1.a) class student (accept & display for one object)

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

class Student {
public:
    int roll_no;
    string name;
    string cls;
    void accept() {
        cout << "Roll no: "; cin >> roll_no;
        cout << "Name: "; cin.ignore(); getline(cin, name);
        cout << "Class: "; getline(cin, cls);
    }
    void display() {
        cout << "Roll: " << roll_no << ", Name: " << name << ", Class: " << cls << endl;
    }
};

int main() {
    Student s;
    s.accept();
    s.display();
    return 0;
}
```

1.b) class Book (2 objects, display book with greater price)

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

class Book {
public:
    string name;
    double price;
    int pages;
    void accept() {
        cout << "Name: "; cin.ignore(); getline(cin, name);
        cout << "Price: "; cin >> price;
        cout << "Pages: "; cin >> pages;
    }
    void display() {
        cout << name << " | " << price << " | " << pages << endl;
    }
};

int main() {
    Book b1, b2;
    cout << "Book 1:\n"; b1.accept();
    cout << "Book 2:\n"; b2.accept();
    cout << "Higher priced book:\n";
    (b1.price >= b2.price ? b1 : b2).display();
    return 0;
}
```

1.c) class time (HH:MM:SS -> total seconds & display)

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

class Time {
public:
    int hh, mm, ss;
    void accept() {
        char c;
        cin >> hh >> c >> mm >> c >> ss; // read HH:MM:SS
    }
    int toSeconds() {
        return hh*3600 + mm*60 + ss;
    }
    void displaySeconds() {
        cout << "Total seconds: " << toSeconds() << endl;
    }
};

int main() {
    Time t;
    cout << "Enter time (HH:MM:SS): ";
    t.accept();
    t.displaySeconds();
    return 0;
}
```

2.a) array of city (5 cities - display highest population)

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

class City {
public:
    string name;
    long long population;
    void accept() { cout << "Name: "; cin >> name; cout << "Population: "; cin >>
population; }
};

int main() {
    const int N=5;
    City arr[N];
    for(int i=0;i<N;i++){ cout<<"City "<<i+1<<":\n"; arr[i].accept(); }
    int idx=0;
    for(int i=1;i<N;i++) if(arr[i].population>arr[idx].population) idx=i;
    cout<<"Highest population city: "<<arr[idx].name<<" ("<<arr[idx].population<<")\n";
    return 0;
}
```

2.b) class Account (10 accounts, interest 10% if balance>=5000)

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

class Account {
public:
    int accno;
    double balance;
    void accept() { cout<<"Acc no: "; cin>>accno; cout<<"Balance: "; cin>>balance; }
    void applyInterest() { if(balance>=5000) balance *= 1.10; }
    void display() { cout<<accno<<" : "<<balance<<endl; }
};

int main() {
    vector<Account> a(10);
    for(int i=0;i<10;i++){ cout<<"Account "<<i+1<<":\n"; a[i].accept(); }
    for(auto &ac: a) ac.applyInterest();
    cout<<"After interest:\n";
    for(auto &ac: a) ac.display();
    return 0;
}
```

2.c) class Staff (5 staff, display who are HOD)

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

class Staff {
public:
    string name, post;
    void accept() { cout<<"Name: "; cin.ignore(); getline(cin, name); cout<<"Post: ";
getline(cin, post); }
};

int main() {
    const int N=5;
    Staff s[N];
    for(int i=0;i<N;i++){ cout<<"Staff "<<i+1<<":\n"; s[i].accept(); }
    cout<<"HODs:\n";
    for(int i=0;i<N;i++) if(s[i].post=="HOD") cout<<s[i].name<<endl;
    return 0;
}
```

3) Pointer to object, 'this' pointer, nested class (book example, pointer display)

