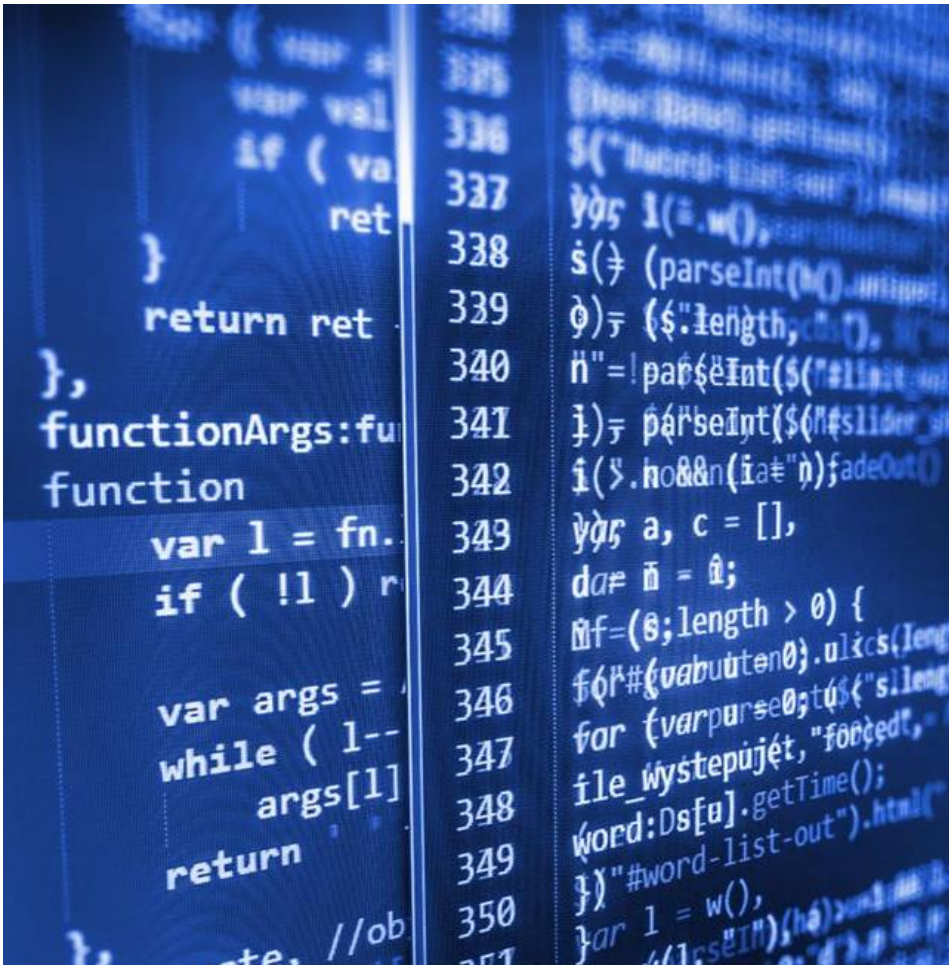


PRACTICA 2

SIMULACIONES CON EL TADCOLA

Equipo: DGA TEAM

INTITUTO POLITECNICO NACIONAL

ESCOM

ESCOM

INTEGRANTES

OLEDO ENRIQUEZ GILBERTO IRVING

ANALIS RAMIREZ DAMIAN

MENDIETA TORRES ALFONSO ULISES

GRUPO
1CM13

UNIDAD DE APRENDIZAJE

Estructuras de Datos

Practica 02: Simulaciones con el TADCola

Introducción.

Tipos de datos abstractos.

Una abstracción es la simplificación de un objeto o de un proceso de la realidad en la que sólo se consideran los aspectos más relevantes.

La abstracción se utiliza por los programadores para dar sencillez de expresión al algoritmo.

La abstracción tiene dos puntos de vista en programación:

1. Funcional.

2. De datos.

Tipos de abstracción.

La abstracción funcional: - Permite dotar a la aplicación de operaciones que no están definidas en el lenguaje en el que se está trabajando.

- Se corresponden con el mecanismo del subprograma (acción que se realiza y argumentos a través de los cuales toma información y devuelve resultados).

