```cpp
/*
* Juego de la Serpiente v4
* Pablo_Villa 08/11/2023
*/
#include <iostream>
#include "terminal.h"
#include <cstdlib>
#include <ctime>

using namespace std;

const char TECLA_SIGUIENTE = ' ';
const char TECLA_FIN = 'F';
const char SERPIENTE = '@';
const char CUERPO_SERPIENTE = 'o';
const char ARRIBA = 'W';
const char ABAJO = 'S';
const char IZQUIERDA = 'A';
const char DERECHA = 'D';
const char MANZANA = 'M';
const char SIMBOLO_VERTICAL = '|';
const char SIMBOLO_INTERMEDIO = ' ';
const char SIMBOLO_HORIZONTAl = '-';
const char SIMBOLO_EXTERIOR = '+';
const int PREMIO = 100;
const int LONG_SERPIENTE = 15;
const int BASE = 80;
const int ALTURA = 22;
const int RETARDO = 50;
const int SERPIENTE_X_INICIAL = 10;
const int LIMITE_SUPERIOR = 1;
const int LIMITE_INFERIOR = 20 ;
const int LIMITE_IZQUIERDA = 2;
const int LIMITE_DERECHA = 78;
const int SERPIENTE_Y_INICIAL = 15;
const int MOVIMIENTO_X_DERECHA = 1;
const int MOVIMIENTO_Y_DESCENDENTE = 1;
const int MOVIMIENTO_X_IZQUIERDA = -1;
const int MOVIMIENTO_Y_ASCENDENTE = -1;
const int MAX_MANZANAS = 10;
const string TITULO = "Juego de la serpiente ";
const string VERSION = "4.0";
const string TECLA_CONTINUAR = "ESPACIO";

struct posicion {
    int x = 0;
    int y = 0;
};

struct inc_unitario_posicion {
    int x = 0;
    int y = 0;
};

void iniciar_pantalla_inicial();
void inicializar_juego(char tecla,posicion serpiente[],inc_unitario_posicion& inc_unitario_posicion);
void pantalla_incial();
void dibujar_linea(const char c_exterior, const char c_interior, const int largo);
void dibujar_rectangulo(const int base, const int altura);
void inicializar_serpiente(posicion serpiente[], inc_unitario_posicion& inc_unitario_posicion);
bool juego_terminado(char tecla,posicion serpiente[]);
void obtener_direccion(const char tecla,inc_unitario_posicion& inc_unitario_posicion);
void pintar_serpiente(const posicion serpiente[]);
void borrar_serpiente( const posicion serpiente[]);
bool serpiente_tocada(const posicion serpiente[]);
void mover_serpiente(posicion serpiente[], inc_unitario_posicion inc_unitario_posicion);
```

```
67
68  int main() {
69      char tecla = '\0';
70      posicion serpiente[LONG_SERPIENTE];
71      inc_unitario_posicion inc_unitario_posicion = {0, 0};
72
73      srand(time(0));
74      setlocale(LC_ALL, "");
75
76      inicializar_juego(tecla,serpiente, inc_unitario_posicion);
77      while ( ! juego_terminado(tecla, serpiente)) {
78          pintar_serpiente(serpiente);
79
80          retardar(RETARDO);
81
82          borrar_serpiente(serpiente);
83
84          obtener_direccion(tecla, inc_unitario_posicion);
85          mover_serpiente(serpiente, inc_unitario_posicion);
86
87          tecla = leer_tecla();
88      }
89      deshabilitar_modo_crudo_terminal();
90      borrar_terminal();
91  }
92
93  void iniciar_pantalla_inicial(){
94      retardar(RETARDO);
95      hacer_cursor_visible(false);
96      pantalla_incial();
97  }
98
99  void inicializar_juego(char tecla ,posicion serpiente[], inc_unitario_posicion& inc_unitario_posicion){
100     iniciar_pantalla_inicial();
101     while(leer_tecla()!= TECLA_SIGUIENTE){
102             retardar(RETARDO);
103     }
104     deshabilitar_modo_crudo_terminal();
105     borrar_terminal();
106
107     inicializar_serpiente(serpiente, inc_unitario_posicion);
108
109     dibujar_rectangulo(BASE, ALTURA);
110     retardar(RETARDO);
111
112     habilitar_modo_crudo_terminal();
113     hacer_cursor_visible(false);
114     tecla = leer_tecla();
115 }
116
117 void pantalla_incial(){
118
119     poner_cursor(1,1);
120     cout << " **************************** " << endl;
121     poner_cursor(1,2);
122     cout << " * "<< TITULO << VERSION << " * " << endl;
123     poner_cursor(1,3);
124     cout << " **************************** " << endl;
125     poner_cursor(1,6);
126     cout << "   ______" << endl;
127     poner_cursor(1,7);
128     cout << " _/      \\ " << endl;
129     poner_cursor(1,8);
130     cout << "  \\\___    \\ " << endl;
131     poner_cursor(1,9);
132     cout << "      \\\    \\\_____" << endl;
```

```
133     poner_cursor(1,10);
134     cout << "       \\         \\ " << endl;
135     poner_cursor(1,11);
136     cout << "        \\____     \\_____      _|_ " << endl;
137     poner_cursor(1,12);
