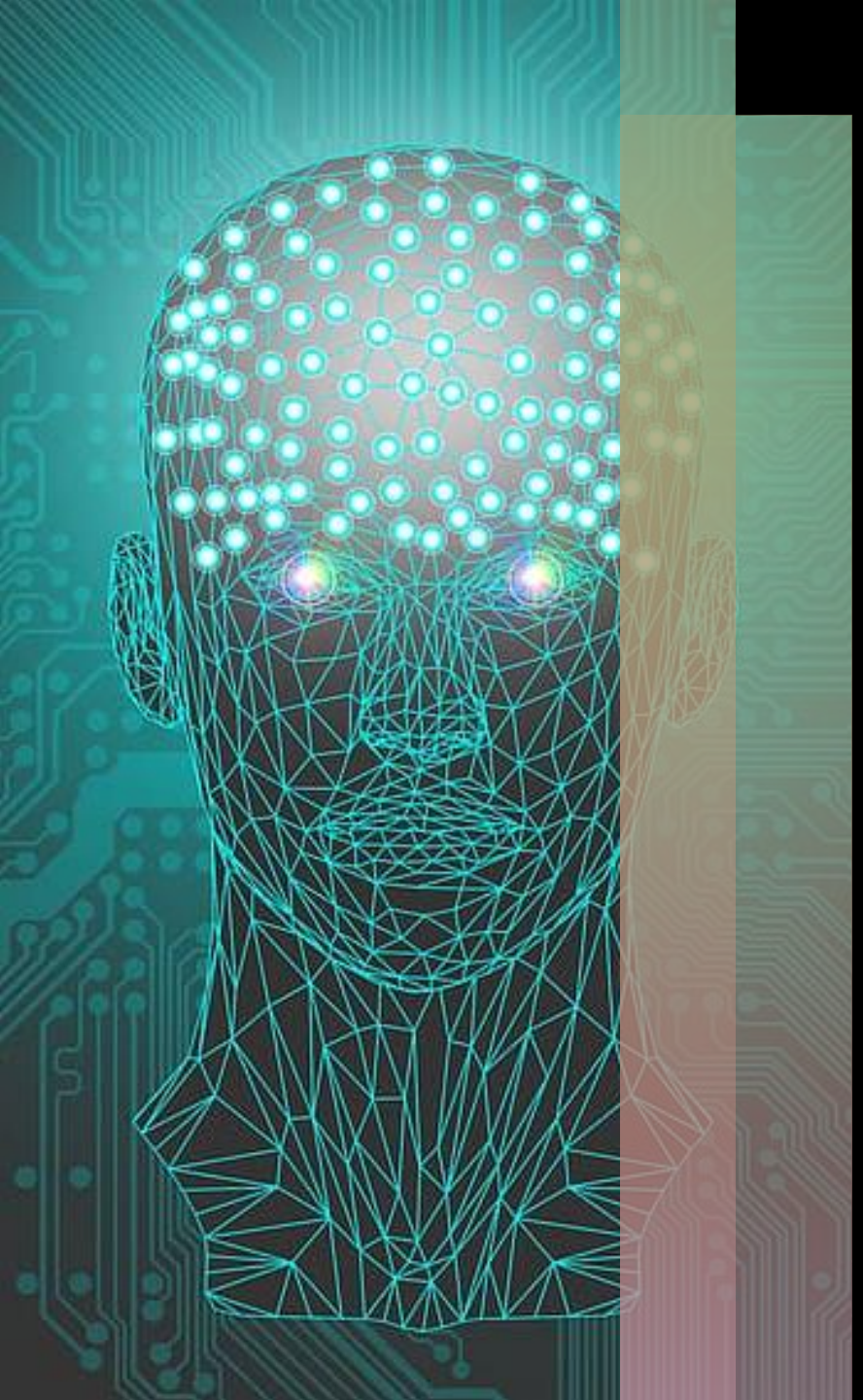
```
void ReadFile( ifstream& infile )
{
// This is a formatting command. It says do not skip
// white space when reading from the input stream
  infile >> noskipws;
  while( !infile.eof() )
  {
     char ch;
     infile >> ch;
     if( !infile.eof() )
     {
        cout << ch;
     }
  }
}
```



Handling End-of-File Main

```
int main( )
{
  ifstream infile;
  OpenFile( infile );
  ReadFile( infile);
}
```

- You could change the way these functions work in your own code.

