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

#include <iostream>
using namespace std;
const int MAX_PRODUCTOS = 100;
const int ALTA = 1;
const int BORRAR = 2;
const int LISTAR = 3;
const int ANADIR = 4;
const int ELIMINAR = 5;
const int BUSCAR = 6;
const int FIN = 7;

struct Producto{
    int codigo = 0;
    string descripcion = " ";
    int reserva = 0;
};

void anadir_producto(Producto producto[],int &numero_productos);
void leer_producto(Producto &producto);
void listar_producto(const Producto producto[], int &numero_productos);
void anadir(Producto Producto[],int &numero_productos);
void eliminar(Producto Producto[],int &numero_productos);
void borrar_producto(Producto producto[], int& numero_productos);
void buscar(Producto producto[], int& numero_productos);

int main()
{
  Producto producto[MAX_PRODUCTOS];
  int opcion = 0;
  int numero_productos = 0;

  while(opcion != FIN){
   cout << "1->Alta 2->Baja 3->Listado 4->Entrada 5->Salida 6-> Buscar 7->Acabar: ";
   cin >> opcion;

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

        case BORRAR:
            borrar_producto(producto,numero_productos);
            break;

        case LISTAR:
            listar_producto(producto,numero_productos);
            break;

        case ANADIR:
            anadir(producto,numero_productos);
            break;

        case ELIMINAR:
            eliminar(producto,numero_productos);
            break;

        case BUSCAR:
            buscar(producto,numero_productos);
            break;
```

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

```
133     bool encontrado = false;
134
135     cout << "codigo: ";
136     cin >> codigo;
137     cout << "Añadir reserva: ";
138     cin >> reserva;
139
140     for (int i = 0; i < numero_productos; i++){
141       if(codigo == Producto[i].codigo){
142         Producto[i].reserva += reserva;
143         encontrado = true;
144
145         cout << "Se han almacenado ("<< Producto[i].reserva << ") de "
146              << Producto[i].descripcion << " con codigo " << Producto[i].codigo << endl;
147       }
148     }
149     if( ! encontrado){
150         cout << " No se ha encontrado ningun codigo en el almacen" << endl;
151     }
152 }
153
154 void eliminar(Producto Producto[],int &numero_productos){
155     int codigo = 0;
156     int reserva = 0;
157     bool encontrado = false;
158
159     cout << "Codigo: ";
160     cin >> codigo;
161     cout << "Eliminar reserva: ";
162     cin >> reserva;
163
164     for (int i = 0; i < numero_productos; i++){
165       if(codigo == Producto[i].codigo){
166         Producto[i].reserva -= reserva;
167         encontrado = true;
168
169         cout << "Se han retirado ("<< Producto[i].reserva << ") de "
170              << Producto[i].descripcion << " con codigo " << Producto[i].codigo << endl;
171       }
172     }
173     if( ! encontrado){
174         cout << " No se ha encontrado ningun codigo en el almacen" << endl;
175     }
176 }
177
178 void buscar(Producto producto[], int& numero_productos){
179     int codigo = 0;
180     bool encontrado = false;
181
182     cout << "Introduce un codigo que desees buscar ";
183     cin >> codigo;
184     for (int i = 0; i < numero_productos; i++){
185         if(codigo == producto[i].codigo){
186             encontrado = true;
187             cout << "Codigo: ";
188             cout << producto[i].codigo << endl;
189             cout << endl;
190             cout << "Descripcion: ";
191             cout << producto[i].descripcion << endl;
192             cout << endl;
193             cout << "Reservas: ";
194             cout << producto[i].reserva << endl;
195     }
196  }
197  if(! encontrado){
198     cout << "Producto no encontrado" << endl;
```

```
199   }
200  }
```

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 esta definida, por eso no es posible mostrar el resultado por pantalla

```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 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 = 60;
const int SERPIENTE_X_INICIAL = 10;
const int LIMITE_SUPERIOR = 2;
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 = "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 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 colision_cuerpo(const posicion serpiente[]);
void inicializar_manzanas(posicion& manzana);
void pintar_manzana(const posicion& manzana);
void borrar_manzana(const posicion& manzana);
bool tocar_manzana(const posicion& manzana, const posicion serpiente[]);
```

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

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

```
            serpiente[0].y == LIMITE_INFERIOR);
}

void obtener_direccion(const char tecla,inc_unitario_posicion& inc_unitario_posicion){
    switch (toupper(tecla)) {
        case ARRIBA:
            inc_unitario_posicion.x = 0;
            inc_unitario_posicion.y = MOVIMIENTO_Y_ASCENDENTE;
            break;

        case ABAJO:
            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 << "o";
    }
}

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 colision_cuerpo(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;
```

```
}

void inicializar_manzanas(posicion& manzana) {
    manzana.x = LIMITE_IZQUIERDA + 4 + rand() % (LIMITE_DERECHA - LIMITE_IZQUIERDA - 5 );
    manzana.y = LIMITE_SUPERIOR + 4 + rand() % (LIMITE_INFERIOR - LIMITE_SUPERIOR - 5);
}

void pintar_manzana(const posicion& manzana){
    poner_cursor(manzana.x, manzana.y);
    cout << MANZANA;
}

void borrar_manzana(const posicion& manzana){
    poner_cursor(manzana.x, manzana.y);
    cout << " ";
}

bool tocar_manzana(const posicion& manzana, const posicion serpiente[]){
    return(manzana.x == serpiente[0].x&&
           manzana.y == serpiente[0].y);
}

void actualizar_puntos(const int puntos){
    poner_cursor(1,26);
    cout << "PUNTOS: " << puntos;
}
```