# INSTITUTO POLITÉCNICO NACIONAL

# ESCUELA SUPERIOR DE CÓMPUTO

# ESTRUCTURAS DE DATOS

PRÁCTICA #2

SIMULACIÓN CON EL TAD COLA

**GRUPO: 1CM8**

**EQUIPO: LAS MÁS PERRAS**

## INTEGRANTES:


- JIMENEZ DELGADO LUIS DIEGO 2019630461
- SÁNCHEZ CASTRO AARÓN GAMALIEL 2019630079
- SANCHEZ TIRADO CITLALLI YASMIN 2019630096


**PROFESOR:** FRANCO MARTINEZ EDGARDO ADRIAN

**FECHA DE ENTREGA:** 11 DE MARZO 2019

## Introducción

En el desarrollo de esta práctica se busca implementar una cola para representar los funcionamientos como es el de un banco, un supermercado o incluso la ejecución de procesos en el sistema.

## Marco teórico

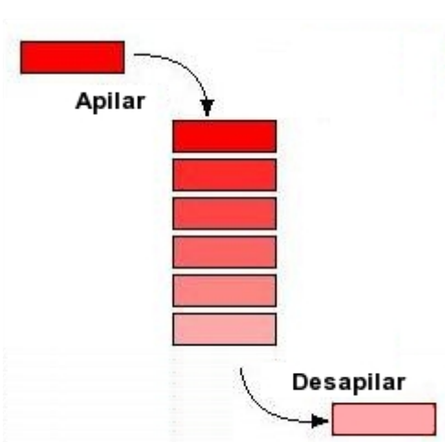
Concepto de cola: Una cola es una estructura de datos, caracterizada por ser una secuencia de elementos en la que la operación de inserción Push se realiza por un extremo y la operación de extracción pop por el otro. También se le llama estructura FIFO (del inglés First In First Out), debido a que el primer elemento en entrar será también el primero en salir.


**Utilización**

Las colas se utilizan en sistemas informáticos, transportes y operaciones de investigación (entre otros), dónde los objetos, personas o eventos son tomados como datos que se almacenan y se guardan mediante colas para su posterior procesamiento. Este tipo de estructura de datos abstracta se implementa en lenguajes orientados a objetos mediante clases, en forma de listas enlazadas.

Una cola es una estructura de datos, caracterizada por ser una secuencia de elementos en la que la operación de inserción Push se realiza por un extremo y la operación de extracción pop por el otro. También se le llama estructura FIFO (del inglés First In First Out), debido a que el primer elemento en entrar será también el primero en salir.

Las colas se utilizan en sistemas informáticos, transportes y operaciones de investigación (entre otros), dónde los objetos, personas o eventos son tomados como datos que se almacenan y se guardan mediante colas para su posterior procesamiento. Este tipo de estructura de datos abstracta se implementa en lenguajes orientados a objetos mediante clases, en forma de listas enlazadas.



**Operaciones de una cola**

**Empty (Vacía):** Se da siempre que la estructura no tiene elementos

**Queue (Agregar un elemento):** Agregar elemento; podemos agregar elementos siempre que la cola no esté llena y estos siempre los agregamos al final.

**Dequeue (Extraer un elemento):** Podemos extraer el elemento insertado más antiguo, el cual se encuentra al frente siempre y cuando la cola no esté vacía

**Front (Frente):** Sin quitar el elemento podemos ver el elemento que está próximo a salir.

**Final (Final):** Sin quitar el elemento podemos ver el elemento que está hasta el final de la cola.

**Element (Elemento):** Sin quitar el elemento podemos ver el elemento que está en una posición dada.

**Size (Tamaño):** Regresa el tamaño total de la cola.

**Destroy (Destruir):** Libera los elementos de la cola y los elimina.

## Simulación No. 1: Súper mercado.

**Implementación**

Mercadito:

En estás líneas de código podemos observar que corresponden a la declaración o bien llamado de las librerías que requerimos.

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <time.h>
#include <string.h>
#include "TADColaDin.h"
#include "rep.h"
```

Aquí hacemos la declaración de constantes

```
#define ancho 90
#define repaintTime     20
#define tiempoEspaciado 200
#define espacioClientes 5
```

Está función nos sirve para ocultar el cursor.

```
void hidecursor(void){
   HANDLE consoleHandle = GetStdHandle(STD_OUTPUT_HANDLE);
   CONSOLE_CURSOR_INFO info;
   info.dwSize = 100;
   info.bVisible = FALSE;
   SetConsoleCursorInfo(consoleHandle, &info);
}
```

En esta función se  dibuja una caja que hace la simulación  del supermecado

Recibe 2 enteros(x, y) que proporcionan las coordenadas en donde se comenzará a dibujar la caja; un entero (NumCaja)que guarda el numero de la caja que se dibujara; un carácter(estado), el cual describe el estado de la caja (si está abierta o cerrada); y finalmente un entero(velocidad), que almacena la  velocidad en que la cajera atenderá los clientes

```
void dibujaCaja(int x, int y, int NumCaja, char estado, int velocidad){
	MoverCursor(x,y);
	printf("***\t");
	if(estado == 'A'){
		MoverCursor(x+7, y);
		printf("                                ");
		MoverCursor(x+7, y);
		printf("Atendiendo");
	}
	else
		if(estado == 'F'){
			MoverCursor(x+7, y);
			printf("                                ");
			MoverCursor(x+7, y);
			printf("CAJA CERRADA\n");
		}
		else{
			MoverCursor(x+7, y);
			printf("                                ");
			MoverCursor(x+7, y);
			printf("Vacio");
		}
	MoverCursor(x+2,y+1);
	printf("*\t Caja %d", NumCaja+1);
	MoverCursor(x,y+2);
	printf("***\t");
	printf("Atendiendo 1 cliente/%d seg\n", velocidad);
}
```

Función Cajeras:

Descripción: Dibuja todas las cajas que se indiquen, además imprime en pantalla: El estado de los clientes, el número de clientes que han sido atendidos, el nombre del supermercado y el estado del mercado (abierto o cerrado).

Recibe La cantidad de cajas que se van a dibujar; el espacio que tendrá la fila de cada cajera; el estado de cada una de las cajas, el nombre del supermercado, la velocidad de llegada de los clientes, la velocidad en que cada cajera atenderá a los clientes, la cantidad de clientes que han llegado y finalmente la cantidad de clientes atendidos.

Devuelve: un entero que representa una posición abajo de lo que dibujamos