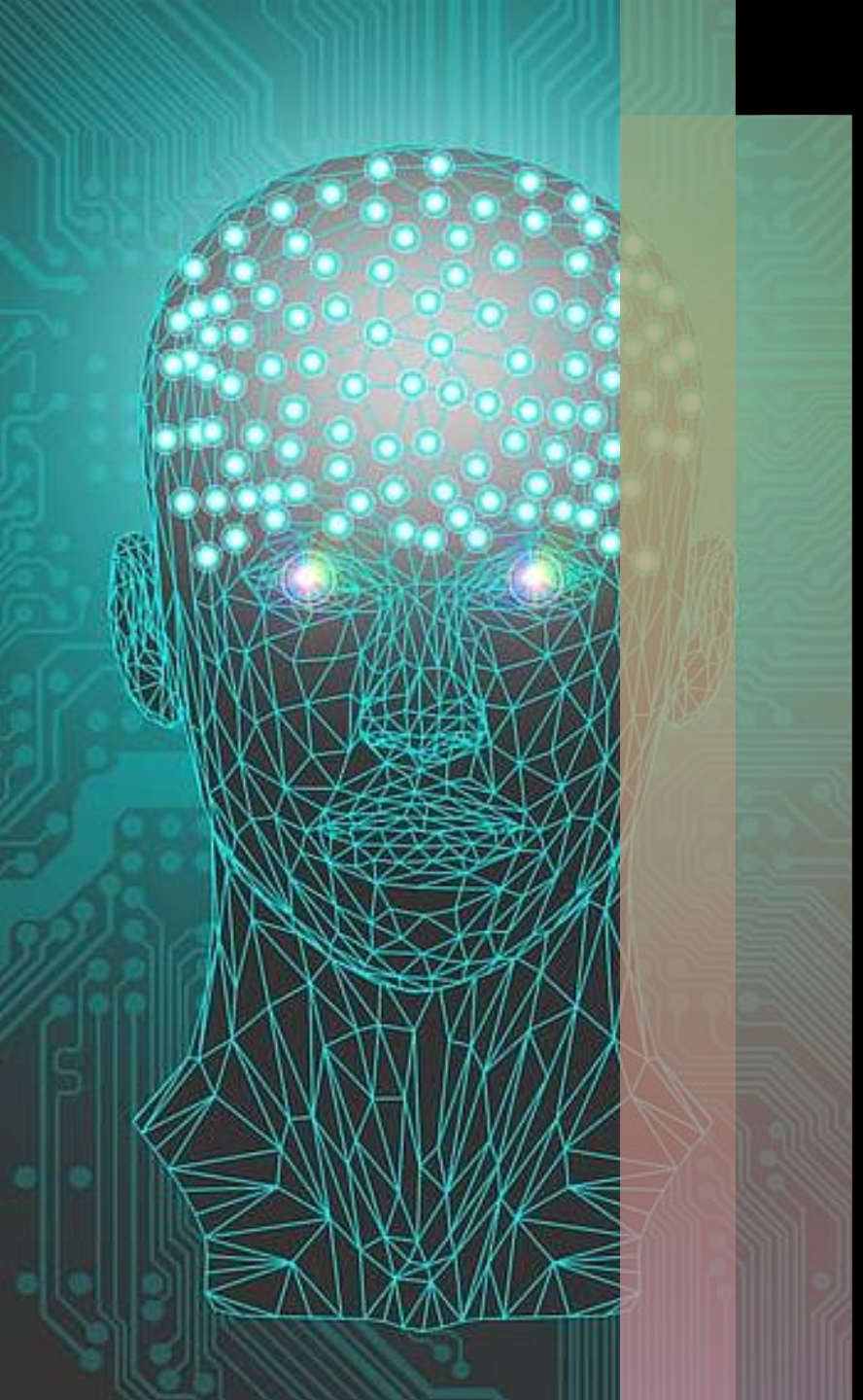


Writing to a File

- Use ofstream for file output (in main):

```
ofstream outfile( "output.txt" );
if ( !outfile ) // The file handle equates to a boolean
{
    cout << "Cannot open output file" << endl;
    exit(1);
}
outfile << "Written to the file" << endl;
outfile.close( );
```

