Great. Starting now, I'll make **each C++ program separately**, using **simple syntax**, as per your list.

✅ 1. File Read / Write – All Basic Operations

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

int main() {
    // Writing to file
    ofstream writeFile("sample.txt");
    if (writeFile.is_open()) {
        writeFile << "Hello, File!\n";
        writeFile << "Second line.\n";
        writeFile.close();
    }

    // Reading from file
    ifstream readFile("sample.txt");
    string line;
    if (readFile.is_open()) {
        while (getline(readFile, line)) {
            cout << line << endl;
        }
        readFile.close();
    }

    return 0;
}
```

✅ 2. Modes to Write into File

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

int main() {
    // Overwrite mode
    ofstream out1("file1.txt");  // default: ios::out
    out1 << "This will overwrite.\n";
    out1.close();

    // Append mode
    ofstream out2("file1.txt", ios::app);
    out2 << "This will append.\n";
    out2.close();

    return 0;
}
```

✅ 3. Exception and Custom Exception

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

class MyException {
public:
    void display() {
        cout << "Custom Exception caught!" << endl;
    }
};

int main() {
    try {
        int x = 0;
        if (x == 0) throw MyException();
    }
    catch (MyException e) {
        e.display();
    }

    return 0;
}
```

✅ 4. Friend Class and Friend Function

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

class B;  // Forward declaration

class A {
    int num;
public:
    A() { num = 100; }
    friend void show(A);
    friend class B;
};

void show(A a) {
    cout << "Friend Function: " << a.num << endl;
}

class B {
public:
    void display(A a) {
        cout << "Friend Class: " << a.num << endl;
    }
};

int main() {
    A obj;
    B b;
    show(obj);
    b.display(obj);
    return 0;
}
```

I'll continue with **inheritance** next.

✅ 5. Inheritance

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

class Animal {
public:
    void sound() {
        cout << "Animal makes a sound" << endl;
    }
};

class Dog : public Animal {
public:
    void bark() {
        cout << "Dog barks" << endl;
    }
};

int main() {
    Dog d;
    d.sound();  // inherited
    d.bark();
    return 0;
}
```

✅ 6. Virtual Function & Pure Virtual Function

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

class Shape {
public:
    virtual void draw() {
        cout << "Drawing Shape" << endl;
    }
    virtual void show() = 0;  // pure virtual function
};

class Circle : public Shape {
public:
    void draw() {
        cout << "Drawing Circle" << endl;
    }

    void show() {
        cout << "Showing Circle Info" << endl;
    }
};

int main() {
    Shape* s;
    Circle c;
    s = &c;
    s->draw();
    s->show();
```

```
    return 0;
}
```

✅ 7. String Class – All Functions (Basic)

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

int main() {
    string s1 = "Hello";
    string s2 = "World";

    // Concatenation
    string s3 = s1 + " " + s2;
    cout << "Concatenated: " << s3 << endl;

    // Length
    cout << "Length: " << s3.length() << endl;

    // Access character
    cout << "First char: " << s3[0] << endl;

    // Substring
    cout << "Substring (0-5): " << s3.substr(0, 5) << endl;

    // Find
    cout << "'World' at: " << s3.find("World") << endl;

    return 0;
}
```

✅ 8. Operator Overloading (Unary and Binary)

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

class Count {
    int value;
public:
    Count(int x = 0) { value = x; }

    // Unary operator overloading
    void operator++() {
        value++;
    }

    // Binary operator overloading
    Count operator+(Count c) {
        return Count(value + c.value);
    }

    void display() {
        cout << "Value: " << value << endl;
    }
};
```

```
int main() {
    Count c1(5), c2(10), c3;
    ++c1;
    c3 = c1 + c2;
    c1.display();
    c3.display();
    return 0;
}
```

✅ 9. Function Overload and Override

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

class A {
public:
    void show(int x) {
        cout << "Show int: " << x << endl;
    }
    void show(string s) {
        cout << "Show string: " << s << endl;
    }
};

class B : public A {
public:
    void show(int x) {  // overriding
        cout << "B's show: " << x << endl;
    }
};

int main() {
    B b;
    b.show(10);        // override
    b.A::show("Hi");   // overload from base
    return 0;
}
```

✅ 10. Operator Overloading using Friend Function

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

class Complex {
    int real, imag;
public:
    Complex(int r = 0, int i = 0) {
        real = r; imag = i;
    }

    friend Complex operator+(Complex c1, Complex c2);

    void display() {
        cout << real << " + " << imag << "i" << endl;
    }
};
```

```
Complex operator+(Complex c1, Complex c2) {
    return Complex(c1.real + c2.real, c1.imag + c2.imag);
}

int main() {
    Complex a(3, 2), b(1, 7), c;
    c = a + b;
    c.display();
    return 0;
}
```