138     cout << "             \\         0 \\_/  /   \\ " << endl;
139     poner_cursor(1,13);
140     cout << "              \\__________/ \\  \\___/" << endl;
141     poner_cursor(1,17);
142     cout << "Pulsa la tecla de " <<  TECLA_CONTINUAR << " para continuar" << endl;
143 }
144
145 void inicializar_serpiente(posicion serpiente[], inc_unitario_posicion& inc_unitario_posicion) {
146     serpiente[0].x = SERPIENTE_X_INICIAL;
147     serpiente[0].y = SERPIENTE_Y_INICIAL;
148
149     inc_unitario_posicion.x = MOVIMIENTO_X_DERECHA;
150     inc_unitario_posicion.y = MOVIMIENTO_Y_ASCENDENTE;
151
152     for (int i = 1; i < LONG_SERPIENTE; i++) {
153         serpiente[i].x = serpiente[i - 1].x + 1;
154         serpiente[i].y = serpiente[i - 1].y;
155     }
156 }
157
158 void dibujar_linea(const char c_exterior, const char c_interior, const int largo){
159     cout << c_exterior;
160     for(int i = 0; i < largo - 2; i++){
161         cout << c_interior;
162     }
163     cout << c_exterior << endl;
164 }
165
166 void dibujar_rectangulo(const int base, const int altura){
167     poner_cursor(2,1);
168     cout << "+------------------------- " << TITULO
169         << VERSION << " --------------------------+ " << endl;
170     for (int i = 2; i < altura -2; i++){
171         poner_cursor(2,i);
172         dibujar_linea(SIMBOLO_VERTICAL,SIMBOLO_INTERMEDIO,base);
173     }
174     poner_cursor(2,altura - 2);
175     dibujar_linea(SIMBOLO_EXTERIOR,SIMBOLO_HORIZONTAl,base);
176     poner_cursor(2, altura);
177     cout << ARRIBA << "-> Subir " << ABAJO << "-> Bajar " << IZQUIERDA
178          << "-> Izda " << DERECHA << "-> Dcha " << TECLA_FIN << "-> Fin" << endl;
179 }
180
181 bool juego_terminado(char tecla, posicion serpiente[]){
182     return(toupper(tecla) == TECLA_FIN ||
183            serpiente_tocada(serpiente) ||
184            serpiente[0].x == LIMITE_IZQUIERDA ||
185            serpiente[0].x == LIMITE_DERECHA ||
186            serpiente[0].y == LIMITE_SUPERIOR ||
187            serpiente[0].y == LIMITE_INFERIOR);
188 }
189
190 void obtener_direccion(const char tecla, inc_unitario_posicion& inc_unitario_posicion) {
191     switch (toupper(tecla)) {
192         case ARRIBA:
193             inc_unitario_posicion.x = 0;
194             inc_unitario_posicion.y = MOVIMIENTO_Y_ASCENDENTE;
195             break;
196
197         case ABAJO:
198             inc_unitario_posicion.x = 0;
```

```
            inc_unitario_posicion.y = MOVIMIENTO_Y_DESCENDENTE;
            break;

        case IZQUIERDA:
            inc_unitario_posicion.x = MOVIMIENTO_X_IZQUIERDA;
            inc_unitario_posicion.y = 0;
            break;

        case DERECHA:
            inc_unitario_posicion.x = MOVIMIENTO_X_DERECHA;
            inc_unitario_posicion.y = 0;
            break;
  }
}

void mover_serpiente(posicion serpiente[], inc_unitario_posicion inc_unitario_posicion) {
    posicion cabeza_anterior;
    cabeza_anterior = serpiente[0];

    serpiente[0].x = serpiente[0].x + inc_unitario_posicion.x;
    serpiente[0].y = serpiente[0].y + inc_unitario_posicion.y;

    for (int i = LONG_SERPIENTE - 1; i > 0; --i) {
        serpiente[i] = serpiente[i - 1];
    }
    serpiente[1] = cabeza_anterior;
}


void pintar_serpiente(const posicion serpiente[]) {
    poner_cursor(serpiente[0].x, serpiente[0].y);
    cout << SERPIENTE;

    for (int i = 1; i < LONG_SERPIENTE - 1; i++) {
        poner_cursor(serpiente[i].x, serpiente[i].y);
        cout << CUERPO_SERPIENTE;
    }
}

void borrar_serpiente( const posicion serpiente[]) {
    for (int i = 0; i < LONG_SERPIENTE - 1; i++) {
        poner_cursor(serpiente[i].x, serpiente[i].y);
        cout << " ";
    }
}

bool serpiente_tocada(const posicion serpiente[]) {
    for (int i = 1; i < LONG_SERPIENTE - 1; ++i) {
        if (serpiente[0].x == serpiente[i].x &&
            serpiente[0].y == serpiente[i].y) {
            return true;
        }
    }
    return false;
}
```

```
/*
* Juego de la Serpiente v1
*
* Pablo_Villa 08/11/2023
*
* version en la que por cada manzana
* comida se incrementa en 1 el tamaño
*/
#include <iostream>
#include "terminal.h"
#include <cstdlib>
#include <ctime>

using namespace std;

const char TECLA_SIGUIENTE = ' ';
const char TECLA_FIN = 'F';
const char SERPIENTE = '@';