- Es irrelevante cómo realiza la acción y no importa su tiempo de ejecución.

Las abstracciones de datos (= Clase): - Permiten utilizar nuevos tipos de datos que se definirán especificando sus posibles valores y las operaciones que los manipulan.

- Cada operación constituye una abstracción funcional.

Definición Un tipo abstracto de datos (TAD) es un tipo definido por el usuario que:

- Tiene un conjunto de valores y un conjunto de operaciones.

- Cumple con los principios de abstracción, ocultación de la información y se puede manejar sin conocer la representación interna. Es decir, los TADs ponen a disposición del programador un conjunto de objetos junto con sus operaciones básicas que son independientes de la implementación elegida.

Metodología de la programación de un TAD.

Debido al proceso de abstracción, la programación de un TAD deberá realizarse siguiendo tres pasos fundamentales:

1. Análisis de datos y operaciones.

2. Especificación del tipo de datos abstracto.

3. Implementación. [1]

TAD 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.

Usos concretos de la cola

Ejemplo de Cola

La particularidad de una estructura de datos de cola es el hecho de que sólo podemos acceder al primer y al último elemento de la estructura. Así mismo, los elementos sólo se pueden eliminar por el principio y sólo se pueden añadir por el final de la cola.

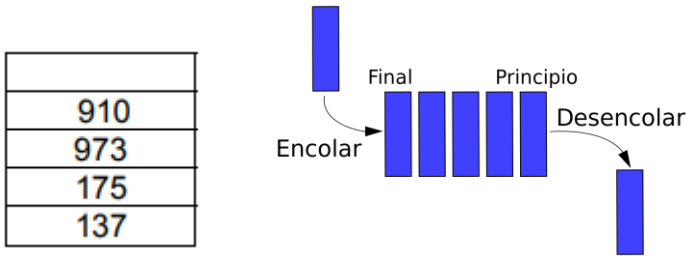
Ejemplos de colas en la vida real serían: personas comprando en un supermercado, esperando para entrar a ver un partido de béisbol, esperando en el cine para ver una película, una pequeña peluquería, etc. La idea esencial es que son todas líneas de espera.

En estos casos, el primer elemento de la lista realiza su función (pagar comida, pagar entrada para el partido o para el cine) y deja la cola. Este movimiento está representado en la cola por la función pop o desencolar. Cada vez que otro elemento se añade a la lista de espera se añaden al final de la cola representando la función push o encolar. Hay otras funciones auxiliares para ver el tamaño de la cola (size), para ver si está vacía en el caso de que no haya nadie esperando (empty) o para ver el primer elemento de la cola (front).

Tipos de colas.

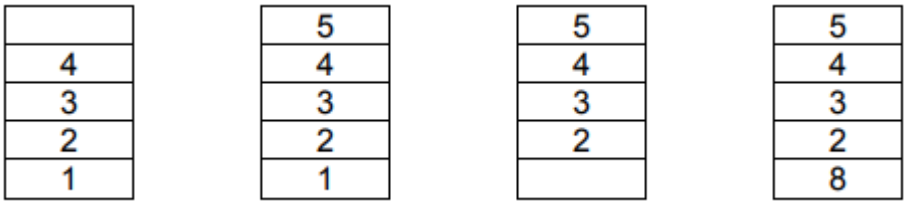
Colas simples:

Se inserta por un sitio y se saca por otro, en el caso de la cola simple sebinserta por el final y se saca por el principio. Para gestionar este tipo de cola hay que recordar siempre cual es el siguiente elemento que se va a leer y cuál es el último elemento que se ha introducido.