```
int Cajeras(int cant, int largo, char estadoCajeras[], char marketName[], int llegadaClientes, int tiempoAtencion[], int cantClientesLlegados, int cantClientesAtendidos){
	int i;
	MoverCursor(0,0);
	p	r	i	n	t	f	(	"
");
	MoverCursor(0,0);
	printf("%s \tRecibiendo 1 cliente/%d seg\tHan llegado %d clientes\t\t%d clientes han sido atendidos\t", marketName, llegadaClientes, cantClientesLlegados, cantClientesAtendidos);
	if(cantClientesAtendidos>100){
		printf("\t\tEL SUPERMERCADO CERRARA\n");
	}
	for(i=0; i<cant; i++){
		dibujaCaja(largo+1, i*4+1, i, estadoCajeras[i], tiempoAtencion[i]);
	}
	return i*4+5;
}
```

Función pintar:
Está función dibuja cada uno de los clientes del supermercado, recibe la fila en la que se dibujara, la posición, el cliente y un carácter que nos indicará si la fila ha terminado.

Para esto creamos un arreglo auxiliar de caracteres y luego en la siguiente línea hacemos una conversión a cadena el entero que esta dentro de nuestra estrucutura. Luego ubicamos las dos cordenadas (x,y) Hacemos una verificación para saber si ya no hay espacio en la fila y dependiendo el caso, se mueve el cursor.

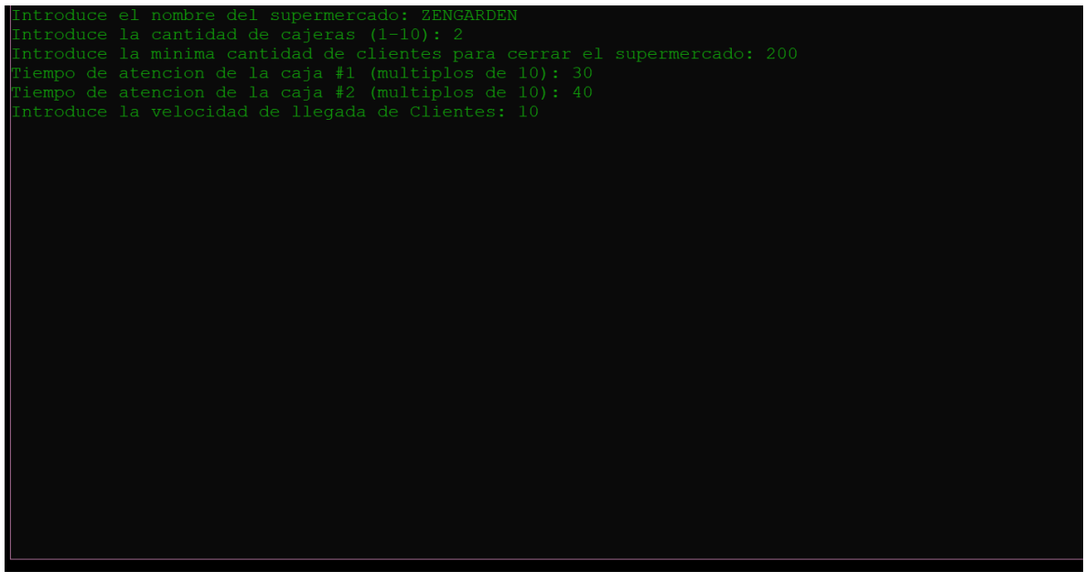
```
void pintar(int filaPintar, int Posicion, elemento Cliente, char
acaboFila){
	char temp[4];
	itoa(Cliente.n,temp,10);
	int x = ancho - espacioClientes*Posicion - 3;
	int y = filaPintar * 4 + 2;
	if(acaboFila == 'S'){
		int i;
		for( i=0; i<x; i++){
			MoverCursor(i,y);
			printf(" ");
		}
	}
	else{
		MoverCursor(x,y);
		printf("     ");
		MoverCursor(x,y);
		fflush(stdout);
		printf("#%s", temp);
	}
	return;
}

void limpiar(int i){
	int y = i*4+2;
	int c;
	for(c=0; c<ancho; c++)
		MoverCursor(0,y);
		printf("#");
	return;
}
```

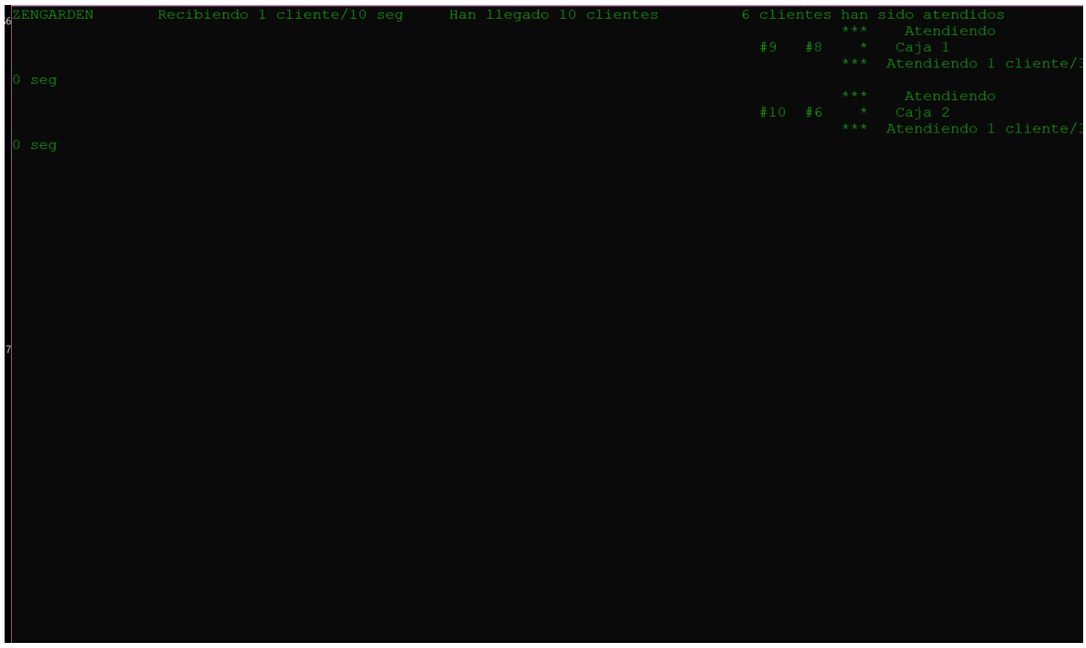
Nuestra función principal se encarga de pedirle al usuario el nombre del supermercado, la cantidad de cajeras, cantidad de clientes, tiempo o bien velocidad en que serán atendidos, velocidad de llegada, así mismo también vamos llamando nuestras funciones para que comience el funcionamiento del super mercado.

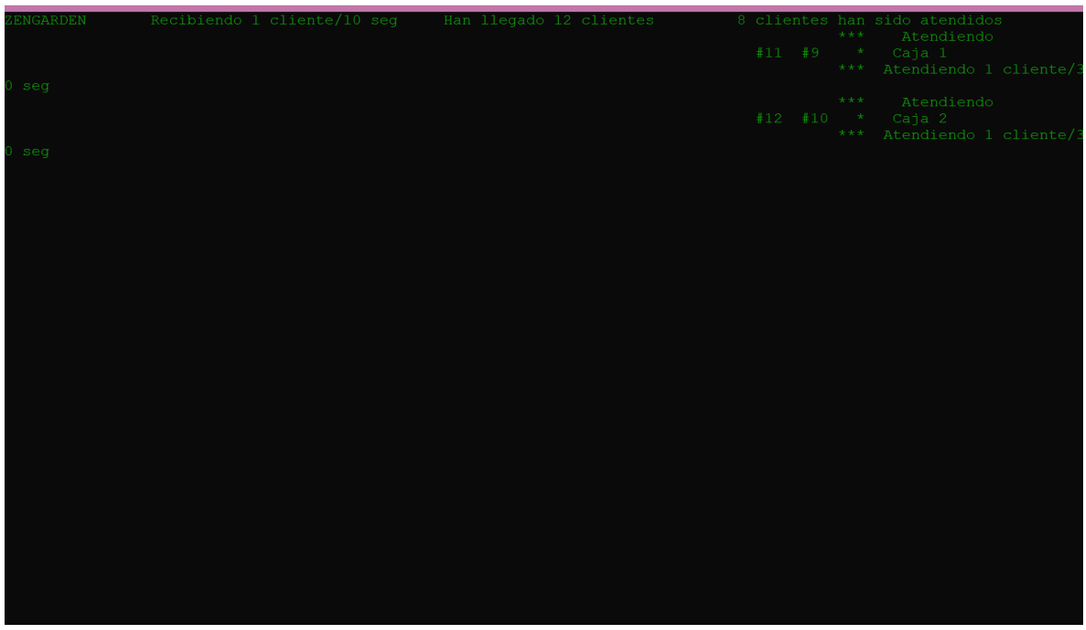
```
int main(void){
	unsigned int tiempo, cliente, atendidos, minClientes, fila,columna,
alto, cantCajeras, filaElegida, llegadaClientes;
	elemento aux;
	char marketName[20], fin = 'N';

	printf("Introduce el nombre del supermercado: ");
	scanf("%s", &marketName);
	printf("Introduce la cantidad de cajeras (1-10): ");
	scanf("%d", &cantCajeras);

	if(cantCajeras>10 || cantCajeras < 1){
		printf("Cantidad de cajeras no valida\n");
		exit(0);
	}

	printf("Introduce la minima cantidad de clientes para cerrar el
supermercado: ");
	scanf("%d", &minClientes);

	char estadoCajeras[cantCajeras];
	int tiempoAtencion[cantCajeras];
	cola filaCajera[cantCajeras];
	int i;
	for( i=0; i<cantCajeras; i++)
		Initialize(&filaCajera[i]);

