```cpp
#include <iostream>
#include <string>
#include <stdlib.h>
#include <cstring>

const int TAM=100;

using namespace std;
class Empleado {
private:
    int claveEmpleado;
    char nombre[100];
    char domicilio[100];
    float sueldo;
    char reportaA[100];

public:
    // Constructor
     Empleado(){};
    Empleado(int clave, const char* nombre, const char* domicilio, float sueldo,
const char* reportaA) {
        claveEmpleado = clave;
        strcpy(this->nombre, nombre);
        strcpy(this->domicilio, domicilio);
        this->sueldo = sueldo;
        strcpy(this->reportaA, reportaA);
    }

    // Método para imprimir los datos del empleado
    void imprime() {
        cout << "Clave de Empleado: " << claveEmpleado << endl;
        cout << "Nombre: " << nombre << endl;
        cout << "Domicilio: " << domicilio << endl;
        cout << "Sueldo: " << sueldo << endl;
        cout << "Reporta a: " << reportaA << endl;
    }

    // Método para cambiar el domicilio
    void cambiaDomic(const char* nuevoDomicilio) {
        strcpy(domicilio, nuevoDomicilio);
    }

    // Método para cambiar el supervisor al que reporta
    void cambiaReportaA(const char* nuevoSupervisor) {
        strcpy(reportaA, nuevoSupervisor);
    }

    // Método para actualizar el sueldo
    void actualSueldo(float nuevoSueldo) {
        sueldo = nuevoSueldo;
    }

    bool operator == (const Empleado& e){
        return nombre == e.nombre;
    }
    bool operator < (const Empleado& e){
        return nombre < e.nombre;
    }
    bool operator <= (const Empleado& e){
        return nombre <= e.nombre;
    }
    bool operator > (const Empleado& e){
        return nombre > e.nombre;
    }
    bool operator >= (const Empleado& e){
        return nombre >= e.nombre;
```

```
66      }
67      friend std::istream& operator >> (std::istream& g, Empleado& o){
68          cout << endl;
69          cout <<" Nombre: ";
70          g>>o.nombre;
71          cout << " Domicilio: ";
72          g>>o.domicilio;
73          cout << " Sueldo: ";
74          g>>o.sueldo;
75          cout << endl << endl;
76          return g;
77
78      }
79      friend std::ostream& operator << (std::ostream& g, const Empleado& o){
80          g << " Nombre: " << o.nombre << endl;
81          g << " Domicilio: " << o.domicilio << endl;
82          g << " Sueldo: " << o.sueldo << endl;
83          return g;
84
85      }
86
87
88  };
89  class Pila{
90  private:
91      Empleado datos[TAM];
92      int ult;
93        bool elimina(int pos){
94          if(vacia() || pos<0 || pos>ult){
95              std::cout<<"\n error de eliminacion";
96              return true;
97          }
98          int i=pos;
99          while(i<ult){
100             datos[i]=datos[i+1];
101             i++;
102         }
103         ult--;
104         return false;
105     }
106      int inserta(Empleado& elem, int pos){
107         if(llena()|| pos<0 || pos>ult+1){
108             std::cout<<"\n Error de insercion";
109             return 0;
110         }
111         int i=ult+1;
112         while(i>pos){
113             datos[i]=datos[i-1];
114             i--;
115         }
116         datos[pos]=elem;
117         ult++;
118         return 1;
119     }
120 public:
121     Pila():ult(-1){}
122     bool vacia()const{
123         if(ult==-1)
124             return true;
125         return false;
126     }
127     bool llena()const{
128         if(ult==TAM-1)
129             return true;
130         return false;
131     }
```

```
132
133     int ultimo()const{
134         return ult;
135     }
136
137     friend std::ostream& operator<<(std::ostream & o, Pila& L){
138         int i=0;
139         std::cout<<"\n";
140         while(i<=L.ultimo()){;
141             o<<L.datos[i];
142             i++;
143         }
144         return o;
145     }
146
147     void apilar(Empleado& elem){
148             inserta(elem,ultimo()+1);
149     }
150     Empleado& desapilar(){
151         ult--;
152         return datos[ult+1];
153     }
154
155 };
156
157 int main()
158 {
159     Empleado miEmpleado;
160     Pila miPila;
161     int opc;
162
163     do{
164
165     cout << "\n |***PILAS ESTATICAS***|" << endl << endl;
166     cout << " 1. Push " << endl;
167     cout << " 2. Pop " << endl;
168     cout << " 3. Top " << endl;
169     cout << " 4. Salir " << endl;
170     cout << " Elige una opcion: ";
171     cin >> opc;
172     cin.ignore();
173
174     switch(opc){
175     case 1: cout << " ***PUSH*** " << endl;
176             cin >> miEmpleado;
177             miPila.apilar(miEmpleado);
178         break;
179     case 2: cout << " ***POP*** " << endl;
180             miPila.desapilar();
181         break;
182     case 3: cout << " ***TOP*** " << endl;
183             cout << miPila.ultimo();
184         break;
185     case 4: cout << " Saliendo... " << endl;
186         break;
187     default: cout << " Opcion invalida..." << endl;
188         break;
189
190     }
191
192     }while(opc!=4);
193
194     return 0;
195 }
```