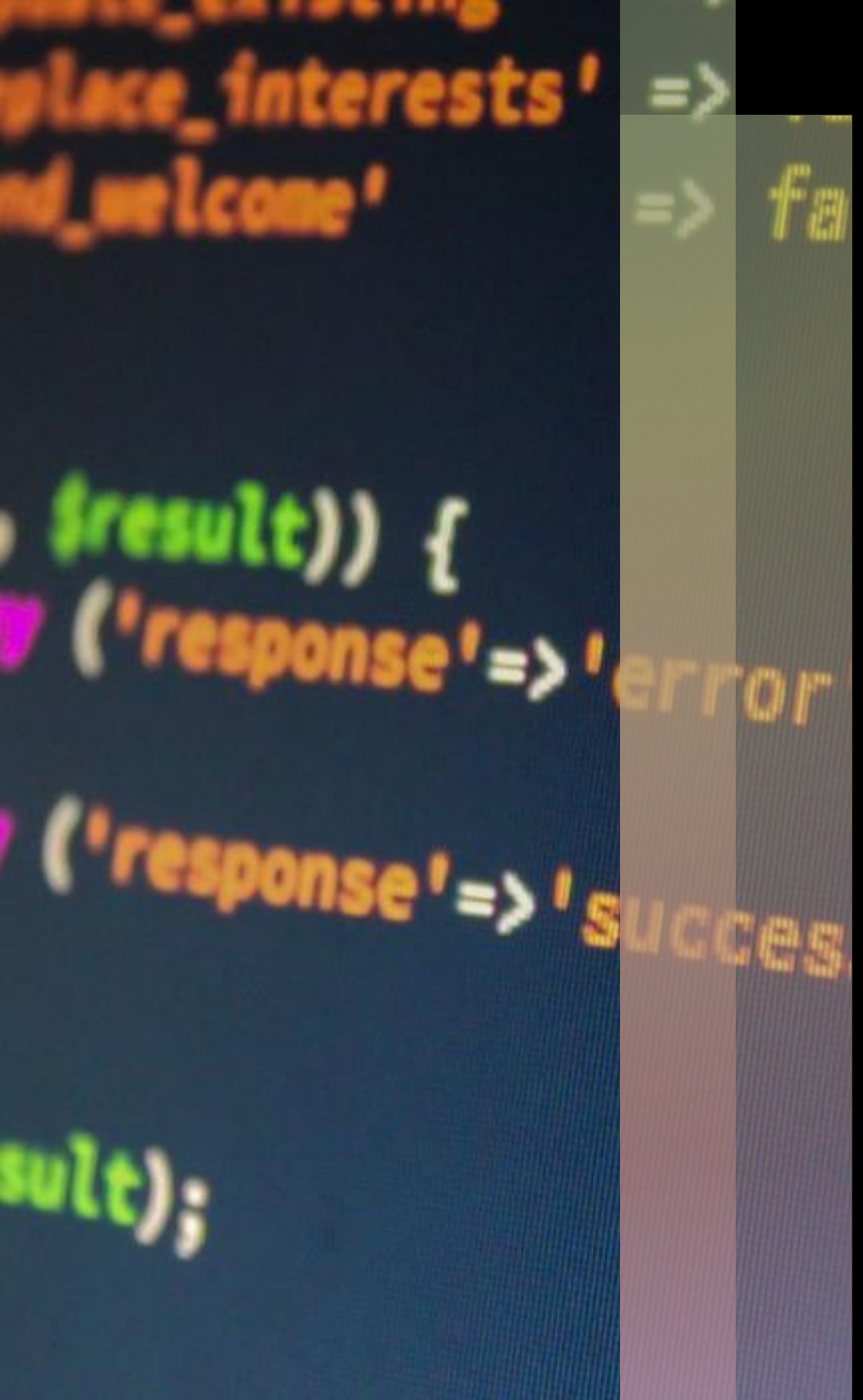
- check if the file is successfully opened before writing to it.
- Close the output file after writing.