	for( i=0; i<cantCajeras; i++){
		printf("Tiempo de atencion de la caja #%d (multiplos de 10):
", i+1);
```

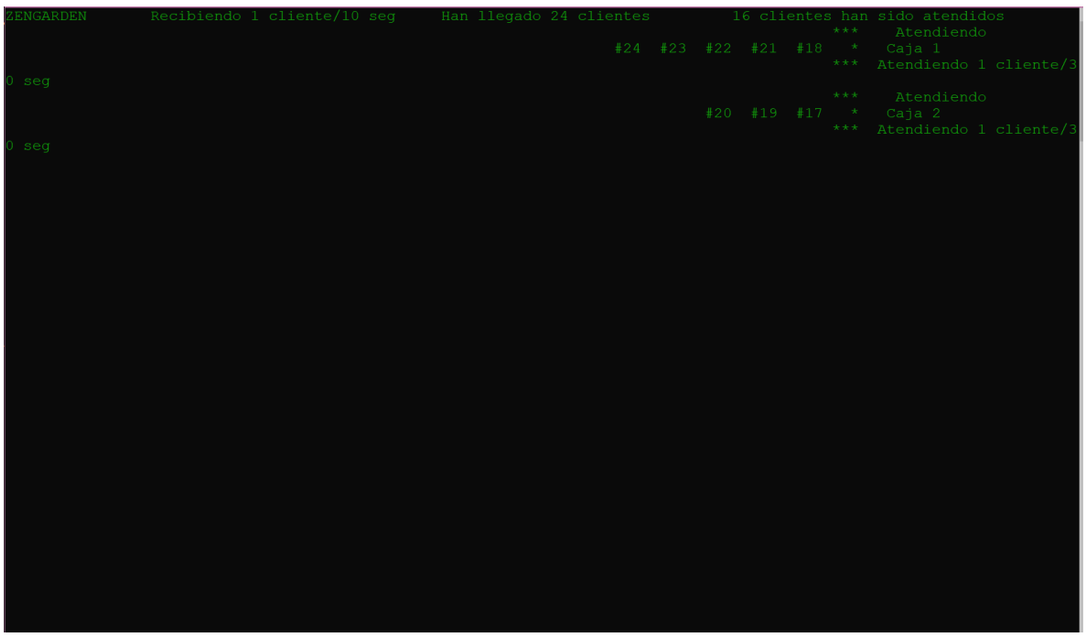
```
            scanf("%d", &tiempoAtencion[i]);
      }
      printf("Introduce la velocidad de llegada de Clientes: ");
      scanf("%d", &llegadaClientes);

      alto = Cajeras(cantCajeras, ancho, estadoCajeras, marketName,
llegadaClientes, tiempoAtencion, cliente, atendidos);
      tiempo = 0; cliente = 0; atendidos = 0;

      system("cls");
      hidecursor();
      srand(time(NULL));

      while(fin == 'N'){
            tiempo++;
            fin = 'S';
            int i;
            for(i=0; i<cantCajeras; i++){
                  if((!Empty(&filaCajera[i])) && estadoCajeras[i] != 'F')
{
                        if(tiempo % tiempoAtencion[i] == 0){
                              aux = Dequeue(&filaCajera[i]);
                              atendidos++;
                        }
                        estadoCajeras[i] = 'A';
                  }
                  else{

                        estadoCajeras[i] = 'Z';
                        if(atendidos>minClientes)
                              estadoCajeras[i] = 'F';
                  }
                  Cajeras(cantCajeras,ancho, estadoCajeras, marketName,
llegadaClientes, tiempoAtencion, cliente, atendidos);
            }


            for(int i=0; i<cantCajeras; i++){

                  for(int k=0; k<=Size(&filaCajera[i]); k++){
                        if(k == Size(&filaCajera[i])){
                              pintar(i,k-1,aux, 'S');
                        }
                        else{
                              aux = Element(&filaCajera[i],k+1);
                              pintar(i, k, aux, 'N');
                        }
                  }
            }

            if((tiempo % llegadaClientes == 0) && atendidos<=minClientes)
{
                  cliente++;
                  aux.n = cliente;
                  filaElegida = rand()%cantCajeras;
                  Queue(&filaCajera[filaElegida], aux);
            }

            Sleep(repaintTime);
            for(int a=0; a<cantCajeras; a++){
                  if(estadoCajeras[a] != 'F')
                        fin = 'N';

                  MoverCursor(0, alto+5);
                  printf("%c\n", fin);
            }
      }

      MoverCursor(0,alto+1);

      return 0;
}
```

## Funcionamiento


Introduce el nombre del supermercado: ZENGARDEN
Introduce la cantidad de cajeras (1-10): 2
Introduce la minima cantidad de clientes para cerrar el supermercado: 200
Tiempo de atencion de la caja #1 (multiplos de 10): 30
Tiempo de atencion de la caja #2 (multiplos de 10): 40
Introduce la velocidad de llegada de Clientes: 10


ZENGARDEN Recibiendo 1 cliente/10 seg Han llegado 10 clientes 6 clientes han sido atendidos
*** Atendiendo
#9 #8 * Caja 1
*** Atendiendo 1 cliente/
0 seg
*** Atendiendo
#10 #6 * Caja 2
*** Atendiendo 1 cliente/
0 seg


ZENGARDEN Recibiendo 1 cliente/10 seg Han llegado 12 clientes 8 clientes han sido atendidos
*** Atendiendo
#11 #9 * Caja 1
*** Atendiendo 1 cliente/
0 seg
*** Atendiendo
#12 #10 * Caja 2
*** Atendiendo 1 cliente/
0 seg




### Errores detectados

Uno de los errores detectados fue que al momento de que usamos una librería como lo es “Windows.h” no compilaba en sistemas operativos como MAC OS.

## Simulación No. 2: Procesos

### Planteamiento del problema

Se plantea que se requiere simular como se gestionan los procesos de un sistemas operativo en un equipo que tiene un monoprocesador sin importar el manejo de prioridades. Siendo que se maneja un cola de los procesos listos para realizar así como otra cola para los procesos terminados.

Cada proceso requiere un nombre, nombre de la activistas que está realizando, un identificador y el tiempo que se requiere para la ejecución. Así como se va ejecutando un Quantum de cada proceso que dure un 1 segundo y el proceso ejecutado se agregue a la cola de listos y así hasta terminar con los procesos por lo que hasta que ya no tengo tiempo de ejecución un proceso se agregue a la cola de los procesos finalizado guardando el tiempo que le tomó a la computadora realizarlo.

### Propuesta de solución

Se crea un proyecto específicamente con el uso de la cola que implemente todos estos métodos para el manejo de los procesos y de las pilas, siendo así que este proceso de las pilas sea como tal la simulación ese requiere.

### Implementación

```
//Librerías a ocupar
#include <curses.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "TADColaDin.h"

//declara los métodos
elemento getProceso();
int ejecutaProceso(cola *listos, cola *finalizados);
void creaResumen(cola *finalizados, int tiempo);
```

```
void imprimeElemento(elemento e);

int main()
{
    char res = 's';
    cola listos, finalizados;
    int tiempo;
    initscr();
    Initialize(&listos);
    Initialize(&finalizados);
    //incializa las pila
PROCESO:
    while (res == 's' || res == 'S')
    {
        //Ejecuta el proceso de solcitar los procesos al ususario.
        Queue(&listos, getProceso());
        printw("\nDeseas ingresar más procesos?:\n");
        scanw("%s", &res);
        clear();
    }
    tiempo = ejecutaProceso(&listos, &finalizados);
    creaResumen(&finalizados, tiempo);
    printw("\nQuieres repetir el programa?:\n");
    scanw("%s", &res);
    //Verifica si el ususario quiere volver a ejecutar el programa.
    if (res == 's' || res == 'S')
        goto PROCESO;
    Destroy(&finalizados);
    Destroy(&listos);
    endwin();
    return EXIT_SUCCESS;
}
```

La función “main” como así lo dice es el que lleva el control de las demás funciones, en si de todo el programa, el flujo de toda la solución propuesta está aquí.