const char CUERPO_SERPIENTE = 'o';
const char ARRIBA = 'W';
const char ABAJO = 'S';
const char IZQUIERDA = 'A';
const char DERECHA = 'D';
const char MANZANA = 'M';
const char SIMBOLO_VERTICAL = '|';
const char SIMBOLO_INTERMEDIO = ' ';
const char SIMBOLO_HORIZONTAl = '-';
const char SIMBOLO_EXTERIOR = '+';
const int PREMIO = 100;
const int LONG_SERPIENTE = 100;
const int BASE = 80;
const int ALTURA = 22;
const int RETARDO = 50;
const int SERPIENTE_X_INICIAL = 10;
const int LIMITE_SUPERIOR = 1;
const int LIMITE_INFERIOR = 20 ;
const int LIMITE_IZQUIERDA = 2;
const int LIMITE_DERECHA = 78;
const int SERPIENTE_Y_INICIAL = 15;
const int MOVIMIENTO_X_DERECHA = 1;
const int MOVIMIENTO_Y_DESCENDENTE = 1;
const int MOVIMIENTO_X_IZQUIERDA = -1;
const int MOVIMIENTO_Y_ASCENDENTE = -1;
const int MARGEN_INI_MANZANA = 5;
const int MARGEN_MARCADOR = 5;
const int MAX_MANZANAS = 10;
const string TITULO = "Juego de la serpiente ";
const string VERSION = "5.0";
const string TECLA_CONTINUAR = "ESPACIO";

struct posicion {
    int x = 0;
    int y = 0;
};

struct inc_unitario_posicion {
    int x = 0;
    int y = 0;
};

void iniciar_pantalla_inicial();
void inicializar_juego(char tecla,posicion serpiente[],posicion& manzana,
                       inc_unitario_posicion& inc_unitario_posicion, int longitud_serpiente);
void pantalla_incial();
void dibujar_linea(const char c_exterior, const char c_interior, const int largo);
void dibujar_rectangulo(const int base, const int altura);
```

```
 67  void inicializar_serpiente(posicion serpiente[], inc_unitario_posicion& inc_unitario_posicion,int
longitud_serpiente);
 68  bool juego_terminado(char tecla,posicion serpiente[],int longitud_serpiente);
 69  void obtener_direccion_serpiente(const char tecla,inc_unitario_posicion& inc_unitario_posicion);
 70  void pintar_serpiente(const posicion serpiente[], int longitud_serpiente);
 71  void borrar_serpiente( const posicion serpiente[], int longitud_serpiente);
 72  bool serpiente_tocada(const posicion serpiente[],int longitud_serpiente);
 73  void inicializar_manzana(posicion& manzana);
 74  void pintar_manzana(const posicion& manzana);
 75  void borrar_manzana(const posicion& manzana);
 76  bool manzana_tocada(const posicion& manzana, const posicion serpiente[]);
 77  void mover_serpiente(posicion serpiente[], inc_unitario_posicion inc_unitario_posicion);
 78  void actualizar_marcador(int& puntos);
 79  void actualizar_longitud(int& longitud_serpiente);
 80
 81  int main() {
 82      int puntos = 0;
 83      bool hay_manzana = false;
 84      char tecla = '\0';
 85      int longitud_serpiente = 5;
 86      posicion serpiente[LONG_SERPIENTE];
 87      posicion manzana = {0, 0};
 88      inc_unitario_posicion inc_unitario_posicion = {0, 0};
 89
 90      inicializar_juego(tecla, serpiente, manzana, inc_unitario_posicion,longitud_serpiente);
 91      while ( ! juego_terminado(tecla, serpiente,longitud_serpiente)) {
 92
 93          if( ! hay_manzana){
 94              inicializar_manzana(manzana);
 95              pintar_manzana(manzana);
 96              hay_manzana = true;
 97          }
 98          if( manzana_tocada(manzana, serpiente)) {
 99              hay_manzana = false;
100              actualizar_longitud(longitud_serpiente);
101              poner_cursor(1,28);
102              cout << longitud_serpiente;
103              actualizar_marcador(puntos);
104          }
105
106          pintar_serpiente(serpiente,longitud_serpiente);
107
108          retardar(RETARDO);
109
110          borrar_serpiente(serpiente,longitud_serpiente);
111
112          obtener_direccion_serpiente(tecla, inc_unitario_posicion);
113          mover_serpiente(serpiente, inc_unitario_posicion);
114
115          tecla = leer_tecla();
116      }
117      deshabilitar_modo_crudo_terminal();
118      borrar_terminal();
119  }
120
121  void iniciar_pantalla_inicial(){
122      retardar(RETARDO);
123      hacer_cursor_visible(false);
124      pantalla_incial();
125  }
126
127  void inicializar_juego(char tecla, posicion serpiente[], posicion& manzana,
128                         inc_unitario_posicion& inc_unitario_posicion,int longitud_serpiente){
129      srand(time(0));
130      setlocale(LC_ALL, "");
131      iniciar_pantalla_inicial();
```

```
132
133     while(leer_tecla()!= TECLA_SIGUIENTE){