Function Prototypes

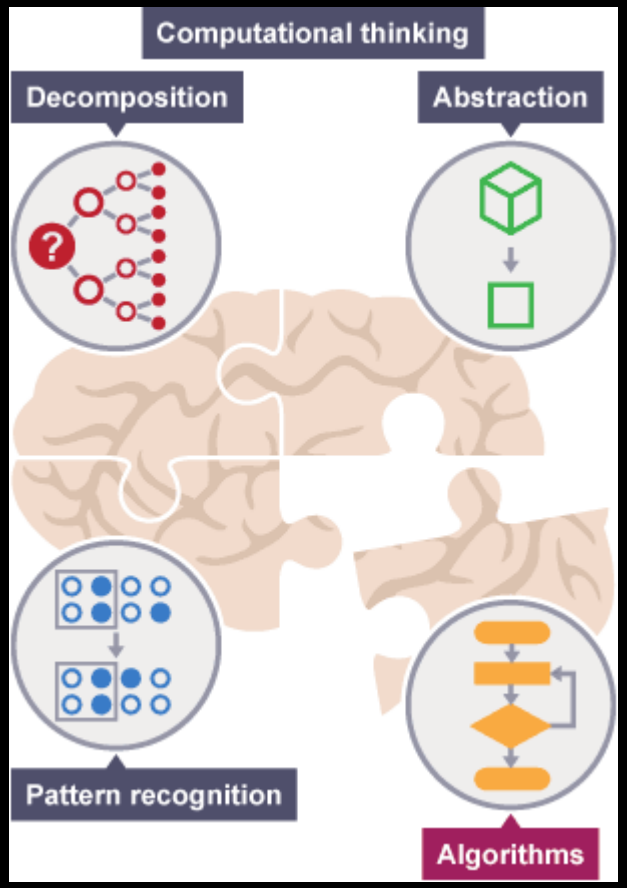
- Two separate bits of code:
  - The function prototype
  - The function definition
- The prototype must occur before the function is used.
- The definition can then be provided anywhere in the code.

```
void ReferanceParameter(int& number); // Prototype

int main() // Can now define ReferanceParameter() after
{          // the main function
}

void ReferanceParameter( int& number ) // Definition
{
  number = 30; // original number value is changed
}
```



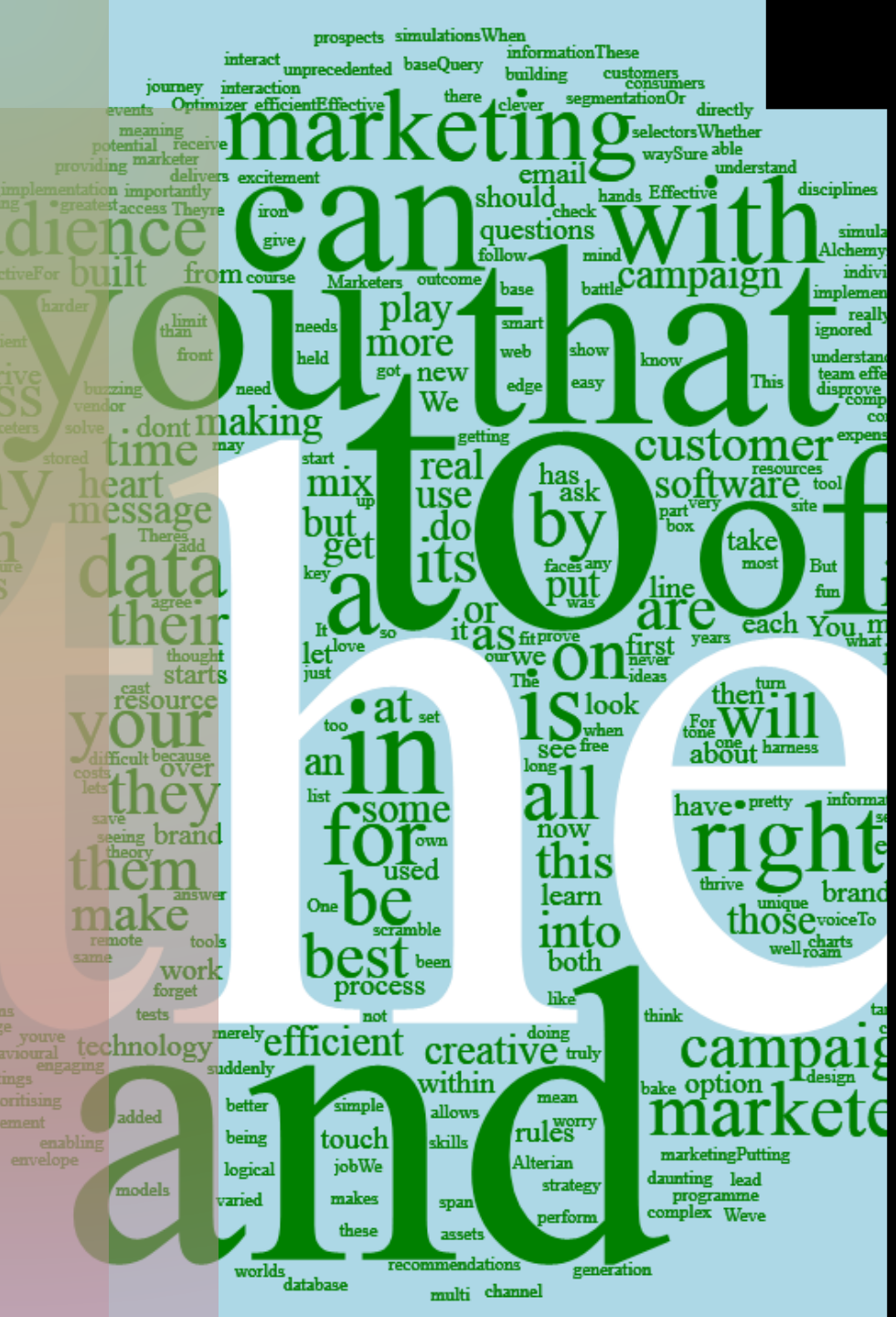
What is an Algorithm?