```
/*
FUNCIÓN: int ejecutaProceso(cola *listos, cola *finalizados)
DESCRIPCIÓN: Realiza toda la simulación de la ejecución de los procesos.
RECIBE: cola *listos es el apuntador a la cola de los procesos listos ,
cola *finalizados  es el apuntador a la cola de los procesos finalizados
con el método main main
DEVUELVE: int tiempo el tiempo que se tomó para realizar el proceso.
OBSERVACIONES:
*/
int ejecutaProceso(cola *listos, cola *finalizados)
{
    int tiempo = 1;
    elemento aux, anterior;
          printw("\n--------------------------------  INICIO
-------------------------------\n");
    clear();
    while (!Empty(listos))
    {
        //Cominenza el proceso de ejecución de procesador
        aux = Dequeue(listos);
        --aux.time;
        move(0, 0);
        printw("Proceso Anterior:\n");
        if (tiempo > 1)
        {
            imprimeElemento(anterior);
        }
        else
        {
            printw("NO HAY PROCESO ANTERIOR");
        }
        move(10, 0);
        printw("Proceso Posterior:\n");
        if (!Empty(listos))
        {
            imprimeElemento(Front(listos));
        }
        else
        {
            printw("FIN DE PROCESOS");
```

```
        }
        //Imprime el proceso actual
        move(5, 40);
        printw("Proceso actual:");
        move(6, 40);
        printw("Nombre: %s", aux.processName);
        move(7, 40);
        printw("ID: %s", aux.id);
        move(8, 40);
        printw("Actividad: %s", aux.activity);
        move(9, 40);
        printw("Tiempo: %i", aux.time);
        getch();
        clear();
        anterior = aux;
        if (aux.time == 0)
        {
            aux.time = tiempo;
            Queue(finalizados, aux);
        }
        else
            Queue(listos, aux);
        ++tiempo;
    }
          printw("\n---------------------------------       FIN
--------------------------------\n");
    getch();
    clear();
    return tiempo;
}
```

La función “ejecutaProceso” fue diseñada con el propósito de que se simule la ejecución del los procesos sin prioridad, por lo mismo se requiere de una cola de elementos que dio el usuario previamente, así como un apuntador a la cola de finalizados que ha sido creada e inicializada en un proceso anterior para almacenar todos los procesos que se vayan terminando.

```
/*
FUNCIÓN: void creaResumen(cola *finalizados, int tiempo)
DESCRIPCIÓN: genera el resumen de todos los elementos que se
RECIBE: cola *finalizados es el apuntador a la cola de los procesos
finalizados, int tiempo se recibe el tiempo solicitados.
DEVUELVE: void.
OBSERVACIONES: Se genera el resumen del uso de los procesos.
*/
void creaResumen(cola *finalizados, int tiempo)
{
    elemento aux;
          printw("\n---------------------------------  RESUMEN
---------------------------------\n");
    while (!Empty(finalizados))
    {
        aux = Dequeue(finalizados);
                  printw("ID:  %s  PROCESO:  %s  TIEMPO:  %i  \n",  aux.id,
aux.processName, aux.time);
    }
    printw("\nTiempo total: %i", tiempo);
    getch();
    clear();
}
```

La función “creaResumen” tiene como único objetivo revisar todos los elementos que están en la cola de finalizados para obtener el resumen del tiempo que se tardaron en ejecución así como señala el tiempo total de la ejecución por eso requiere de estos dos elementos  señalados (cola finalizados y int tiempo) usando las operaciones de Dequeue de la cola para obtener todos los elementos almacenados en la misma.

```
/*
FUNCIÓN: void imprimeElemento(elemento e)
DESCRIPCIÓN: Dado un elemento imprime el proceso y el tiempo restante.
RECIBE: elemento e a imprimir.
DEVUELVE: void.
OBSERVACIONES: el elemento requiere tener información.
*/
void imprimeElemento(elemento e)
```

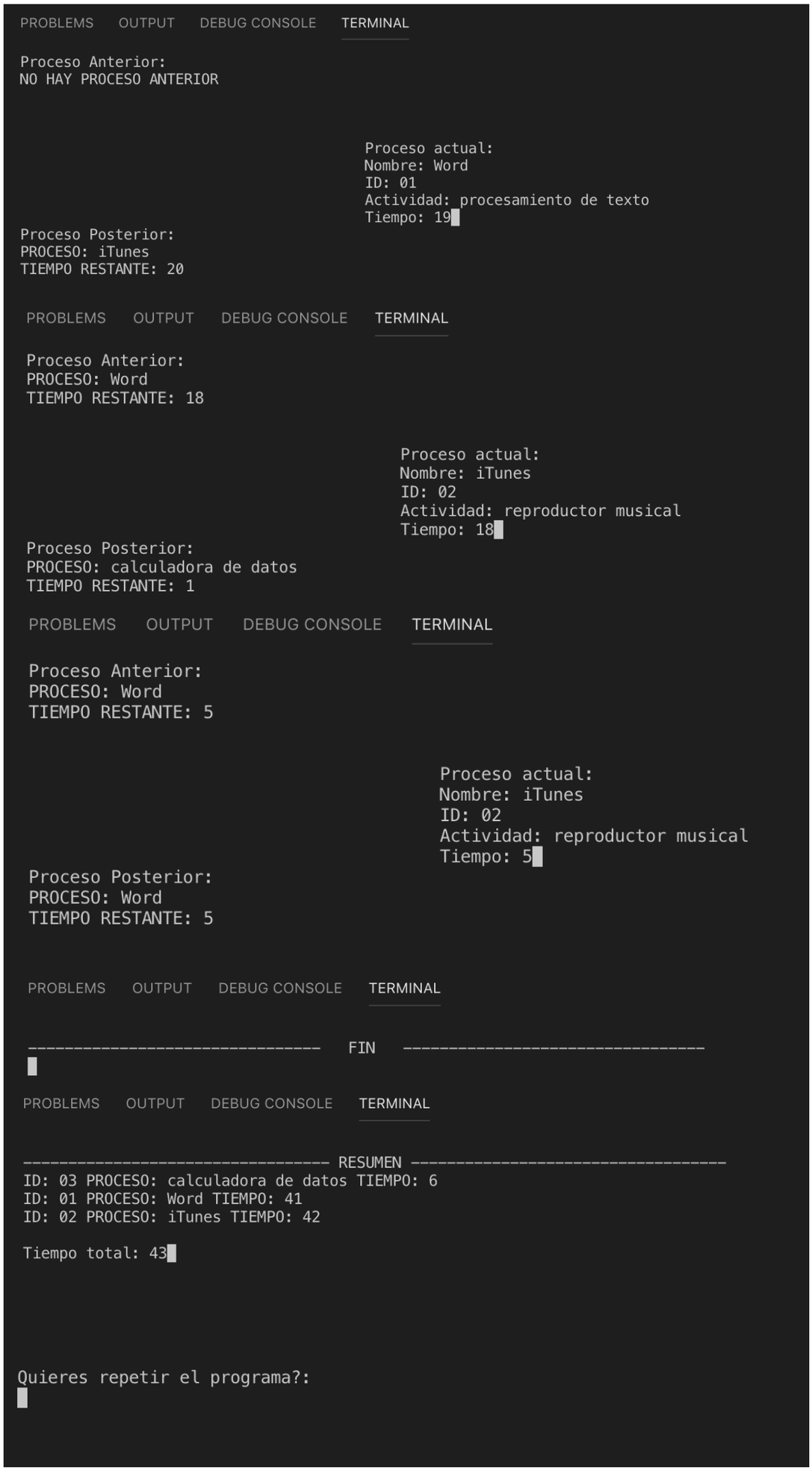
```
{
    printw("PROCESO: %s\n", e.processName);
    printw("TIEMPO RESTANTE: %i\n", e.time);
    return;
}
```

La función “imprimeElemento” es especificamente para imprimir algunas de las características de un elemento e dado. Así que por más sencillo que se escuche es su único objetivo.

