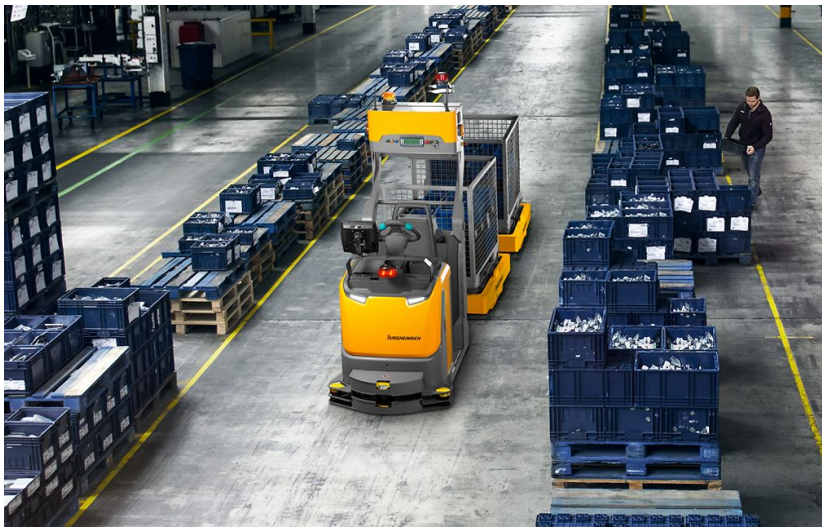
Universidad Francisco de Vitoria

UFV Madrid



Universidad Francisco de Vitoria

# Práctica Vehículo de Guiado Automático (AGV)

Introducción a la programación

# PRÁCTICA 2: VEHÍCULO DE GUIADO AUTOMÁTICO (AGV)

El objetivo de esta práctica es comprobar que el alumno ha comprendido y utiliza correctamente los arrays y las estructuras, así como las estructuras básicas del lenguaje (condicionales, bucles, declaración de variables, …).

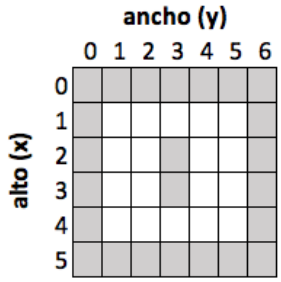
*Nota: el diseño de este sistema no es el más óptimo debido a que no se ha impartido el tema de memoria dinámica y punteros.*

## Especificación

En esta práctica se trata de implementar el programa de control de un Vehículo de Guiado Automático (AGV) para el manejo del mismo.

Los elementos básicos de la sistema son los siguientes:

- **Mapa**




Vista de la planta del almacén (es un ejemplo)

El mapa se representará por un array de dos dimensiones, donde la primera dimensión representa el alto del almacén (x) y la segunda dimensión representa el ancho (y) del almacén. El array puede almacenar los siguiente valores:

- -1, representa una pared o una zona no transitable por el AGV
- 0, representa un espacio por donde el AGV puede transitar
- 2, ubicación actual del AGV

- **Estructura orden**

```
struct orden {
    char info[MAX_CAD];
    int x; // representa la altura en el mapa
    int y; // representa la anchura en el mapa
};
```

La estructura *orden* tiene la información de la orden, por ejemplo, r*ecoger pastillas de freno*, así como la ubicación en el almacén donde se tiene que procesar la orden.

- **Estructura agv**

```
struct agv {
    int mapa[ALTO_ALM][ANCHO_ALM];
    int posX; // posicion dentro del mapa del agv en el alto del mapa
    int posY; // posicion dentro del mapa del agv en el ancho del mapa
    struct orden ordenes[MAX_ORDENES];
};
```

La estructura *agv* tiene el mapa del almacén, la posición actual del AGV en el almacén (el campo *posX* representa la posición del AGV en el *alto* del *mapa* y el campo *posY* representa la posición del AGV en el *ancho* del *mapa*). Se acuerda que la posición inicial del AGV es la *posX = 1* y la *posY = 1*.