- "An **algorithm** is a plan, a set of **step-by-step** instructions to solve a problem. If you can tie shoelaces, make a cup of tea, get dressed or prepare a meal then you already know how to follow an algorithm" (What is an algorithm?).



Other Algorithms

- Sorting algorithms for organising data.
  - Bubble Sort
  - Quick Sort
  - Insertion Sort
  - Merge Sort
- Search algorithms for finding information quickly.
  - Linear Search
  - Binary Search
  - Hashing
- Encryption algorithms for secure data transmission.
  - Advanced Encryption Standard (AES)
  - Rivest Cipher (RSA)
  - Elliptic Curve Cryptography (ECC)



Example Algorithm 1

- Bubble Sort Algorithm
- Written in **pseudocode**
- Good programmers can write solutions in pseudocode and then find that they work when they implement them

```
procedure bubbleSort(A : list of sortable items)
    n = length(A)
    repeat
        newn = 0
        for i = 1 to n-1 inclusive do
            if A[i-1] > A[i] then
                swap(A[i-1], A[i])
                newn = i
            end if
        end for
        n = newn
    until n = 0
end procedure
```

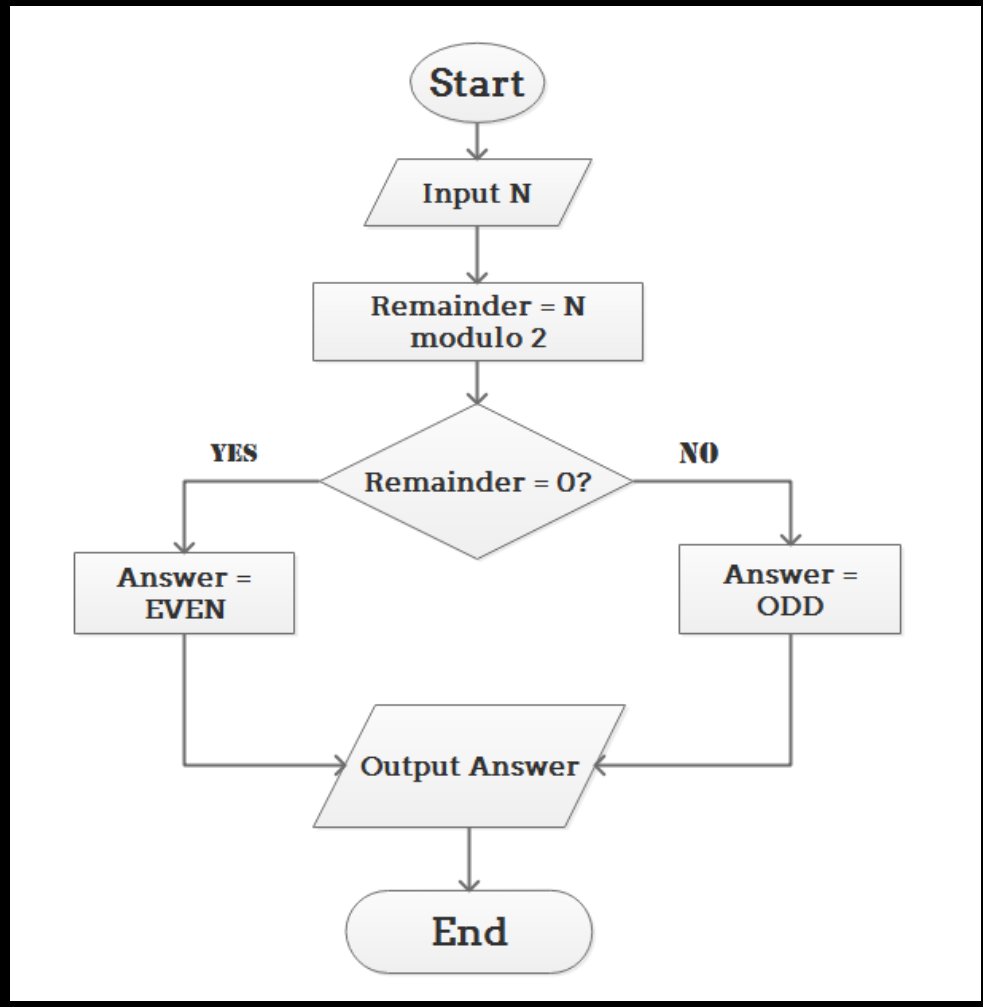
Code written from Pseudocode

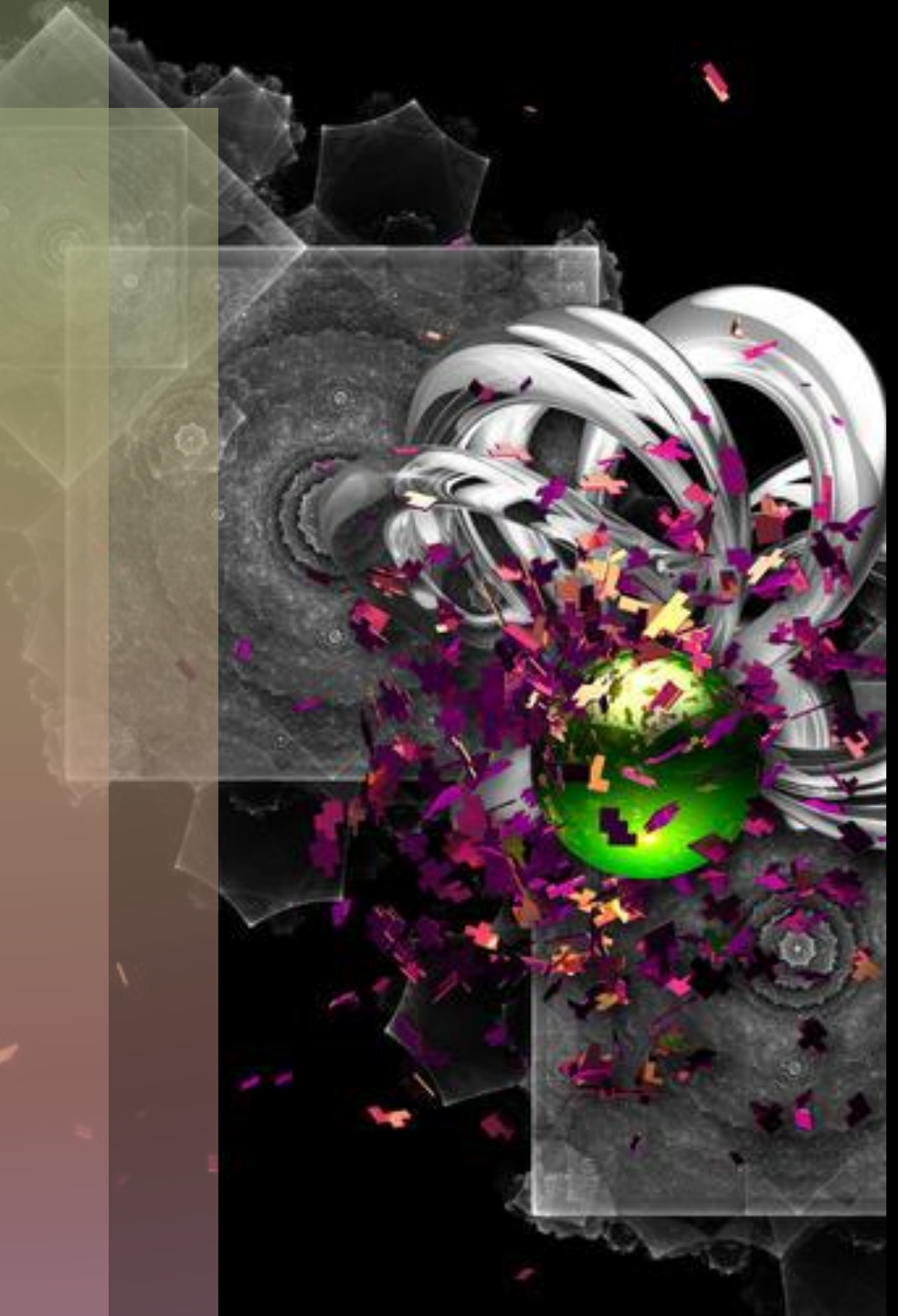
```
BFS(G, s)
1  for each vertex u ∈ V(G) \ {s}
2       color[u] = white
3       d[u] = ∞
4       π[u] = nil
5  color[s] = gray
6  d[s] = 0
7  π[s] = nil
8  Q = ∅
9  Enqueue(Q, s)
10 while Q ≠ ∅
11      u = Dequeue(Q)
12      for each v ∈ Adj[u]
13           if color[v] == white
14                color[v] = gray
15                d[v] = d[u] +
16                π[v] = u
17                Enqueue(Q, v)
18      color[u] = black
```

