```cpp
#include <iostream>
#include <fstream>
#include <unordered_map>
#include <vector>
#include <list>

using namespace std;

class academiaChino {

private:

    vector<string> estudiantesGrad;
    unordered_map<string, pair<int, list<string>::iterator>> personas;
    unordered_map<int, list<string>> clases;

    void OrdenarBurbuja() {

        bool Ordenado;
        string temp;
        int t = estudiantesGrad.size();
        Ordenado = false;
         int i = 0;
        while (!Ordenado)
        {
            Ordenado = true;
            {
                for (int j = t - 1; j > i; j--)
                {
                    if (estudiantesGrad[j] < estudiantesGrad[j - 1])
                    {
                        temp = estudiantesGrad[j];
                        estudiantesGrad[j] = estudiantesGrad[j - 1];
                        estudiantesGrad[j - 1] = temp;
                        Ordenado = false;
                    }
                }
            }
            i++;
        }
    }

public:

    academiaChino() {}

    void nuevo_estudiante(string d, int n) {

        if (personas.count(d) == 1) {
            throw domain_error("Estudiante existente");
        }

        if (n < 1 || n > 6) {
            throw domain_error("Grupo incorrecto");
```

```
        }

        auto it = clases[n].insert(clases[n].end(), d);

        personas[d] = {n, it};
    }

    int grupo_estudiante(string dni) {

        if (personas.count(dni) == 0) {
            throw domain_error("Estudiante no existente");
        }

        int grupo = personas[dni].first;

        if (grupo == 7) {
            throw domain_error("Estudiante ya graduado");
        }

        return grupo;
    }

    void promocionar(string dni) {

        if (personas.count(dni) == 0) {
            throw domain_error("Estudiante no existente");
        }

        int grupo = personas[dni].first;

        if (grupo == 7) {
            throw domain_error("Estudiante ya graduado");
        }

        clases[grupo].erase(personas[dni].second);

        if (grupo == 6) {

            estudiantesGrad.push_back(dni);
            personas[dni] = { 7, clases[grupo].end() };
        }
        else {

            grupo++;

            auto it = clases[grupo].insert(clases[grupo].end(), dni);

            personas[dni] = {grupo, it};
        }
    }

    vector<string> graduados() {

        OrdenarBurbuja();
```

```
        return estudiantesGrad;
    }

    string novato(int grupo) {

        if (grupo < 1 || grupo > 6) {
            throw domain_error("Grupo incorrecto");
        }

        if (clases[grupo].empty()) {
            throw domain_error("Grupo vacio");
        }

        return clases[grupo].back();
    }
};


bool resuelveCaso() {

    string comando, dni;

    int grupo;

    cin >> comando;

    if (!cin) {
        return false;
    }
    else {

        academiaChino academia;

        while (comando != "FIN") {

            try {
                if (comando == "nuevo_estudiante") {

                    cin >> dni >> grupo;
                    academia.nuevo_estudiante(dni, grupo);
                }
                else if (comando == "promocionar") {

                    cin >> dni;
                    academia.promocionar(dni);
                }
                else if (comando == "grupo_estudiante") {

                    cin >> dni;

                    grupo = academia.grupo_estudiante(dni);

                    cout << dni << " esta en el grupo " << grupo << endl;
```

```
                }
                else if (comando == "graduados") {

                    vector<string> sol;

                    sol = academia.graduados();

                    cout << "Lista de graduados:";

                    for (long unsigned int i = 0; i < sol.size(); i++) {
                        cout << " " << sol[i];
                    }

                    cout << endl;
                }
                else if (comando == "novato") {

                    cin >> grupo;

                    string dni = academia.novato(grupo);

                    cout << "Novato de " << grupo << ": " << dni << endl;
                }
            }
            catch (domain_error& e) {
                cout << "ERROR: " << e.what() << endl;
            }

            cin >> comando;
        }

        cout << "---" << endl;

        return true;
    }
}

int main() {
    // ajustes para que cin extraiga directamente de un fichero
#ifndef DOMJUDGE
    std::ifstream in("datos.txt");
    auto cinbuf = std::cin.rdbuf(in.rdbuf());
#endif

    while (resuelveCaso());

    // para dejar todo como estaba al principio
#ifndef DOMJUDGE
    std::cin.rdbuf(cinbuf);
    system("PAUSE");
#endif
    return 0;
}
```