Por último, la estructura *agv* almacena un listado de todas las órdenes que tiene que tiene que cumplir el AGV. El listado de ordenes se encuentra inicialmente vacío. Para considerar que una orden es vacía se va a utilizar la siguiente convención: todos los campos de texto deben almacenar la cadena "?" y los campos numéricos deben almacenar un -1.

La operaciones a realizar en el sistema son las siguientes (estas son las funciones y procedimiento que se debe implementar de forma obligatoria y son genéricas, esto es, deben funcionar con cualquier tamaño y mapa de almacén y con cualquier tamaño del listado de órdenes):

```
int insertarOrden(int alto, int ancho, int mapa[][ancho],
                  struct orden ordenes[], int capacidad,
                  struct orden nueva);
```

Esta función permite añadir una *nueva* orden al listado de *ordenes*, que se pasa en el parámetro *nuevo*, siempre que no se haya alcanzado la *capacidad* máxima de la misma y la ubicación de la *nueva* orden en el *mapa* sea válida (posición transitable por el AGV).

La nueva orden se añade en la primera posición libre que se encuentre en el listado de ordenes.

La función devuelve 1 si se ha podido insertar la nueva orden en el listado de órdenes, y devuelve 0 en caso que el listado de órdenes se encuentre lleno y no se pueda almacenar el contacto o bien si la posición de la nueva orden dentro del mapa no es válida (los parámetros de la función *ancho* y *alto* indican el tamaño del *mapa*).

Si el almacenamiento de las órdenes se hace forma ordenada (de menor a mayor), acorde a la distancia de Manhattan respecto a la posición inicial del AGV en el mapa [1, 1], se obtendrá una puntuación superior en este apartado. La formula para calcular la distancia Manhattan entre dos puntos es la siguiente:

$$dM(A,B) = |x_A - x_b| + |y_A - y_B|$$

```
void listarOrdenes(struct orden ordenes[], int max_ordenes);
```

Esta función imprime en pantalla todos las órdenes (no vacías) del listado de órdenes acorde al siguiente formato (tal cual están almacenados - respeta mayúsculas y minúsculas)

```
#orden          info        x     y
#orden          info        x     y
#orden          info        x     y
```

A continuación se muestra un ejemplo:

```
#Ord    Info                          X   Y
=========================================
   1    Pick 16 Front brake disc      3   5
   2    Pick 32 Brake pads            3   9
   3    Pick 16 Brake calipers       10   5
```

En caso que el listado de *órdenes* se encuentre vacío, se debe mostrar en pantalla el mensaje "No hay ordenes".

```
struct orden buscarOrden(struct orden ordenes[], int capacidad,
                         int x, int y);
```

Esta función busca en el listado de *órdenes* una orden en la posición *[x, y]* del almacén. En caso de encontrarla, devuelve la información de la orden en una nueva orden con la información encontrada en el listado de *órdenes*.

En caso de tener más de una orden en la posición buscada, se debe devolver la primera ocurrencia encontrada. Si no hay ninguna orden en la posición indicada, se devolverá una orden vacía (campos de texto con '?' y campos numéricos con valor -1).

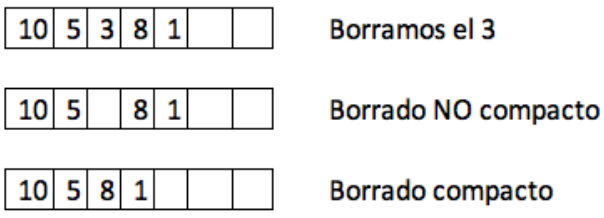
```
int borrarOrdenes(struct orden ordenes[], int capacidad, char
                  info[])
```

Esta función borra del listado de *órdenes* todos los órdenes con la misma información que se pasa en el parámetro *info* (NO DISTINGUE ENTRE MAYÚSCULAS Y MINÚSCULAS, la *info pick* es igual que Pick), y devuelve el número de órdenes borradas.