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

int main()
{
    int a[6] = { 4,1,6,2,8,0 };
    int n = 6;
    while (n != 0)
    {
        int newn = 0;
        for (int i = 1; i <= n-1; i++)
        {
            if (a[i-1] > a[i])
            {
                swap(a[i - 1], a[i]);
                newn = i;
            }
        }
        n = newn;
    }
}
```

Example Algorithm 2

- Odd and Even Function Algorithm
- Flowchart representation





Code written from Flowchart

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

// Function returns string, takes in integer parameter
// If no return then specify func return type as void
string OddOrEven(int n)
{
    int remainder = n % 2;
    // ternary operator
    string result = (remainder == 0) ? "even" : "odd";
    return result;
}

int main()
{
    int a[6] = { 4,1,6,2,8,0 };

    for (int i : a) // for i in range a (iterable)
    {
        cout << i << " : " << OddOrEven(i) << endl;
    }
}
```

What is printed out here?

What is Computational Thinking?

"Computational thinking allows us to take a **complex problem**, understand what the problem is and develop possible solutions. We can then present these solutions in a way that a computer, a human, or both, can understand."
- What is computational thinking?

---- Key Components of Computational Thinking ----

Decomposition - Breaking down a problem into smaller sub-problems

Pattern Recognition - Identifying trends or similarities within data

Abstraction - Extracting essential details while ignoring irrelevant information

Algorithm Design - Creating step-by-step solutions to problems

What Makes for a Good Algorithm?

A good algorithm is a precise set of instructions that, when followed, solves a specific problem.

Algorithms must have the following properties...

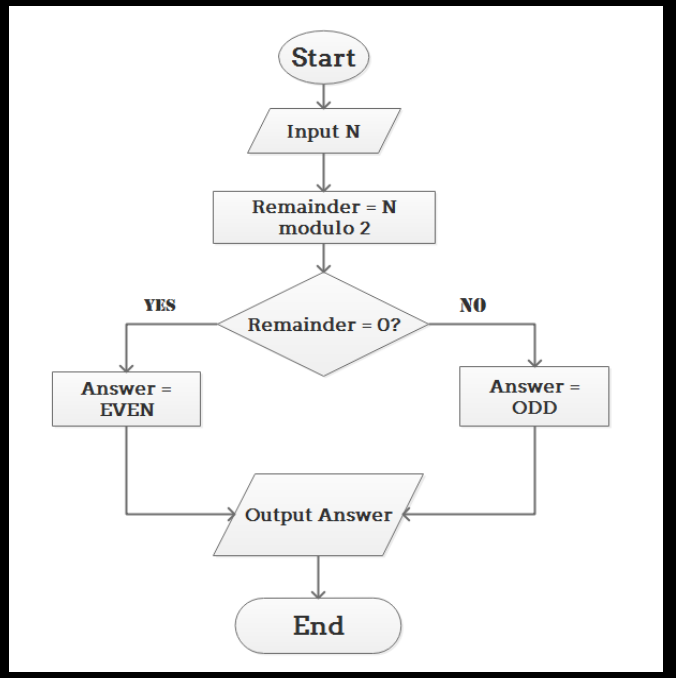
- **Accuracy** - The algorithm should produce correct results.
- **Efficiency** - The algorithm should perform the task in a reasonable amount of time.
- **Modularity** - Breaking down complex problems into smaller, manageable components.

Components of a Good Algorithm

Components of an Algorithm:

- **Input** - Information provided to the algorithm.
- **Output** - The result produced by the algorithm.
- **Control Structures** - Decision-making (if-else) and looping (for, while) structures.
- **Operations** - Actions performed on the input to produce the desired output.

```
procedure bubbleSort(A : list of sortable items)
   n = length(A)
   repeat
      newn = 0
      for i = 1 to n-1 inclusive do
         if A[i-1] > A[i] then
            swap(A[i-1], A[i])
            newn = i
         end if
      end for
      n = newn
   until n = 0
end procedure
```