134             retardar(RETARDO);
135     }
136     deshabilitar_modo_crudo_terminal();
137     borrar_terminal();
138
139     inicializar_serpiente(serpiente, inc_unitario_posicion,longitud_serpiente);
140
141     dibujar_rectangulo(BASE, ALTURA);
142
143     habilitar_modo_crudo_terminal();
144     hacer_cursor_visible(false);
145     tecla = leer_tecla();
146 }
147
148 void pantalla_incial(){
149
150     poner_cursor(1,1);
151     cout << " **************************** " << endl;
152     poner_cursor(1,2);
153     cout << " * "<< TITULO << VERSION << " * " << endl;
154     poner_cursor(1,3);
155     cout << " **************************** " << endl;
156     poner_cursor(1,6);
157     cout << "   ______" << endl;
158     poner_cursor(1,7);
159     cout << " _/      \\ " << endl;
160     poner_cursor(1,8);
161     cout << "  \\___    \\ " << endl;
162     poner_cursor(1,9);
163     cout << "      \\    \\_____" << endl;
164     poner_cursor(1,10);
165     cout << "       \\         \\ " << endl;
166     poner_cursor(1,11);
167     cout << "        \\____     \\______      _|_ " << endl;
168     poner_cursor(1,12);
169     cout << "             \\         0 \\_/  /   \\ " << endl;
170     poner_cursor(1,13);
171     cout << "              \\__________/ \\  \\___/" << endl;
172     poner_cursor(1,17);
173     cout << "Pulsa la tecla de " <<  TECLA_CONTINUAR << " para continuar" << endl;
174 }
175
176 void inicializar_serpiente(posicion serpiente[], inc_unitario_posicion& inc_unitario_posicion,int
longitud_serpiente) {
177     serpiente[0].x = SERPIENTE_X_INICIAL;
178     serpiente[0].y = SERPIENTE_Y_INICIAL;
179
180     inc_unitario_posicion.x = MOVIMIENTO_X_DERECHA;
181     inc_unitario_posicion.y = MOVIMIENTO_Y_ASCENDENTE;
182
183     for (int i = 1; i < longitud_serpiente; i++) {
184         serpiente[i].x = serpiente[i - 1].x + 1;
185         serpiente[i].y = serpiente[i - 1].y;
186     }
187 }
188
189 void dibujar_linea(const char c_exterior, const char c_interior, const int largo){
190     cout << c_exterior;
191     for(int i = 0; i < largo - 2; i++){
192         cout << c_interior;
193     }
194     cout << c_exterior << endl;
195 }
196
```

```
197 void dibujar_rectangulo(const int base, const int altura){
198     poner_cursor(2,1);
199     cout << "+------------------------- " << TITULO
200         << VERSION << " -------------------------+ " << endl;
201     for (int i = 2; i < altura -2; i++){
202         poner_cursor(2,i);
203         dibujar_linea(SIMBOLO_VERTICAL,SIMBOLO_INTERMEDIO,base);
204     }
205     poner_cursor(2,altura - 2);
206     dibujar_linea(SIMBOLO_EXTERIOR,SIMBOLO_HORIZONTAl,base);
207     poner_cursor(2, altura);
208     cout << ARRIBA << "-> Subir " << ABAJO << "-> Bajar " << IZQUIERDA
209          << "-> Izda " << DERECHA << "-> Dcha " << TECLA_FIN << "-> Fin" << endl;
210 }
211
212 bool juego_terminado(char tecla, posicion serpiente[], int longitud_serpiente){
213     return(toupper(tecla) == TECLA_FIN ||
214            serpiente_tocada(serpiente,longitud_serpiente) ||
215            serpiente[0].x == LIMITE_IZQUIERDA ||
216            serpiente[0].x == LIMITE_DERECHA ||
217            serpiente[0].y == LIMITE_SUPERIOR ||
218            serpiente[0].y == LIMITE_INFERIOR);
219 }
220
221 void obtener_direccion_serpiente(const char tecla, inc_unitario_posicion& inc_unitario_posicion) {
222     switch (toupper(tecla)) {
223         case ARRIBA:
224             inc_unitario_posicion.x = 0;
225             inc_unitario_posicion.y = MOVIMIENTO_Y_ASCENDENTE;
226             break;
227
228         case ABAJO:
229             inc_unitario_posicion.x = 0;
230             inc_unitario_posicion.y = MOVIMIENTO_Y_DESCENDENTE;
231             break;
232
233         case IZQUIERDA:
234             inc_unitario_posicion.x = MOVIMIENTO_X_IZQUIERDA;
235             inc_unitario_posicion.y = 0;
236             break;
237
238         case DERECHA:
239             inc_unitario_posicion.x = MOVIMIENTO_X_DERECHA;
240             inc_unitario_posicion.y = 0;
241             break;
242   }
243 }
244
245 void mover_serpiente(posicion serpiente[], inc_unitario_posicion inc_unitario_posicion) {
246     posicion cabeza_anterior;
247     cabeza_anterior = serpiente[0];
248