```
/*
FUNCIÓN: elemento getProceso()
DESCRIPCIÓN: Hace el proceso de solicitar datos al usuario acerca de los procesos
RECIBE: void.
DEVUELVE: elemento e que es el elemento con los datos solicitados al usuario.
OBSERVACIONES: El tiempo es validado para evitar errores con el tiempo
*/
elemento getProceso()
{
    elemento e;
    printw("Nuevo Proceso\n");
    printw("Ingresa nombre del proceso:\n");
    scanw("%[^\n]%*c", e.processName);
    printw("Ingresa actividad del proceso:\n");
    scanw("%[^\n]%*c", e.activity);
    printw("Ingresa id del proceso:\n");
    scanw("%s", e.id);
TIEMPO:
    printw("Ingresa tiempo del proceso:\n");
    scanw("%i", &e.time);
    if (e.time < 1)
    {
        printw("No se permiten Tiempos negativos\n");
        goto TIEMPO;
    }
    fflush(stdin);
    return e;
}
```

La función “getProceso” esta hecha con el propósito de que el usuario le de características a un elemento e, así que por así decirlo el elemento e está siendo pedido directamente al usuario. Acepta que el nombre del proceso y el nombre de la actividad sean cadenas con espacios para legibilidad y valida que el tiempo de ejecución sea mayor 0 ya que es posible que si un tiempo no lo es, el método de ejecución no funcione correctamente.

**Funcionamiento**




**Errores detectados**

Se ha detectado problemas en la ejecución con windows, en todos los sistemas operativos UNIX funciona sin ningún problema dado que incluyen la librería ncurses.

**Posibles mejoras**

Se considera que la interfaz se puede mejorar considerablemente para la visualización de procesos mientras que se ejecutan.

## Simulación No. 3: Atención en un banco

**Planteamiento del problema**

Se requiere simular la atención de personas en un banco, cuidando que sean respetadas las políticas de atención de este, y evitando que las personas no dejen de ser atendidas. Las políticas mencionadas son las siguientes:

- El banco cuenta con una a diez cajas en operación para atender a tres filas; clientes, usuarios y preferentes.
- Los **usuarios** del banco son personas sin una cuenta en el mismo, son atendidos según la disponibilidad de la caja, nunca permitiendo que pasen más de 5 personas de las otras dos filas sin que un usuario sea atendido.
- Los clientes **preferentes** son personas con cuenta en el banco y privilegios preferenciales, son atendidos por cualquier cajero disponible con mayor prioridad que los **clientes** y **usuarios**.
- Los **clientes** del banco son personas con cuenta en el banco (sin privilegios preferenciales) son atendidos por cualquier cajero disponible y nunca dejan de ser atendidos por alguna caja.

El programa funciona según los datos ingresados por el usuario, los cuales son (todos los tiempos son dados en milisegundos y múltiplos de 10):

- Número de cajeros en el banco (entre 1 y 10).
- Tiempo de atención para los cajeros.
- Tiempo de llegada de los clientes del banco.
- Tiempo de llegada para los usuarios del banco.
- Tiempo de llegada para los clientes preferentes del banco.

Además, deberá mostrar lo siguiente:

- Llegada de los distintos clientes a las respectivas filas.
- Clientes en espera en cada fila
- Cliente que es atendido en cada caja, mostrar el tipo de cliente.
- Cajeros sin atender clientes, en caso de que así sea.

**Propuesta de solución**

Se crea un programa cuyo funcionamiento está basado en el TAD cola, debido a la naturaleza del problema; es necesario que la cola sea dinámica para poder almacenar tantos clientes como se requiera.

Al implementar la librería TADColaDin se permite la creación de las tres colas para almacenar a los tres tipos de clientes del banco, posteriormente se agrega un nuevo cliente a cada fila, de acuerdo con los tiempos establecidos por el usuario de la simulación.

***Figura 1: Creación de las filas, 1 para clientes, 2 para preferentes y 3 para usuarios***

***Figura 2: Después de t milisegundos, las filas comienzan a llenarse.***

Las cajas comienzan a brindar atención a los clientes preferentes, en caso de no haber preferentes, pasan a verificar la existencia de los clientes y en caso de no haber clientes verifican la existencia de usuarios. Los clientes preferentes son atendidos primero, cada tres preferentes se atiente un cliente, y cada cinco personas con cuenta atendidas (clientes y preferentes) se atiende un usuario.

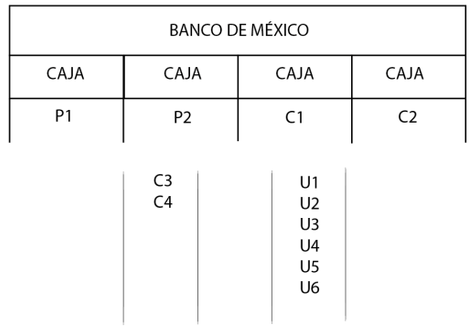



***Figura 3: Las cajas comienzan a dar atención a los clientes, preferentes y clientes primero.***

## Implementación

Como primer punto de la implementación se tiene a las variables que han sido declaradas como globales.

```
//VARIABLES GLOBALES, CONVIENE MANEJARLAS ASÍ PORQUE VARIAS FUNCIONES LAS UTILIZAN
int num_cajas,aux,tClints,tPrefs,tUsus,tiempoAtencion;

/*
	FUNCIÓN: printCajas(int *cordCliCajas).
	RECIBE: int *cordCliCajas (APUNTADOR A UN ARREGLO DE COORDENADAS UBICADO EN simulacion()).
	DEVUELVE: EXPLÍCITAMENTE NO DEVUELVE ALGÚN VALOR. MODIFICA EL ARREGLO DE COORDENADAS PARA LOS ESPACIOS EN BLANCO
		DE CADA CAJA.
	DESCRIPCIÓN: ENCARGADA DE IMPRIMIR LAS CAJAS EN LA CONSOLA, ADEMÁS DE GUARDAR LAS COORDENADAS
		DEL INTERIOR DE CADA CAJA, ESTO CON EL FIN DE SABER DÓNDE COLOCAR A LOS CLIENTES AL MOMENTO
		DE SER ATENDIDOS.
	OBSERVACIONES: NINGUNA.
*/
void printCajas(int *cordCliCajas){
	int cordX;//SE ESTABLECEN LAS COORDENADAS (X) PARA COLOCAR A LOS CLIENTES POSTERIORMENTE
	int ANCHO_DISPLAY=num_cajas*ANCHO_CAJA;
	system("cls");

printf("------------------------------------------------------------");
	printf("\n\t\t\tBANCO DE MEXICO\n");

printf("------------------------------------------------------------\n");
	for(aux=0;aux<num_cajas;aux++){
		printf("#|CAJ|");//SE IMPRIMEN LAS CAJAS, EL ESPACIO EN BLANCO SE LLENARÁ POSTERIORMENTE CON C#,U#,P#
	}
	printf("\n");
	for(aux=0;aux<num_cajas;aux++){
		printf("#|     |");//SE IMPRIMEN LAS CAJAS, EL ESPACIO EN BLANCO SE LLENARÁ POSTERIORMENTE CON C#,U#,P#
	}
```

```
	//EL SIGUIENTE FOR SE UTILIZA PARA LLENAR UN ARREGLO QUE USAREMOS AL REALIZAR LA ANIMACIÓN DE ATENDER CLEINTES

for(cordX=2,aux=0;cordX<ANCHO_DISPLAY,aux<num_cajas;cordX=cordX+ANCHO_CAJA,aux++){
		cordCliCajas[aux]=cordX;//GUARDAMOS LAS COORDENADAS CALCULADAS EN UN ARREGLO DE DIMENSIÓN num_cajas
	}
	return;
}
```

Esta primera función tiene como propósito imprimir en la consola las cajas con las que cuenta el banco, lo primero que realiza es la impresión del nombre del banco dentro de guiones, simulando un letrero. Posteriormente con ayuda de la sentencia “for”, se imprimen barras separadas por “#” con la leyenda “CAJ” indicando que se trata de una caja, igualmente con una sentencia “for” se imprimen barras con espacios en blanco entre ellas y separadas por “#”. Los espacios en blanco de éstas últimas barras indican el área donde se atenderá a los clientes, se guarda en el arreglo cordCliCajas la componente X de dicha área.