**NO SE PUEDE HACER USO DE FUNCIONES QUE NO SE HAYAN VISTO EN CLASE**

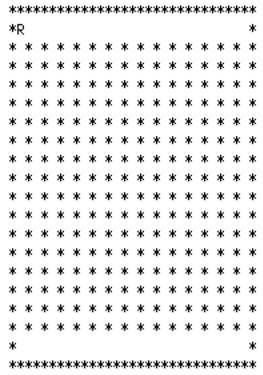
El borrado se puede hacer de dos formas:

- Borrado no compacto: se elimina el contacto poniendo, haciendo que el contacto sea vacío (campos de texto con '?' y campos numéricos con valor -1). Este tipo de borrado deja huecos en la agenda.
- Borrado compacto: elimina el contacto y el hueco dejado en la agenda por el borrado del mismo. Si se realiza este tipo de borrado se obtendrá una puntuación superior en este apartado.



```
void mostrarMapa(int filas, int cols, int mapa[][cols]);
```

Esta función debe imprimir el *mapa* en la pantalla. Las zonas no transitables deben marcarse con un '*', las zonas transitables se representan por un espacio en blanco y la posición del AGV se debe arcar con una 'R'.

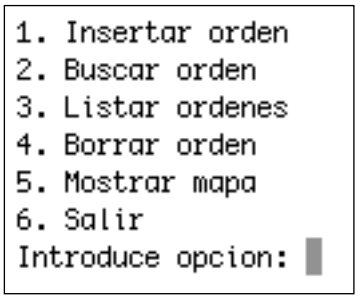


## Funcionamiento del sistema

Asociado a las funciones anteriores (FUNCIONES GENÉRICAS Y QUE DEBEN FUNCIONAR APARA CUALQUIER TAMAÑO DE LISTADO DE ÓRDENES Y CUALQUIER DISTRIBUCIÓN DE ALMACÉN) el sistema debe funcionar acorde a las siguientes especificaciones.

Para definir el listado de órdenes, vamos a suponer que tenemos un máximo de 150 órdenes (pero podría ser cualquier valor). Si se alcanza el numero máximo de órdenes, no se podrán almacenar más órdenes a partir de ese momento, mientras no se libere espacio. Cuando se inicia la sistema, todos las órdenes deben tener en los campos de texto el símbolo '?' y en los campos numéricos un -1 (esto también debe ocurrir cuando se elimina una orden del listado de órdenes)

El funcionamiento de cada una de las opciones es el que se describe a continuación




### Insertar orden

El programa debe pedir los datos de la orden que se quiere introducir por teclado, el usuario los teclea (se almacenan tal cual los teclea el usuario), si es posible almacenar la orden se muestra el mensaje `"Orden insertada con exito"`, y si no es posible el programa debe mostrar el mensaje `"Error, orden no valida"`.

Cuando se termina la operación se debe volver al menú principal.

### Buscar orden

El programa debe pedir la posición de la orden a buscar y el usuario lo introduce por teclado. Si la orden no se encuentra en ley listado de órdenes se muestra el mensaje

"No hay ninguna orden en la posición". En otro caso, se muestra la información del contacto en el siguiente formato:

```
info        posX        posY
```

## Listar órdenes

El programa debe mostrar por pantalla todas las órdenes almacenadas en el listado de órdenes acorde a lo especificado en la función *listarOrdenes*.

## Borrar orden

El programa debe pedir el nombre del contacto a borrar y el usuario lo introduce por teclado. El programa elimina de la agenda los contactos acorde a los especificado en la función *borrarPersona* e imprimir en pantalla el siguiente mensaje "X contacto(s) eliminado(s)" siendo X el numero de contactos eliminados en la función *borrarPersona*.