249     serpiente[0].x = serpiente[0].x + inc_unitario_posicion.x;
250     serpiente[0].y = serpiente[0].y + inc_unitario_posicion.y;
251
252     for (int i = LONG_SERPIENTE - 1; i > 0; --i) {
253         serpiente[i] = serpiente[i - 1];
254     }
255     serpiente[1] = cabeza_anterior;
256 }
257
258
259 void pintar_serpiente(const posicion serpiente[],int longitud_serpiente) {
260     poner_cursor(serpiente[0].x, serpiente[0].y);
261     cout << SERPIENTE;
262
```

```
263     for (int i = 1; i < longitud_serpiente - 1; i++) {
264         poner_cursor(serpiente[i].x, serpiente[i].y);
265         cout << CUERPO_SERPIENTE;
266     }
267 }
268 
269 void borrar_serpiente( const posicion serpiente[], int longitud_serpiente) {
270     for (int i = 0; i < longitud_serpiente - 1; i++) {
271         poner_cursor(serpiente[i].x, serpiente[i].y);
272         cout << " ";
273     }
274 }
275 
276 bool serpiente_tocada(const posicion serpiente[],int longitud_serpiente) {
277     for (int i = 1; i < longitud_serpiente - 1; ++i) {
278         if (serpiente[0].x == serpiente[i].x &&
279             serpiente[0].y == serpiente[i].y) {
280             return true;
281         }
282     }
283     return false;
284 }
285 
286 void inicializar_manzana(posicion& manzana) {
287     manzana.x = LIMITE_IZQUIERDA + MARGEN_INI_MANZANA +
288                 rand() % (LIMITE_DERECHA - LIMITE_IZQUIERDA - MARGEN_INI_MANZANA );
289 
290     manzana.y = LIMITE_SUPERIOR + MARGEN_INI_MANZANA +
291                 rand() % (LIMITE_INFERIOR - LIMITE_SUPERIOR - MARGEN_INI_MANZANA);
292 }
293 
294 void pintar_manzana(const posicion& manzana){
295     poner_cursor(manzana.x, manzana.y);
296     cout << MANZANA;
297 }
298 
299 void borrar_manzana(const posicion& manzana){
300     poner_cursor(manzana.x, manzana.y);
301     cout << " ";
302 }
303 
304 bool manzana_tocada(const posicion& manzana, const posicion serpiente[]){
305     return(manzana.x == serpiente[0].x &&
306            manzana.y == serpiente[0].y);
307 }
308 
309 void actualizar_marcador(int& puntos){
310     puntos = puntos + PREMIO;
311     poner_cursor(LIMITE_IZQUIERDA,LIMITE_INFERIOR + MARGEN_MARCADOR);
312     cout << "PUNTOS: " << puntos;
313 }
314 void actualizar_longitud(int& longitud_serpiente){
315 
316  longitud_serpiente = longitud_serpiente + 1;
317 
318 }
```

```cpp
/*
 * Pablo_Villa 874773
 * 01/12/23
 */

#include <iostream>
using namespace std;

const int MAX_PRODUCTOS = 100;
const int PRODUCTO_NO_ENCONTRADO = -1;
const int ALTA = 1;
const int BAJA = 2;
const int LISTADO = 3;
const int ENTRADA = 4;
const int SALIDA = 5;
const int BUSCAR = 6;
const int FIN = 7;

struct producto_almacen {
    int codigo = 0;
    string descripcion = "";
    int existencias = 0;
};

void anadir_producto(producto_almacen productos[], int &numero_productos);
void leer_producto(producto_almacen &producto);
void listar_productos(producto_almacen productos[], int numero_productos);
void anadir(producto_almacen productos[], int &numero_productos);
void eliminar(producto_almacen productos[], int &numero_productos);
void borrar_producto(producto_almacen productos[], int &numero_productos);
void buscar(producto_almacen productos[], int &numero_productos);
int busqueda(const producto_almacen productos[], int codigo, int numero_productos);

int main(){
    producto_almacen productos[MAX_PRODUCTOS];
    int opcion = 0;
    int numero_productos = 0;

    while(opcion != FIN){
        cout << ALTA << "->Alta " << BAJA << "->Baja " << LISTADO << "->Listado " << ENTRADA
             << "->Entrada " << SALIDA << "->Salida " << BUSCAR << "-> Buscar " << FIN << "->Acabar: ";

        cin >> opcion;

        switch(opcion){
            case ALTA:
                anadir_producto(productos, numero_productos);
                break;

            case BAJA:
                borrar_producto(productos, numero_productos);
                break;

            case LISTADO:
                listar_productos(productos, numero_productos);
                break;

            case ENTRADA:
                anadir(productos, numero_productos);
                break;

            case SALIDA:
                eliminar(productos, numero_productos);
                break;

            case BUSCAR:
```

```
                buscar(productos, numero_productos);