✅ 11. Operator Overloading by the Friend Class

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

class B;  // Forward declaration

class A {
    int data;
public:
    A(int d) { data = d; }
    friend class B;
};

class B {
public:
    A operator+(A x, A y) {
        return A(x.data + y.data);
    }

    void display(A a) {
        cout << "Value: " << a.data << endl;
    }
};

int main() {
    A a1(10), a2(20);
    B b;
    A result = b + a1 + a2;
    b.display(result);
    return 0;
}
```

All C++ programs from your image are now complete, each written separately using **standard syntax only** — no vectors, templates, or STL.

Let me know if you want each saved as a `.cpp` file or want to move on to another topic (like DSA, OOP, etc.).

✅ 1. Fibonacci Number – Generate Sequence

```
#include <iostream>
```

```
using namespace std;

int main() {
    int n, a = 0, b = 1, c;
    cout << "Enter number of terms: ";
    cin >> n;

    cout << "Fibonacci Sequence: ";
    for (int i = 0; i < n; i++) {
        cout << a << " ";
        c = a + b;
        a = b;
        b = c;
    }

    return 0;
}
```

✅ 2. Prime Number – Check and Generate Sequence

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

bool isPrime(int num) {
    if (num < 2) return false;
    for (int i = 2; i <= num / 2; i++)
        if (num % i == 0) return false;
    return true;
}

int main() {
    int n;
    cout << "Enter upper limit: ";
    cin >> n;

    cout << "Prime Numbers: ";
    for (int i = 2; i <= n; i++)
        if (isPrime(i)) cout << i << " ";

    return 0;
}
```

✅ 3. Armstrong Number

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

int main() {
    int num, temp, digit, sum = 0;
    cout << "Enter a number: ";
    cin >> num;

    temp = num;
    while (temp > 0) {
        digit = temp % 10;
```

```
        sum += pow(digit, 3);
        temp /= 10;
    }

    if (sum == num)
        cout << "Armstrong number";
    else
        cout << "Not an Armstrong number";

    return 0;
}
```

✅ 4. Palindrome Number

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

int main() {
    int num, rev = 0, rem, orig;
    cout << "Enter a number: ";
    cin >> num;

    orig = num;
    while (num > 0) {
        rem = num % 10;
        rev = rev * 10 + rem;
        num /= 10;
    }

    if (rev == orig)
        cout << "Palindrome number";
    else
        cout << "Not a palindrome";

    return 0;
}
```

✅ 5. Perfect Number

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

int main() {
    int num, sum = 0;
    cout << "Enter a number: ";
    cin >> num;

    for (int i = 1; i < num; i++) {
        if (num % i == 0)
            sum += i;
    }

    if (sum == num)
        cout << "Perfect number";
    else
        cout << "Not a perfect number";
```

```
    return 0;
}
```

✅ 6. HCF and LCM

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

int main() {
    int a, b, hcf = 1;
    cout << "Enter two numbers: ";
    cin >> a >> b;

    for (int i = 1; i <= a && i <= b; i++) {
        if (a % i == 0 && b % i == 0)
            hcf = i;
    }

    int lcm = (a * b) / hcf;

    cout << "HCF = " << hcf << endl;
    cout << "LCM = " << lcm << endl;

    return 0;
}
```

✅ 7. Linear Search on Array

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

int main() {
    int arr[100], n, key;
    cout << "Enter number of elements: ";
    cin >> n;

    cout << "Enter elements:\n";
    for (int i = 0; i < n; i++)
        cin >> arr[i];

    cout << "Enter element to search: ";
    cin >> key;

    for (int i = 0; i < n; i++) {
        if (arr[i] == key) {
            cout << "Found at index " << i;
            return 0;
        }
    }

    cout << "Element not found.";
    return 0;
}
```

✅ 8. Binary Search on Array

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

int main() {
    int arr[100], n, key;
    cout << "Enter number of elements (sorted): ";
    cin >> n;

    cout << "Enter elements:\n";
    for (int i = 0; i < n; i++)
        cin >> arr[i];

    cout << "Enter element to search: ";
    cin >> key;

    int low = 0, high = n - 1, mid;
    while (low <= high) {
        mid = (low + high) / 2;

        if (arr[mid] == key) {
            cout << "Found at index " << mid;
            return 0;
        } else if (arr[mid] < key)
            low = mid + 1;
        else
            high = mid - 1;
    }

    cout << "Element not found.";
    return 0;
}
```