```
/*
	FUNCIÓN: printCola(cola *cola, int cordX, int cordY).
	RECIBE: cola *cola (APUNTADOR A LA COLA QUE SE IMPRIMIRÁ), int cordX (COMPONENTE X DEL PUNTO DONDE SE IMPRIMIRÁ LA COLA)
		int cordY (COMPONENTE Y DEL PUNTO DONDE SE IMPRIMIRÁ LA COLA).
		(APUNTADOR A UN ARREGLO DE ELEMENTOS GLOBAL, CORRESPONDIENTE A CADA TIPO DE CLIENTE).
	DEVUELVE: EXPLÍCITAMENTE NO DEVUELVE ALGÚN VALOR. IMPRIME LA COLA ESPECIFICADA.
	DESCRIPCIÓN: ENCARGADA DE IMPRIMIR LA COLA DEL TIPO DE CLIENTE SOLICITADA, SE DEBE INDICAR LA POSICIÓN
		EN DONDE SE DESEA IMPRIMIR. LOS CLIENTES A IMPRIMIR NO DEBERÁN SOBREPASAR EL LÍMITE DE CLIENTE POR FILA (ÚNICAMENTE PARA MOSTRAR).
	OBSERVACIONES: SE VALIDA QUE LA COLA ESPECIFICADA NO ESTÉ VACÍA, DE LO CONTRARIO NO ES EJECUTADA LA FUNCIÓN
		POR COMPLETO. ADEMÁS, SI EL NÚMERO DE ELEMENTOS CONTENIDOS EN LA COLA ES MAYOR AL TAMAÑO DEL ARREGLO
		DE ELEMENTOS GLOBAL, SE IMPRIME UN + INDICANDO QUE LA LONGITUD DE LA COLA ES MAYOR. SE UTILIZA LA FUNCIÓN
		MoverCursor() INCLUÍDA EN LA LIBRERÍA Gotoxy.h
*/
void printCola(cola *cola,int cordX, int cordY){
	int tamCola;
	if(!Empty(cola)){
		tamCola=Size(cola);
		for(aux=0;aux<MAX_PER_FILA;aux++){//IMPRIMIMOS EL ARREGLO DE ELEMENTOS, EL NUMERO CON EL TIPO DE PERSONA
			if(Element(cola,aux+1).n!=0){//
				MoverCursor(cordX,cordY++);//MOVEMOS EL CURSOR PARA IMPRIMIR DE MANERA VERTICAL Y CENTRADA

printf("%c%i",Element(cola,aux+1).tipo,Element(cola,aux+1).n);
			}
		}
		if(tamCola>MAX_PER_FILA)
			printf("+");//SI EL TAMAÑO DE LA FILA SOBREPASA AL MÁXIMO PERMITIDO, ENTONCES IMPRIMO UN +
	}
	return;
}
```

La función “printCola” utiliza un par coordenado (cordX,cordY) para marcar el inicio de la impresión para la cola especificada (*cola), antes de entrar completamente a la función se verifica que la cola no esté vacía, de lo contrario no hay elementos a imprimir y la función sale. En caso de haber elementos dentro de ella, se recorre la cola hasta la posición “MAX_PER_FILA” (se modificó la librería “TADColaDin.c” para retornar “0” cuando una posición no es válida, en lugar de terminar el programa), si el elemento contenido es diferente de “0” se mueve el cursor una posición abajo verticalmente y se imprime el elemento en esa posición. En el caso de que el tamaño de la cola sea mayor a “MAX_PER_FILA” se imprime un “+” indicando que hay elementos en esa cola.

```
/*
      FUNCIÓN: atenderPersona(int numeroCaja, elemento pAtendida, int
*cordCliCajas).
      RECIBE: int numeroCaja (POSICIÓN DE LA CAJA DENTRO DEL ARREGLO
cordCliCajas), elemento pAtendida
            (PERSONA QUE ES ATENDIDA POR ESA CAJA), int *cordCliCajas
(APUNTADOR A UN ARREGLO CUYA DIMENSIÓN
            ES IGUAL AL NÚMERO DE CAJAS Y QUE CONTIENE LAS COORDENADAS DE
LOS ESPACIOS EN BLANCO DE CADA CAJA).
      DEVUELVE: EXPLÍCITAMENTE NO DEVUELVE ALGÚN VALOR. IMPRIME AL
CLIENTE EN LA VENTANA INDICADA.
      DESCRIPCIÓN: ENCARGADA DE COLOCAR AL CLIENTE ESPECIFICADO EN LA
CAJA INDICADA, ESTO SE LOGRA A TRAVÉS DEL ARREGLO
            DE COORDENADAS cordCliCajas, PUES ÉL ALMACENA LA POSICIÓN DE
LOS ESPACIOS EN BLANCO DE CADA CAJA.
      OBSERVACIONES: SE UTILIZA LA FUNCIÓN MoverCursor() INCLUÍDA EN LA
LIBRERÍA Gotoxy.h
*/
void atenderPersona(int numeroCaja, elemento pAtendida, int
*cordCliCajas){
      MoverCursor(cordCliCajas[numeroCaja],Y_CAJAS);
      printf("%c%i",pAtendida.tipo,pAtendida.n);
      return;
}
```

Utilizando únicamente la función “MoverCursor” (incluida en “Gotoxy.h”), “atenderPersona” imprime el elemento indicado (pAtendida) en la coordenada dada por “cordCliCajas” y “Y_CAJAS”, indicando el índice que se desea consultar del arreglo “cordCliCajas” mediante “numeroCaja”.

```
/*
      FUNCIÓN: simulacionBanco(int tiempoAtencion)
      RECIBE: int tiempoAtencion (TIEMPO QUE TARDARÁN LAS CAJAS EN
ATENDER A LOS CLIENTES)
      DEVUELVE: EXPLÍCITAMENTE NO DEVUELVE ALGÚN VALOR.
      DESCRIPCIÓN: ENCARGADA DE CORRER LA SIMULACIÓN, HACIENDO USO DE LAS
FUNCIONES PREVIAMENTE DEFINIDAS.
            EN ESTA FUNCIÓN SE AGREGAN LOS CLIENTES A SUS COLAS
CORRESPONDIENTES, ADEMÁS; LAS CAJAS VERFICAN
            LA EXISTENCIA DE CLIENTES EN LAS COLAS Y ATIENDEN A LOS
MISMOS, MOSTRANDO TODO ESTE PROCESO A TRAVÉS
            DE FUNCIONES QUE IMPRIMEN EL ESTADO ACTUAL DE LA COLA Y DE
LAS CAJAS.
      OBSERVACIONES: PARA FINES PRÁCTICOS LA SIMULACIÓN SE DETIENE AL
ATENDER A 100 PERSONAS (CLIENTES DE CUALQUIER TIPO)<.
*/
void simulacionBanco(int tiempoAtencion){
      int cordCliCajas[num_cajas]; //AREGLO CON LAS COORDENADAS PARA
DIBUJAR A LOS CLIENTES AL MOMENTO EN QUE SON ATENDIDOS
      i                              n                              t
pAtendidas=0,cLLegados=0,uLLegados=0,pLLegados=0,tTranscurrido=0;
      int prefAten=0,ususAten=0,cliAtend=0; //CONTADORES QUE AYUDAN A
CUMPLIR LAS POLITICAS DE LA EMPRESA
      int *arregloCord=cordCliCajas;// APUNTADOR AL ARREGLO DE
COORDENADAS, PARA FACILITAR SU MANIPULACIÃ‘N
      cola cClientes,cPreferentes,cUsuarios;//DECLARAMOS TRES COLAS PARA
TRES TIPOS DE CLIENTES
      elemento persona;
      Initialize(&cClientes);
      Initialize(&cPreferentes);
      Initialize(&cUsuarios);//TERMINAMOS DE INICIALIZAR LAS COLAS Y
DECLARAR VARIABLES

      printCajas(arregloCord);//NECESITAMOS IMPRIMIR LAS CAJAS, PASAMOS
UN APUNTADOR AL ARREGLO DE COORDENADAS PARA CLIENTES

      while(pAtendidas!=100){//EL PROCESO DE LLEGADA Y ATENCIÃ‘N
FUNCIONARÃ CUANDO pAtendidas<100
            tTranscurrido++;
            //CONDICIONES PARA LA LLEGADA DE CLIENTES
            if(tTranscurrido%tClints==0){
                  persona.n=++cLLegados;
                  persona.tipo='C';
                  Queue(&cClientes,persona);
            }
```

