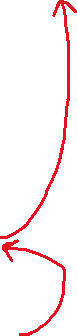
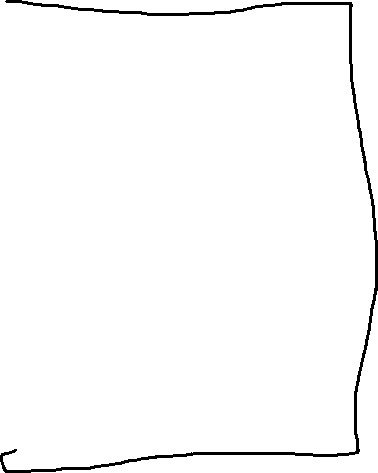
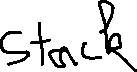
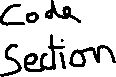
**Section 2: Essential C and C++ Concepts**

**3. Array Basics:**

* Collection of similar data.

**7. Pointers:**

* Pointer is an address variable used to store address of a data.
* Memory is divided as follows:



* Pointers are used for:
  + Accessing heap
  + Accessing resources
  + Parameter passing
* Declare pointer:   
  int\* pointerToAnInteger;  
  pointerToAnInteger = &anIntegerVariable; // & - Address of  
  cout << \*pointerToAnInteger; // \* - Dereference operator
* To get memory in heap: pointer = new int[5];
* All pointers regardless of what datatype the point to, take up the same space (usually 8 bytes).

**9. Reference in C++**

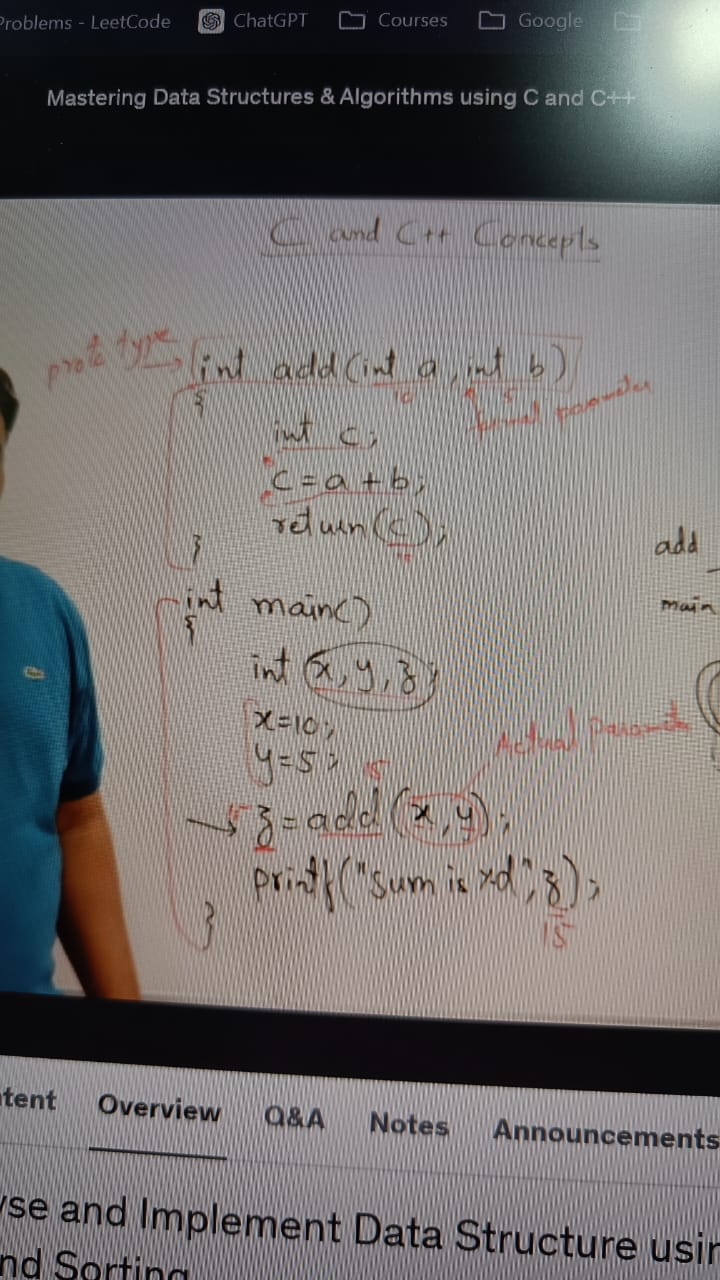
* Alias given to a variable.
* How?  
  int a = 10;  
  int &r = a;

Now “r” can be used in place of a.

* Useful in parameter passing.

**13. Functions:**

* Function is a piece of code (related instructions) that performs a specific task.
* Used for modular/procedural programming.



**15. Parameter Passing Methods**

#include <bits/stdc++.h>

using namespace std;

void swap\_byValue(int a, int b) {

    int temporary = a;

    a = b;

    b = temporary;

}

void swap\_byAddress(int\* a, int \*b) {

    int temporary = \*a;

    \*a = \*b;

    \*b = temporary;

}

void swap\_byReference(int& a, int& b) {

    int temporary = a;

    a = b;

    b = temporary;

}

int main() {

    int a = 1, b = 2;

    swap\_byValue(a, b); //Doesn't affect a and b

    cout << a << " " << b << endl;

    swap\_byAddress(&a, &b); //Affects a and b as addresses are passed; Values at addresses are swapped.

    cout << a << " " << b << endl;

    swap\_byReference(a, b); // Affects a and b as their references are passed

    cout << a << " " << b << endl;

}

**17. Array as Parameter**

* Arrays are always passed as address.  
  int function(int array[]) {}  
  int function(int\* array) {}
* Return array:  
  int[] function(int n) {return pointer}  
  int\* function(int n) {return pointer}

#include <bits/stdc++.h>

using namespace std;

void displayArray(int\* array, int size) {

    for(int i = 0; i < size; i++) {

        cout << array[i] << " ";

    }

    cout << endl;

}

int\* declareArray(int size) {

    return new int[size];

}

int main() {

    int array1[] = {1, 2, 3, 4, 5}, size = 5;

    displayArray(array1, size); //passing array as parameter

    int\* pointerTo\_anArray = declareArray(size); //returns an array pointer

    pointerTo\_anArray[0] = 10;

    displayArray(pointerTo\_anArray, size);

}

**26. Practice: Object Oriented Programming**

class Rectangle {

    private:

        int length;

        int breadth;

    public:

        Rectangle(int length, int breadth) {

            this -> length = length;

            this -> breadth = breadth;

        }

        int area() {

            return length \* breadth;

        }

        int perimeter() {

            return 2 \* (length + breadth);

        }

};

int main() {

    Rectangle rectangle1(10, 5);

    cout << rectangle1.area() << endl;

    cout << rectangle1.perimeter() << endl;

}

**30. Practice: Template Class**

template <class dataType> class Rectangle {

    private:

        dataType length;

        dataType breadth;

    public:

        Rectangle(dataType length, dataType breadth) {

            this -> length = length;

            this -> breadth = breadth;

        }

        dataType area() {

            return length \* breadth;

        }

        dataType perimeter() {

            return 2 \* (length + breadth);

        }

};