                break;

            case FIN:
                cout << "Fin" << endl;
                break;
        }
    }
}

void leer_producto(producto_almacen &producto){
    cout << "Código : ";
    cin >> producto.codigo;
    cout << "Descripción : ";
    cin.ignore();
    getline(cin, producto.descripcion);
    cout << "Existencias : ";
    cin >> producto.existencias;
}

void anadir_producto(producto_almacen productos[], int &numero_productos){
    if (numero_productos > MAX_PRODUCTOS){
        cout << "Número máximo de productos" << endl;
    } else {
        leer_producto(productos[numero_productos]);
        numero_productos++;
    }
}

void borrar_producto(producto_almacen productos[], int &numero_productos){
    int codigo = 0;
    int posicion = 0;
    cout << "Introduce un código de un producto que desees eliminar ";
    cin >> codigo;


    posicion = busqueda(productos, codigo, numero_productos);
    if (posicion != PRODUCTO_NO_ENCONTRADO) {
        productos[posicion] = productos[numero_productos - 1];
        numero_productos--;
        cout << "Producto eliminado" << endl;
    }
    else {
        cout << "Producto no encontrado" << endl;
    }
}

void mostrar_producto(const producto_almacen &producto){
    cout << "Codigo: ";
    cout << producto.codigo << endl;
    cout << "Descripcion: ";
    cout << producto.descripcion << endl;
    cout << "Existencias: ";
    cout << producto.existencias << endl;
}

void listar_productos(producto_almacen productos[], int numero_productos){
    for (int i = 0; i < numero_productos; i++){
        mostrar_producto(productos[i]);
    }
}

void anadir(producto_almacen productos[], int &numero_productos){
    int codigo = 0;
    int existencias = 0;
    bool encontrado = false;
```

```
133     int posicion = 0;
134 
135     cout << "codigo: ";
136     cin >> codigo;
137     cout << "Añadir existencias: ";
138     cin >> existencias;
139 
140     posicion = busqueda(productos, codigo, numero_productos);
141     if (posicion == PRODUCTO_NO_ENCONTRADO) {
142         cout << "El código introducido no corresponde con ningún producto." << endl;
143     } else {
144         productos[posicion].existencias =  productos[posicion].existencias + existencias;
145         cout << "Se han almacenado (" << existencias << ") de "
146              << productos[posicion].descripcion << " con código " << productos[posicion].codigo << endl;
147     }
148 }
149 
150 void eliminar(producto_almacen productos[], int &numero_productos){
151     int codigo = 0;
152     int existencias = 0;
153     bool encontrado = false;
154     int posicion = 0;
155 
156     cout << "Codigo: ";
157     cin >> codigo;
158     cout << "Eliminar existencias: ";
159     cin >> existencias;
160 
161     posicion = busqueda(productos, codigo, numero_productos);
162     if (posicion == PRODUCTO_NO_ENCONTRADO) {
163         cout << "El código introducido no corresponde con ningún producto." << endl;
164     } else {
165         productos[posicion].existencias -= existencias;
166         cout << "Se han retirado (" << existencias << ") de "
167              << productos[posicion].descripcion << " con código " << productos[posicion].codigo << endl;
168     }
169 }
170 
171 void buscar(producto_almacen productos[], int &numero_productos){
172     int codigo = 0;
173     int posicion = 0;
174     cout << "Introduce un código que desees buscar ";
175     cin >> codigo;
176 
177     posicion = busqueda(productos, codigo, numero_productos);
178     if(posicion == PRODUCTO_NO_ENCONTRADO){
179         cout << "El código introducido no corresponde a ningún producto" << endl;
180     } else {
181         mostrar_producto(productos[posicion]);
182     }
183 }
184 
185 int busqueda(const producto_almacen productos[], int codigo, int numero_productos){
186     for (int i = 0; i < numero_productos; i++){
187         if(codigo == productos[i].codigo){
188             return i;
189         }
190     }
191     return PRODUCTO_NO_ENCONTRADO;
192 }
```

Ejercicio 3

Tanto en el primer como en el segundo caso se produce una operación con un resultado infinito. En el primer caso obtenemos inf, por lo que nos muestra inf de infinito positivo. En el segundo caso obtenemos –inf, por lo que nos muestra –inf de infinito negativo.