Problem Solving Strategies

1. Clearly state the requirements of your problem and potential solutions
2. Begin by defining the inputs, outputs and constraints
3. Break down the problem into smaller, more manageable sub-problems
4. Identify patterns (repeated aspects) or similarities to develop a systematic approach
5. Translate the plan into a step-by-step algorithm (I prefer pseudocode)
6. Use programming languages to express the algorithm in a language computers understand
7. Test the algorithm with different inputs to ensure correctness
8. Refine the algorithm based on feedback and optimisation considerations
    1. We will discuss this a couple of weeks

Pseudocode

- Pseudocode is a high-level, plain-language representation of program logic before actual coding.
- It helps us to plan out the structure and flow of a program or algorithm.
- It mainly uses plain and informal constructs.
  - This means that the solutions we write are not language specific.
- Using problem solving strategies you can build a specification for your algorithm, which can be used to write pseudocode.
- Started by typing what you want your code to do in words.
- With each line initiating a block or a statement.

Guidelines for Pseudocode

- Start with the main routine where program execution begins.
- Use standard constructs like loops, conditionals, and subroutines.
- Clearly define variables with meaningful names.
- Describe input and output using standard I/O operations.
- Use Indentation to enhance code structure and readability.
- Add comments for explanation or additional information.
- Focus on logic over specific syntax.

Pseudocode Language

Keyword	Description
BEGIN / END	Marks the beginning and end of the main routine.
DECLARE / SET	Declares variables or sets their initial values.
IF / THEN / ELSE / END IF	Represents conditional statements.
FOR / TO / DO / END FOR	Describes a loop that iterates over a range.
WHILE / DO / END WHILE	Describes a loop that continues based on a condition.
READ / INPUT	Represents input operations.
WRITE / OUTPUT	Represents output operations.
FUNCTION / PROCEDURE / END FUNCTION / END PROCEDURE	Describes functions or procedures.
RETURN / CALL / INVOKE	Represents return of a value or a function call.
EXIT / CASE / OF / END CASE	Describes an exit or a switch/case statement.
AND / OR / NOT	Logical operators.
MOD / DIV	Arithmetic operators for modulo and division.
INCREMENT / DECREMENT	Represents incrementing or decrementing a variable.
COMMENT	Indicates that the following text is a comment.

Example Pseudocode

```
PROCEDURE BubbleSort(INPUT: LIST OF SORTABLE ITEMS)
    n := LENGTH(INPUT)
    REPEAT newn := 0
        FOR i FROM 1 TO LENGTH(INPUT) - 1 DO
            IF INPUT[i - 1] > INPUT[i] THEN
                // Swap if the current element is greater than the next one
                SWAP(INPUT[i - 1], INPUT[i])
                // Set newn to 1 indicating a swap was made
                newn := 1
            END IF
        END FOR
        // Update n with the value of newn
        n := newn
    UNTIL n = 0
END PROCEDURE

// := is less ambiguous as it means equal
// = mean equals to (==) and is reserved for conditions
```

I would like you to implement the Swap function. To swap two integer array elements. Also write pseudocode to test in main.

Keyword	Description
BEGIN / END	Marks the beginning and end of the main routine.
DECLARE / SET	Declares variables or sets their initial values.
IF / THEN / ELSE / END IF	Represents conditional statements.
FOR / TO / DO / END FOR	Describes a loop that iterates over a range.
WHILE / DO / END WHILE	Describes a loop that continues based on a condition.
READ / INPUT	Represents input operations.
WRITE / OUTPUT	Represents output operations.
FUNCTION / PROCEDURE / END FUNCTION / END PROCEDURE	Describes functions or procedures.
RETURN / CALL / INVOKE	Represents return of a value or a function call.
EXIT / CASE / OF / END CASE	Describes an exit or a switch/case statement.
AND / OR / NOT	Logical operators.
MOD / DIV	Arithmetic operators for modulo and division.
INCREMENT / DECREMENT	Represents incrementing or decrementing a variable.
COMMENT	Indicates that the following text is a comment.

Answer for Swap Pseudocode

```
PROCEDURE Swap(INPUT: INTEGER ELEMENT_A, INTEGER ELEMENT_B)
    DECLARE TEMP := ELEMENT_B
    ELEMENT_B := ELEMENT_A
    ELEMENT_A := TEMP
END PROCEDURE

PROCEDURE main()
    // Example usage in the main part of the program
    A := 5
    B := 10

    WRITE "Before Swap: A =", A, ", B =", B // Print in main

    CALL Swap(A, B)

    WRITE "After Swap: A =", A, ", B =", B // Print in main after swap
END PROCEDURE
```

Summary

We've covered quite a few topics today:

- Introducing algorithms and c++

Any questions?