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

class Book {
public:
    string title;
    string author;
    double price;
    void accept() { cout<<"Title: "; cin.ignore(); getline(cin, title); cout<<"Author: ";
getline(cin, author); cout<<"Price: "; cin>>price; }
    void display() { cout<<title<<" by "<<author<<" : "<<price<<endl; }
    void showUsingThis() { cout<<"(this) Title: "<< this->title << endl; }
};

int main() {
    Book *p = new Book();
    p->accept();
    p->display();
    p->showUsingThis();
    delete p;
    return 0;
}
```

4.a) passing object as function argument (result1 & result2 average marks)

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

class Result {
public:
    double marks;
    void accept() { cout<<"Marks: "; cin>>marks; }
    double get() const { return marks; }
};

double average(const Result &r1, const Result &r2) {
    return (r1.get() + r2.get())/2.0;
}

int main() {
    Result a,b;
    cout<<"Result 1:\n"; a.accept();
    cout<<"Result 2:\n"; b.accept();
    cout<<"Average: "<<average(a,b)<<endl;
    return 0;
}
```

4.b) find greatest among two numbers from two different classes using friend

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

class A;
class B;

class A { int x; public: void set(int v){x=v;} friend int greatest(const A&, const B&); };
class B { int y; public: void set(int v){y=v;} friend int greatest(const A&, const B&); };

int greatest(const A &a, const B &b) {
    return (a.x > b.y) ? a.x : b.y;
}

int main(){
    A a; B b; a.set(12); b.set(9);
    cout<<"Greatest: "<<greatest(a,b)<<endl;
    return 0;
}
```

4.c) swap contents of two variables of same class using friend

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

class Number {
    int val;
public:
    Number(int v=0): val(v) {}
    void display(){ cout<<val<<endl; }
    friend void swapNumber(Number&, Number&);
};

void swapNumber(Number &a, Number &b){
    int t=a.val; a.val=b.val; b.val=t;
}

int main(){
    Number n1(10), n2(20);
    swapNumber(n1,n2);
    n1.display(); n2.display();
    return 0;
}
```

5.a) constructor to sum numbers 1..n (value n passed to constructor)

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

class Sum {
    int total;
public:
    Sum(int n) {
        total = 0;
        for(int i=1;i<=n;i++) total += i;
    }
    void display(){ cout<<total<<endl; }
};

int main(){
    int n; cout<<"n: "; cin>>n;
    Sum s(n); s.display();
    return 0;
}
```

5.b) class student with constructor init name & percentage

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

class Student {
    string name;
    double percentage;
public:
    Student(string n, double p): name(n), percentage(p) {}
    void display(){ cout<<name<<" : "<<percentage<<endl; }
};

int main(){
    Student s("Rahul", 78.5);
    s.display();
    return 0;
}
```

5.c) class College with default course value (2 objects)

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

class College {
public:
    int roll;
    string name;
    string course;
    College(int r, string n, string c="Computer Engineering"): roll(r), name(n), course(c)
{}
    void display(){ cout<<roll<<" "<<name<<" "<<course<<endl; }
};

int main(){
    College c1(1,"A"); College c2(2,"B","IT");
    c1.display(); c2.display();
    return 0;
}
```

5.d) constructor overloading demo

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

class Demo {
public:
    Demo() { cout<<"Default\n"; }
    Demo(int x) { cout<<"With int: "<<x<<"\n"; }
    Demo(double x) { cout<<"With double: "<<x<<"\n"; }
};

int main(){
    Demo d1; Demo d2(5); Demo d3(3.14);
    return 0;
}
```

6) Inheritance examples (multilevel, multiple, hierarchical, hybrid, virtual base) - concise demos

```
// Multilevel example (single file)
#include <iostream>
using namespace std;
class A{ public: void f(){ cout<<"A\n"; } };
class B: public A{};
class C: public B{};
int main(){ C c; c.f(); return 0; }
```

7) Compile-time polymorphism: overloading and unary operator examples

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

int area(int l,int b){ return l*b; }
int area(int s){ return s*s; }
double sum(double a,double b,double c,double d,double e){ return a+b+c+d+e; }
int sum(int arr[], int n){ int s=0; for(int i=0;i<n;i++) s+=arr[i]; return s; }