```
            if(tTranscurrido%tPrefs==0){
                  persona.n=++pLLegados;
                  persona.tipo='P';
                  Queue(&cPreferentes,persona);
            }
            if(tTranscurrido%tUsus==0){
                  persona.n=++uLLegados;
                  persona.tipo='U';
                  Queue(&cUsuarios,persona);
            }

            printCola(&cClientes,X_CLIENTES,Y_FILAS);
            printCola(&cPreferentes,X_PREFERENTES,Y_FILAS);
            printCola(&cUsuarios,X_USUARIOS,Y_FILAS);

            if(num_cajas==1){//UN SOLO CAJERO
                  if(tTranscurrido%tiempoAtencion==0){
                        if(!Empty(&cPreferentes)){//HAY PERSONAS EN LA
FILA PREFERENTE.
                              //¿CUÁNTOS PREFERENTES HE ATENDIDO?
                              if(prefAten<3){//MENOS DE 3, PUEDO SEGUIR
ATENDIENDO

atenderPersona(0,Dequeue(&cPreferentes),cordCliCajas);
                                    prefAten++;
                                    pAtendidas++;//PERSONAS (INDISTINTAS)
ATENDIDAS
                              }else{//YA ATENDÍ TRES, ¿YA ATENDÍ A 5
PERSONAS CON CUENTA?
                                    if((prefAten+cliAtend)<=5){//NO, ME
TOCA ATENDER UN CLIENTE ENTONCES.
                                          prefAten=0;//REINICIAMOS EL
CONTADOR
                                          if(!Empty(&cClientes)){//NOS
ASEGURAMOS DE QUE SEA POSIBLE, LO ES
                                                goto ATENDER_CLIENTE;
                                          }else{//NO HAY NADIE EN CLIENTES
                                                goto ATENDER_USUARIO;
                                          }
                                    }else{//SI, HAY QUE ATENDER A UN
USUARIO
                                          prefAten=0;
                                          cliAtend=0;
                                          goto ATENDER_USUARIO;
                                    }
                              }
                        }else  if(!Empty(&cClientes)){//PREFERENTES VACÍA
PERO CLIENTES NO.
                              ATENDER_CLIENTE:

atenderPersona(0,Dequeue(&cClientes),cordCliCajas);
                              cliAtend++;
                              pAtendidas++;
                        }else{//PREFERENTES Y CLIENTES VACÍAS, ¿HAY
USUARIOS?.
                              ATENDER_USUARIO:
                              if(!Empty(&cUsuarios)){

atenderPersona(0,Dequeue(&cUsuarios),cordCliCajas);
                                    pAtendidas++;
                              }
                        }
                  }
            }else{//MAS DE UN CAJERO ¿CUÁNTOS TENGO?
                  for(aux=0;aux<num_cajas;aux++){//RECORREMOS TODOS
                        if(tTranscurrido%tiempoAtencion==0){//LES TOCA
ATENDER
                              if(!Empty(&cPreferentes)){//HAY PERSONAS EN
LA FILA PREFERENTE.
                                    //¿CUÁNTOS PREFERENTES HE ATENDIDO?
                                    if(prefAten<3){//MENOS DE 3, PUEDO
SEGUIR ATENDIENDO

atenderPersona(aux,Dequeue(&cPreferentes),cordCliCajas);
                                          prefAten++;
                                          pAtendidas++;//PERSONAS
(INDISTINTAS) ATENDIDAS
```

```
                                    }else{//YA ATENDÍ TRES, ¿YA ATENDÍ A 5
PERSONAS CON CUENTA?
                                          if((prefAten+cliAtend)<=5){//NO,
ME TOCA ATENDER UN CLIENTE ENTONCES.
                                                prefAten=0;//REINICIAMOS
EL CONTADOR
                                                if(!Empty(&cClientes)){//
NOS ASEGURAMOS DE QUE SEA POSIBLE, LO ES
                                                goto ATENDER_CLIENTE_N;
                                                }else{//NO HAY NADIE EN
CLIENTES
                                                      g     o     t     o
ATENDER_USUARIO_N;
                                                }
                                          }else{//SI, HAY QUE ATENDER A UN
USUARIO
                                                prefAten=0;
                                                cliAtend=0;
                                                goto ATENDER_USUARIO;
                                          }
                                    }
                              }else  if(!Empty(&cClientes)){//PREFERENTES
VACÍA PERO CLIENTES NO.
                                    ATENDER_CLIENTE_N:

atenderPersona(aux,Dequeue(&cClientes),cordCliCajas);

                                    cliAtend++;
                                    pAtendidas++;
                              }else{//PREFERENTES Y CLIENTES VACÍAS, ¿HAY
USUARIOS?.
                                    ATENDER_USUARIO_N:
                                    if(!Empty(&cUsuarios)){