✅ 9. Linear Sort (Bubble Sort)

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

int main() {
    int arr[100], n;
    cout << "Enter number of elements: ";
    cin >> n;

    cout << "Enter elements:\n";
    for (int i = 0; i < n; i++)
        cin >> arr[i];

    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j + 1])
                swap(arr[j], arr[j + 1]);

    cout << "Sorted array: ";
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";

    return 0;
```

```
}
```

✅ 10. Merge Sort

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

void merge(int arr[], int l, int m, int r) {
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    int L[100], R[100];

    for (i = 0; i < n1; i++) L[i] = arr[l + i];
    for (j = 0; j < n2; j++) R[j] = arr[m + 1 + j];

    i = 0; j = 0; k = l;

    while (i < n1 && j < n2)
        arr[k++] = (L[i] <= R[j]) ? L[i++] : R[j++];

    while (i < n1) arr[k++] = L[i++];
    while (j < n2) arr[k++] = R[j++];
}

void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}

int main() {
    int arr[100], n;
    cout << "Enter number of elements: ";
    cin >> n;

    cout << "Enter elements:\n";
    for (int i = 0; i < n; i++) cin >> arr[i];

    mergeSort(arr, 0, n - 1);

    cout << "Sorted array: ";
    for (int i = 0; i < n; i++) cout << arr[i] << " ";

    return 0;
}
```

✅ 11. Bucket Sort (For positive integers only)

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

```
int main() {
    int arr[100], n, max = 0;
    cout << "Enter number of elements: ";
    cin >> n;

    cout << "Enter elements (0 to 99): ";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
        if (arr[i] > max) max = arr[i];
    }

    int bucket[100] = {0};

    for (int i = 0; i < n; i++)
        bucket[arr[i]]++;

    cout << "Sorted array: ";
    for (int i = 0; i <= max; i++)
        while (bucket[i]--)
            cout << i << " ";

    return 0;
}
```

✅ All basic C++ programs from your latest list are now completed, **using only loops, arrays, and fundamental syntax** — nothing advanced or STL-based. Ready for whatever you want next.

✅ 1. Matrix Multiplication

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

int main() {
    int a[10][10], b[10][10], c[10][10];
    int m, n, p, q;

    cout << "Enter rows and columns of matrix A: ";
    cin >> m >> n;

    cout << "Enter rows and columns of matrix B: ";
    cin >> p >> q;

    if (n != p) {
        cout << "Matrix multiplication not possible.";
        return 0;
    }

    cout << "Enter elements of matrix A:\n";
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
            cin >> a[i][j];

    cout << "Enter elements of matrix B:\n";
    for (int i = 0; i < p; i++)
```

```
        for (int j = 0; j < q; j++)
            cin >> b[i][j];

    // Multiply
    for (int i = 0; i < m; i++)
        for (int j = 0; j < q; j++) {
            c[i][j] = 0;
            for (int k = 0; k < n; k++)
                c[i][j] += a[i][k] * b[k][j];
        }

    // Display result
    cout << "Result matrix:\n";
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < q; j++)
            cout << c[i][j] << " ";
        cout << endl;
    }

    return 0;
}
```

✅ 2. Namespace and Custom Namespace

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

namespace MySpace {
    void greet() {
        cout << "Hello from custom namespace!" << endl;
    }
}

int main() {
    MySpace::greet();  // Calling function from custom namespace
    return 0;
}
```

✅ 3. User Defined Header File

🔷 Step 1: Create header file – `myfunctions.h`

```
// myfunctions.h
#ifndef MYFUNCTIONS_H
#define MYFUNCTIONS_H

void greet();
int add(int, int);

#endif
```

🔷 Step 2: Create function file – `myfunctions.cpp`

```
// myfunctions.cpp
#include <iostream>
```

```
#include "myfunctions.h"
using namespace std;

void greet() {
    cout << "Welcome to My Custom Header!" << endl;
}

int add(int a, int b) {
    return a + b;
}
```

🔷 Step 3: Main file – `main.cpp`

```
// main.cpp
#include <iostream>
#include "myfunctions.h"
using namespace std;

int main() {
    greet();
    int result = add(5, 3);
    cout << "Sum = " << result << endl;
    return 0;
}
```

💡 Compile and run using:

```
g++ main.cpp myfunctions.cpp -o program
./program
```

All three topics — **matrix multiplication**, **custom namespace**, and **user-defined header file** — are now complete with clean C++ examples. Ready for anything else you want.