// Unary operator negation and ++ overloading (class)
class Num { public: int v; Num(int x=0):v(x){} Num operator-(){ return Num(-v); } Num
operator++(){ return Num(++v); } };

int main(){ cout<<area(5,4)<<" "<<area(3)<<" "<<sum(1,2,3,4,5)<<"\n"; Num n(5); Num m =
-n; Num p = ++n; cout<<m.v<<" "<<p.v<<endl; return 0; }
```

8.a) overload + for strings

```
#include <iostream>
#include <string>
using namespace std;
class MyString {
public:
    string s;
    MyString(string t=""): s(t) {}
    MyString operator+(const MyString &o) const { return MyString(s + o.s); }
    void show(){ cout<<s<<endl; }
};
int main(){ MyString a("xyz"), b("pqr"); MyString c = a + b; c.show(); return 0; }
```

8.b) ILogin base with EmailLogin and MembershipLogin

```
#include <iostream>
#include <string>
using namespace std;
class ILogin {
public:
    string name, password;
    virtual void accept() = 0;
    virtual void display() = 0;
    virtual ~ILogin(){}
};
class EmailLogin: public ILogin {
public:
    void accept(){ cout<<"Email name: "; cin>>name; cout<<"Pass: "; cin>>password; }
    void display(){ cout<<"Email: "<<name<<" / "<<password<<endl; }
};
class MembershipLogin: public ILogin {
public:
    void accept(){ cout<<"Member name: "; cin>>name; cout<<"Pass: "; cin>>password; }
    void display(){ cout<<"Member: "<<name<<" / "<<password<<endl; }
};
int main(){ ILogin* p = new EmailLogin(); p->accept(); p->display(); delete p; return 0; }
```

9) File operations: copy, count digits/spaces, count words, count occurrence (concise)

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

int main(){
    // 9.a copy
    ifstream in("First.txt");
    ofstream out("Second.txt");
    out << in.rdbuf();
    in.close(); out.close();

    // 9.b count digits and spaces
    ifstream f("First.txt");
    char ch; int digits=0, spaces=0;
    while(f.get(ch)){ if(isdigit(ch)) digits++; if(isspace(ch)) spaces++; }
    cout<<digits<<" digits, "<<spaces<<" spaces\n";
    f.close();

    // 9.c count words
    ifstream f2("First.txt");
    string w; int words=0;
    while(f2 >> w) words++;
    cout<<words<<" words\n";
    f2.close();

    // 9.d count occurrence of a word
    ifstream f3("First.txt");
    string content((istreambuf_iterator<char>(f3)), istreambuf_iterator<char>());
    string key = "the"; // example
    int occ=0; size_t pos=0;
    while((pos = content.find(key, pos))!=string::npos){ occ++; pos += key.size(); }
    cout<<occ<<" occurrences of '"<<key<<"'\n";
    return 0;
}
```

10) Function template & class template examples

```cpp
#include <iostream>
using namespace std;
template<typename T>
T sumArray(T arr[], int n){
    T s = 0;
    for(int i=0;i<n;i++) s += arr[i];
    return s;
}
template<typename T>
T square(T x){ return x*x; }
template<>
string square<string>(string s){ return s + s; }

int main(){
    int a[3] = {1,2,3};
    cout<<sumArray(a,3)<<"\n";
    cout<<square(5)<<"\n";
    cout<<square(string("hi"))<<"\n";
    return 0;
}
```

11) generic vectors: modify element, multiply by scalar, display

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

int main(){
    vector<int> v = {10,20,30};
    // modify second element
    v[1] = 25;
    // multiply by scalar 2
    for(auto &x: v) x *= 2;
    // display
    cout<<"(";
    for(size_t i=0;i<v.size();i++){ cout<<v[i]; if(i+1<v.size()) cout<<","; }
    cout<<")\n";
    return 0;
}
```

12) Write C++ program using STL: stack and queue

```cpp
#include <iostream>
#include <stack>
#include <queue>
using namespace std;
int main(){
    stack<int> s; s.push(1); s.push(2); cout<<s.top()<<endl; s.pop();
    queue<int> q; q.push(10); q.push(20); cout<<q.front()<<endl; q.pop();
    return 0;
}
```