## Mostrar mapa

El programa debe mostrar el mapa del almacén acorde a lo especificado en la función *mostrarMapa*.

```
[1, 1][1, 2][1, 3][1, 4][1, 5][2, 5][3, 5]
Pick 16 Front brake disc

[3, 5][2, 5][1, 5][1, 6][1, 7][1, 8][1, 9][2, 9][3, 9]
Pick 32 Brake pads

[3, 9][2, 9][1, 9][1, 8][1, 7][1, 6][1, 5][2, 5][3, 5][4, 5][5, 5][6, 5][7, 5][8, 5][9, 5][10, 5]
Pick 16 Brake calipers

[10, 5][9, 5][8, 5][7, 5][6, 5][5, 5][4, 5][3, 5][2, 5][1, 5][1, 6][1, 7][1, 8][1, 9][1, 10][1, 11][1, 12][1, 13][1, 14]
[1, 15][1, 16][1, 17][1, 18][1, 19][1, 20][1, 21][2, 21][3, 21][4, 21][5, 21][6, 21][7, 21]
Pick 8 Master cylinder

[7, 21][8, 21][9, 21][10, 21][11, 21][12, 21][13, 21][14, 21][15, 21][16, 21][17, 21][18, 21][18, 22][18, 23][18, 24][18
, 25][18, 26][18, 27][18, 28][18, 29]
Drop all items
    -1 20 -1 18 -1 12 -1 18 -1 20 -1 22 -1 24 -1 26 -1 28 -1 30 -1 32 -1 34 -1 36 -1 38 -1 40 -1
    -1 21 -1 19 -1 13 -1 19 -1 21 -1 23 -1 25 -1 27 -1 29 -1 31 -1 33 -1 35 -1 37 -1 39 -1 41 -1
    -1 22 -1 20 -1 14 -1 20 -1 22 -1 24 -1 26 -1 28 -1 30 -1 32 -1 34 -1 36 -1 38 -1 40 -1 42 -1
    -1 21 -1 19 -1 15 -1 19 -1 21 -1 23 -1 25 -1 27 -1 29 -1 31 -1 33 -1 35 -1 37 -1 39 -1 41 -1
    -1 20 19 18 17 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 -1
    -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
```

## Salir

El programa debe terminar.

# PRÁCTICA 2: VEHÍCULO DE GUIADO AUTOMÁTICO (AGV)

## Entrega

Esta práctica es de realización obligatoria para todos los alumnos. **Se debe realizar de en grupos de 2 alumnos**.

Plazo de entrega: **23:59 horas del día 16 de junio de 2021**.

Habrá que entregar a través de Canvas un **practica2.c** con el siguiente contenido:

- Cada una de las funciones descritas en el enunciado, acorde a los nombres dados en el mismo.
- Funciones adicionales que se utilicen para mejorar la estructura del código
- Si alguna de las partes no se implementa, el programa debe indicarlo en pantalla con el siguiente mensaje: `"Parte no realizada"`

**La práctica SÓLO debe entregarla uno de los alumnos del grupo (el que aparezca en primer lugar en los nombres comentados)**

Se considerará **plagio** entregar un código idéntico total o parcialmente al de otro compañero. Tampoco se admitirá un código idéntico a otro presente en cualquier página de Internet, incluso aun citando la fuente. **El profesor dispone y usará herramientas de detección automática de plagio.**

## Puntuación

Estos ejercicios obligatorios se puntuará de 0 a 10, nota que se contabilizará en el 20% de la nota final de la asignatura (prácticas) en el caso de la evaluación continua. Las prácticas no entregadas se computarán como un 0.

Elementos a evaluar:

- **Incluir el nombre de los autores como las dos primeras líneas (comentadas) del fichero practica2.c**. Si para la correcta compilación del programa hay que incluir algún parámetro opcional al compilador, se debe incluir en una comentario debajo del nombre de alumno la correcta llamada al compilador.
- La no existencia de errores sintácticos en el código (es decir, **el código debe compilar**). En caso contrario, **una práctica cuyo código fuente dé errores de compilación se considerará suspensa directamente**.
- Se valorará positivamente que no haya errores lógicos.
- Limpieza y claridad en el código.
- Presencia de comentarios en el código.
- Errores descritos en el documento "*Errores a evitar.pdf*" que se encuentra en el aula virtual junto con el enunciado de la práctica.

De forma numérica los pesos de cada parte de la practica son

| | |
|---|---|
| insertarOrden | 1,5 |
| mantener las órdenes ordenadas | 1,5 |
| listarOrdenes | 1 |
| buscarOrden | 1,5 |
| borrarOrdenes | 1,5 |
| mantener el listado de órdenes sin huecos (listado compacto) | 1,5 |
| mostrarMapa | 1,5 |
| **Total** | 10 |

La puntuación máxima a obtener puede ser 10 puntos si se realizan todos los apartados.