En el tercer caso obtenemos NaN, que significa “Not an Number”. Obtenemos este resultado debido a un error de representación de reales. Esto es debido a que el segundo factor de la división no puede ser un 0 para poder realizar una operación, porque no existe. En nuestro caso si tenemos ese 0, por lo que obtenemos ese resultado NaN.

En el cuarto y último caso, no obtenemos nada. Al estar dividiendo 0 entre 0, operación que no existe, el resultado tampoco existe, por eso no obtenemos nada al mostrarlo por pantalla. La diferencia con el caso anterior reside en que en el cuarto caso la operación no está definida, por eso no es posible mostrar el resultado por pantalla

```cpp
/*
 * Pablo_Villa 874773
 * 01/12/23
 */

#include <iostream>
using namespace std;

const int MAX_PRODUCTOS = 100;
const int PRODUCTO_NO_ENCONTRADO = -1;
const int ALTA = 1;
const int BAJA = 2;
const int LISTADO = 3;
const int ENTRADA = 4;
const int SALIDA = 5;
const int BUSCAR = 6;
const int FIN = 7;

struct producto_almacen {
    int codigo = 0;
    string descripcion = "";
    int existencias = 0;
};

void anadir_producto(producto_almacen productos[], int &numero_productos);
void leer_producto(producto_almacen &producto);
void listar_productos(producto_almacen productos[], int numero_productos);
void anadir(producto_almacen productos[], int &numero_productos);
void eliminar(producto_almacen productos[], int &numero_productos);
void borrar_producto(producto_almacen productos[], int &numero_productos);
void buscar(producto_almacen productos[], int &numero_productos);
int busqueda(const producto_almacen productos[], int codigo, int numero_productos);
void cargar_datos_desde_archivo(componente componentes[], int &numero_componentes);
void guardar_datos_en_archivo(componente componentes[], int numero_componentes);

const string NOMBRE_ARCHIVO = "datos_componentes.txt";

int main(){
    producto_almacen productos[MAX_PRODUCTOS];
    int opcion = 0;
    int numero_productos = 0;

    cargar_datos_desde_archivo(componentes, numero_componentes);

    while(opcion != FIN){
        cout << ALTA << "->Alta " << BAJA << "->Baja " << LISTADO << "->Listado " << ENTRADA
             << "->Entrada " << SALIDA << "->Salida " << BUSCAR << "-> Buscar " << FIN << "->Acabar: ";

        cin >> opcion;

        switch(opcion){
            case ALTA:
                anadir_producto(productos, numero_productos);
                break;

            case BAJA:
                borrar_producto(productos, numero_productos);
                break;

            case LISTADO:
                listar_productos(productos, numero_productos);
                break;

            case ENTRADA:
                anadir(productos, numero_productos);
                break;
```

```
67
68              case SALIDA:
69                  eliminar(productos, numero_productos);
70                  break;
71
72              case BUSCAR:
73                  buscar(productos, numero_productos);
74                  break;
75
76              case FIN:
77                  guardar_datos_en_archivo(componentes, numero_componentes);
78                  cout << "Fin" << endl;
79                  break;
80          }
81      }
82  }
83
84  void leer_producto(producto_almacen &producto){
85      cout << "Código : ";
86      cin >> producto.codigo;
87      cout << "Descripción : ";
88      cin.ignore();
89      getline(cin, producto.descripcion);
90      cout << "Existencias : ";
91      cin >> producto.existencias;
92  }
93
94  void anadir_producto(producto_almacen productos[], int &numero_productos){
95      if (numero_productos > MAX_PRODUCTOS){
96          cout << "Número máximo de productos" << endl;
97      } else {
98          leer_producto(productos[numero_productos]);
99          numero_productos++;
100     }
101 }
102
103 void borrar_producto(producto_almacen productos[], int &numero_productos){
104     int codigo = 0;
105     int posicion = 0;
106     cout << "Introduce un código de un producto que desees eliminar ";
107     cin >> codigo;
108
109
110     posicion = busqueda(productos, codigo, numero_productos);