atenderPersona(aux,Dequeue(&cUsuarios),cordCliCajas);
                                          pAtendidas++;
                                    }
                              }
                        }
                  }
            }
            Sleep(100);
      }
      return;
}
```

La función anterior declara las colas correspondientes a clientes, usuarios y preferentes. Además, declara algunos contadores que se utilizan para verificar la llegada de las personas, así como su atención cada cierto tiempo determinado por “tiempoAtencion”.

Con ayuda del operador % se comprueba si es tiempo de que un cliente, preferente o usuario llegue al banco, si es el caso se incremente un contador identificador y se agrega a un nuevo elemento, junto con el tipo de cliente y es agregado a su cola correspondiente.

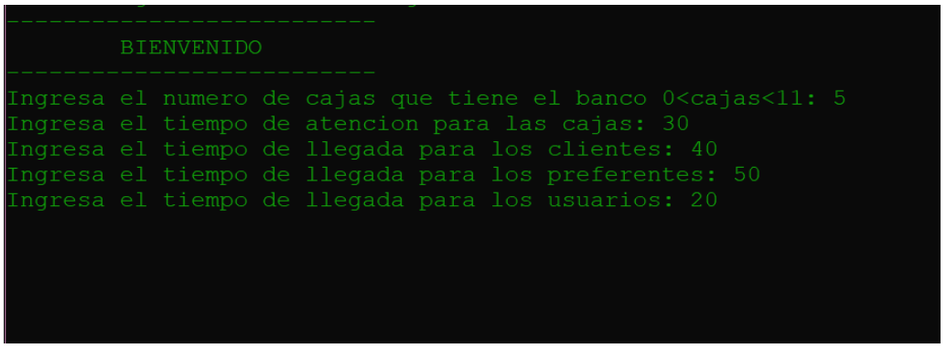
Tanto para una caja operando como para múltiples cajas, el funcionamiento es bastante parecido pues cuando se trata de múltiples cajas, basta con utilizar la sentencia “for” para recorrer todas las cajas disponibles y aplicar la metodología. El proceso consiste los siguientes puntos:

1. Se verifica si es tiempo de atender a un cliente.
2. Se observa si hay clientes en la fila preferente; en caso de no haber preferentes vaya al **punto 3**. En caso de sí haber preferentes verificar si ya han sido atendidas 3 personas de este tipo, si ya han sido atendidas vaya al **punto 3**; si no han sido atendidas, atender un preferente e incrementar los contadores “prefAtend” (preferentes atendidos) y “pAtendidas” (personas atendidas).
3. Se verifica si la suma de preferentes atendidos y clientes atendidos sea menor o igual a 5, en caso de no serlo vaya al **punto 4.** Caso contrario (sí es menor), reiniciar el contador de preferentes atendidos y verificar si la

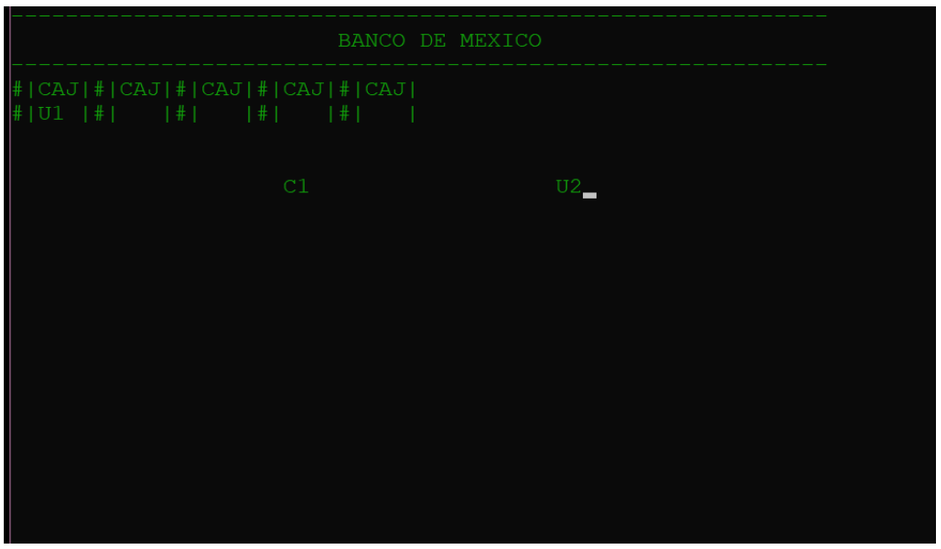
cola de clientes está vacía; en caso de estarlo vaya al **punto 4,** en caso de no estarlo atender un cliente e incrementar los contadores “cliAtend” (clientes atendidos) y “pAtendidas”.

4. Verificar si la cola de usuarios está vacía, en caso de estarlo se vuelve al **punto 1.** En caso de no estarlo, atender a un usuario e incrementar los contadores “usuAtend” (usuarios atendidos) y “pAtendidas”.

**Funcionamiento**



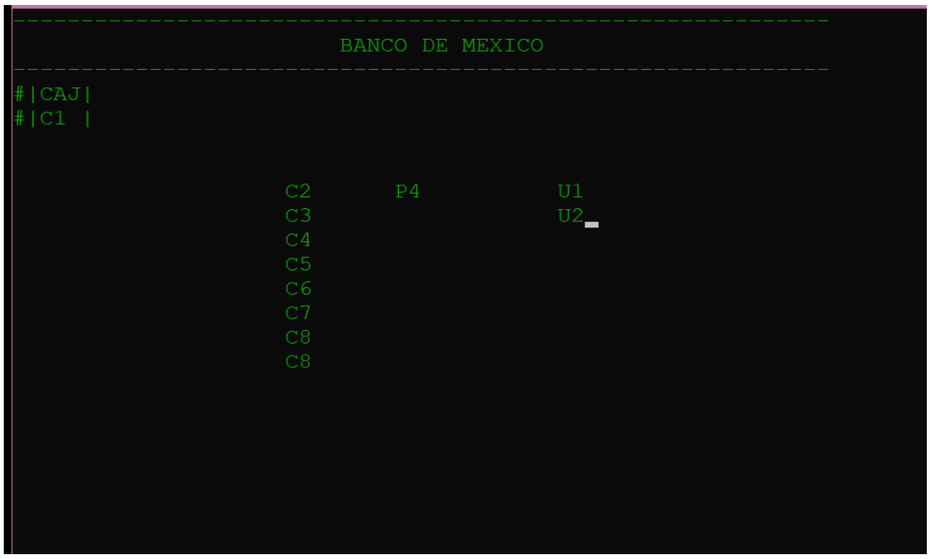


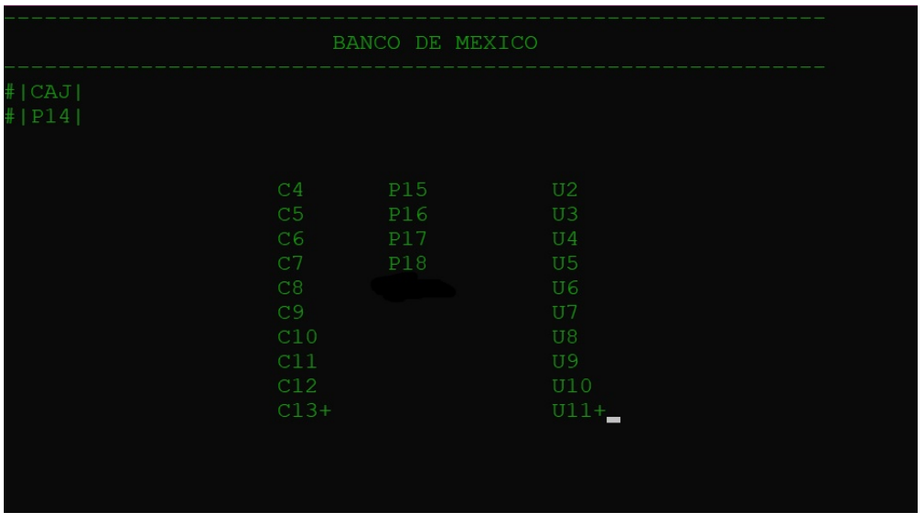


```
------------------------------------------------------------
                      BANCO DE MEXICO
------------------------------------------------------------
#|CAJ|#|CAJ|#|CAJ|#|CAJ|#|CAJ|
#|C4 |#|U8 |#|U9 |#|U6 |#|   |


                    C4      P3
```

```
--------------------------
       BIENVENIDO
--------------------------
Ingresa el numero de cajas que tiene el banco 0<cajas<11: 1
Ingresa el tiempo de atencion para las cajas: 20
Ingresa el tiempo de llegada para los clientes: 10
Ingresa el tiempo de llegada para los preferentes: 20
Ingresa el tiempo de llegada para los usuarios: 30
```

```
------------------------------------------------------------
                      BANCO DE MEXICO
------------------------------------------------------------
#|CAJ|
#|P1 |


                    C1      P1
                    C2
```




**Errores detectados**

Debido a una mala implementación de la función Sleep(miliseconds) el programa no actualiza de manera inmediata la pantalla, por lo que pueden observarse a personas siendo atendidas y formadas al mismo tiempo.

**Posibles mejoras**

Buscar un tiempo para la función Sleep(miliseconds) adecuado en el que se actualice la pantalla de manera casi inmediata. Además, en esta simulación se propuso como límite (para fines prácticos) 100 personas atendidas, ya que el usuario de la simulación puede introducir una combinación que nunca la detenga, una mejora para el programa es eliminar ese límite y finalizar cuando ya no haya clientes. Se debe verificar que la combinación introducida pueda terminarse en un tiempo considerable.

## Conclusiones

**Conclusión de Luis Diego Jiménez Delgado**

A mi parecer la cola es una de la mejores estructuras para el manejo de prioridades considerando que estas mismas pueden incluir otro tipo de elementos que hagan que la pro estructura se vuelva util para millones de procesos a realizar, Su característica FIFO es lo que mayormente la distingue, así como mencioné en un ejemplo de las tortillas. Las colas es, pos así decirlo, una de los TAD que más suelen verse en uso en la vida real, normalmente pueden ser aplicadas para hacer

muchas cosas. Para finalizar es increíble cómo es que la relación de este TAD con la vida real se le puede dar demasiada aplicación en el mundo real.

**Conclusión de Aarón Gamaliel Sánchez Castro**

Una cola es un concepto que está presente en nuestra vida cotidiana, ser capaces de abstraer esta idea es de suma importancia para generar el TAD cola y ser capaces de resolver múltiples problemas en la programación. Al igual que el TAD pila, visto con anterioridad, es posible implementar el TAD cola de manera dinámica y estática; y cada una de estas variantes se debe adaptar al problema.

Al comprender el funcionamiento de los TAD's mencionados anteriormente, es fácil comprender el funcionamiento del TAD lista, cabe mencionar que la cola es simplemente una lista FIFO (*First In First Out*).

**Conclusión de Citlali Yasmín Sánchez Tirado**

Corroboro que una cola es un TDA ya que está dedicado al almacenamiento y manipulación de los elementos, su funcionalidad siempre es la misma independientemente de la implementación que se haya utilizado.

Su funcionamiento cumple con la regla de FIFO que quiere decir que el orden de la salida de los elementos es el mismo que la entrada de estos.

Esto se debe a que las colas están diseñadas para devolver los elementos ordenados tal como llegan, para esto nos dimos cuenta que, se puede decir que estas poseen un punto de acceso y otro de salida que lógicamente están ubicados en los extremos opuestos.