```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;
}
```

```cpp
/*
* Juego de la Serpiente v1
* 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 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);
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_serpiente(const char tecla,inc_unitario_posicion& inc_unitario_posicion);
void pintar_serpiente(const posicion serpiente[]);
```

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

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

```
198         dibujar_linea(SIMBOLO_VERTICAL,SIMBOLO_INTERMEDIO,base);
199     }
200     poner_cursor(2,altura - 2);
201     dibujar_linea(SIMBOLO_EXTERIOR,SIMBOLO_HORIZONTAl,base);
202     poner_cursor(2, altura);
203     cout << ARRIBA << "-> Subir " << ABAJO << "-> Bajar " << IZQUIERDA
204          << "-> Izda " << DERECHA << "-> Dcha " << TECLA_FIN << "-> Fin" << endl;
205 }
206
207 bool juego_terminado(char tecla, posicion serpiente[]){
208     return(toupper(tecla) == TECLA_FIN ||
209            serpiente_tocada(serpiente) ||
210            serpiente[0].x == LIMITE_IZQUIERDA ||
211            serpiente[0].x == LIMITE_DERECHA ||
212            serpiente[0].y == LIMITE_SUPERIOR ||
213            serpiente[0].y == LIMITE_INFERIOR);
214 }
215
216 void obtener_direccion_serpiente(const char tecla, inc_unitario_posicion& inc_unitario_posicion) {
217     switch (toupper(tecla)) {
218         case ARRIBA:
219             inc_unitario_posicion.x = 0;
220             inc_unitario_posicion.y = MOVIMIENTO_Y_ASCENDENTE;
221             break;
222
223         case ABAJO:
224             inc_unitario_posicion.x = 0;
225             inc_unitario_posicion.y = MOVIMIENTO_Y_DESCENDENTE;
226             break;
227
228         case IZQUIERDA:
229             inc_unitario_posicion.x = MOVIMIENTO_X_IZQUIERDA;
230             inc_unitario_posicion.y = 0;
231             break;
232
233         case DERECHA:
234             inc_unitario_posicion.x = MOVIMIENTO_X_DERECHA;
235             inc_unitario_posicion.y = 0;
236             break;
237   }
238 }
239
240 void mover_serpiente(posicion serpiente[], inc_unitario_posicion inc_unitario_posicion) {
241     posicion cabeza_anterior;
242     cabeza_anterior = serpiente[0];
243
244     serpiente[0].x = serpiente[0].x + inc_unitario_posicion.x;
245     serpiente[0].y = serpiente[0].y + inc_unitario_posicion.y;
246
247     for (int i = LONG_SERPIENTE - 1; i > 0; --i) {
248         serpiente[i] = serpiente[i - 1];
249     }
250     serpiente[1] = cabeza_anterior;
251 }
252
253
254 void pintar_serpiente(const posicion serpiente[]) {
255     poner_cursor(serpiente[0].x, serpiente[0].y);
256     cout << SERPIENTE;
257
258     for (int i = 1; i < LONG_SERPIENTE - 1; i++) {
259         poner_cursor(serpiente[i].x, serpiente[i].y);
260         cout << CUERPO_SERPIENTE;
261     }
262 }
263
```

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

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

#include <iostream>
using namespace std;

const int MAX_PRODUCTOS = 100;
const int POSICION_NULA = -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 != POSICION_NULA) {
        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 == POSICION_NULA) {
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 == POSICION_NULA) {
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 == POSICION_NULA){
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 POSICION_NULA;
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