111     if (posicion != PRODUCTO_NO_ENCONTRADO) {
112         productos[posicion] = productos[numero_productos - 1];
113         numero_productos--;
114         cout << "Producto eliminado" << endl;
115     }
116     else {
117         cout << "Producto no encontrado" << endl;
118     }
119 }
120
121 void mostrar_producto(const producto_almacen &producto){
122     cout << "Codigo: ";
123     cout << producto.codigo << endl;
124     cout << "Descripcion: ";
125     cout << producto.descripcion << endl;
126     cout << "Existencias: ";
127     cout << producto.existencias << endl;
128 }
129
130 void listar_productos(producto_almacen productos[], int numero_productos){
131     for (int i = 0; i < numero_productos; i++){
132         mostrar_producto(productos[i]);
```

```
    }
}

void anadir(producto_almacen productos[], int &numero_productos){
    int codigo = 0;
    int existencias = 0;
    bool encontrado = false;
    int posicion = 0;

    cout << "codigo: ";
    cin >> codigo;
    cout << "Añadir existencias: ";
    cin >> existencias;

    posicion = busqueda(productos, codigo, numero_productos);
    if (posicion == PRODUCTO_NO_ENCONTRADO) {
        cout << "El código introducido no corresponde con ningún producto." << endl;
    } else {
        productos[posicion].existencias =  productos[posicion].existencias + existencias;
        cout << "Se han almacenado (" << existencias << ") de "
             << productos[posicion].descripcion << " con código " << productos[posicion].codigo << endl;
    }
}

void eliminar(producto_almacen productos[], int &numero_productos){
    int codigo = 0;
    int existencias = 0;
    bool encontrado = false;
    int posicion = 0;

    cout << "Codigo: ";
    cin >> codigo;
    cout << "Eliminar existencias: ";
    cin >> existencias;

    posicion = busqueda(productos, codigo, numero_productos);
    if (posicion == PRODUCTO_NO_ENCONTRADO) {
        cout << "El código introducido no corresponde con ningún producto." << endl;
    } else {
        productos[posicion].existencias -= existencias;
        cout << "Se han retirado (" << existencias << ") de "
             << productos[posicion].descripcion << " con código " << productos[posicion].codigo << endl;
    }
}

void buscar(producto_almacen productos[], int &numero_productos){
    int codigo = 0;
    int posicion = 0;
    cout << "Introduce un código que desees buscar ";
    cin >> codigo;

    posicion = busqueda(productos, codigo, numero_productos);
    if(posicion == PRODUCTO_NO_ENCONTRADO){
        cout << "El código introducido no corresponde a ningún producto" << endl;
    } else {
        mostrar_producto(productos[posicion]);
    }
}

int busqueda(const producto_almacen productos[], int codigo, int numero_productos){
    for (int i = 0; i < numero_productos; i++){
        if(codigo == productos[i].codigo){
            return i;
        }
    }
    return PRODUCTO_NO_ENCONTRADO;
```

```
}

void cargar_datos_desde_archivo(componente componentes[], int &numero_componentes){
    ifstream archivo(NOMBRE_ARCHIVO);

    if (archivo.is_open()){
        while (!archivo.eof() && numero_componentes < MAX_COMPONENTES){
            archivo >> componentes[numero_componentes].codigo;
            archivo.ignore(); // Ignorar el espacio después del código
            getline(archivo, componentes[numero_componentes].descripcion);
            archivo >> componentes[numero_componentes].existencias;
            numero_componentes++;
        }

        archivo.close();
    } else {
        cout << "No se pudo abrir el archivo para cargar datos." << endl;
    }
}

void guardar_datos_en_archivo(componente componentes[], int numero_componentes){
    ofstream archivo(NOMBRE_ARCHIVO);

    if (archivo.is_open()){
        for (int i = 0; i < numero_componentes; i++){
            archivo << componentes[i].codigo << " " << componentes[i].descripcion << " "
                    << componentes[i].existencias << endl;
        }

        archivo.close();
    } else {
        cout << "No se pudo abrir el archivo para guardar datos." << endl;
    }
}
```