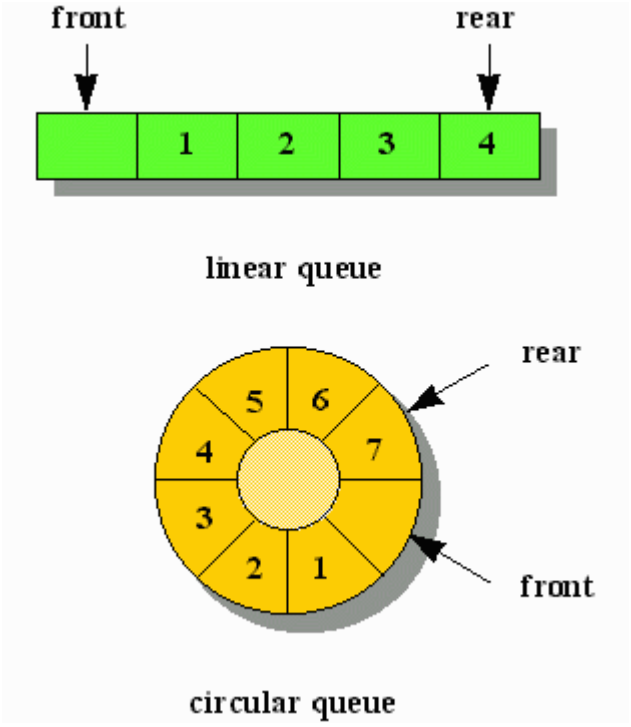
Colas circulares:

En las colas circulares se considera que después del último elemento se accede de nuevo al primero. De esta forma se reutilizan las posiciones extraídas, el final de la cola es a su vez el principio, creándose un circuito cerrado.



Lo que se ha hecho es insertar (5), sacar (1), e insertar (8).

Se sabrá que una tabla está llena cuando "rear" y "front" estén en una posición de diferencia.



Colas con prioridad:

Las colas con prioridad se implementan mediante listas o arrays ordenados. No nos interesa en este caso que salgan en el orden de entrada sino con una prioridad que le asignemos. Puede darse el caso que existan varios elementos con la misma prioridad, en este caso saldrá primero aquel que primero llego (FIFO). [2]

El tipo de Dato COLA (queue) es una estructura de datos que organiza los datos de la siguiente manera:

- A partir de una dirección de memoria, los datos se almacenan sucesivamente como si fueran una colección ordenada de elementos (cartas, clientes frente a una ventanilla, libros en un estante, mensajes a una casillade correo, etc), y
- En cualquier momento se puede recuperar el objeto que se encuentra primero en la estructura (es decir, el primero que fue guardado).

Las características del TAD Cola son:

- Homogénea: ya que almacena elementos del mismo tipo.
- Dinámica: ya que permite agregar y sacar elementos durante la ejecución del programa.
- Acceso FIFO: (First In First Out): los elementos se recuperan en orden inverso al que fueron almacenados.
- En cualquier momento se puede recuperar el elemento que se encuentra en el frente de la cola (es decir, el primero que fue guardado). [3]

Operaciones básicas del TAD Cola.

- ***Inicializar cola (Initialize):*** Recibe una cola y la inicializa para su trabajo normal.
- ***Encolar (Queue):*** Recibe una cola y un elemento y agrega el elemento al final de ella.
- ***Desencolar (Dequeue):*** Recibe una cola y remueve el elemento del frente retornándolo.
- ***Es vacía (Empty):*** Recibe la cola y devuelve verdadero si esta esta vacía.
- ***Frente (Front):*** Recibe una cola y retorna el elemento del frente.
- ***Final (Final):*** Recibe una cola y retorna el elemento del final.
- ***Elemento (Element):*** Recibe una cola y un número de elemento de 1 al tamaño de la cola y retorna el elemento de esa posición.

- ***Eliminar cola (Destroy):*** Recibe una cola y la libera completamente.
- ***Tamaño (Size):*** Recibe una cola y devuelve el tamaño de esta [4]

Planteamiento del Problema.

Simulación 01.

- Simular la atención de clientes en un supermercado, el cual deberá de atender al menos 100 clientes por día para no tener pérdidas, por lo que una vez que ya se atendieron a más de 100 personas y no hay gente formada en las cajas puede cerrar la tienda. Mientras no se cierre la tienda, las personas podrán seguir llegando con productos a las cajas.

Simulación 02.

- Simular la ejecución de los procesos gestionados por el sistema operativo en un equipo monoprocesador sin manejo de prioridades.
- Manejando únicamente el cambio de la cola de listos a ejecución y una vez terminado el proceso este se envía a la cola de terminados

Simulación 03.

- Simular la atención de personas en un banco, cuidando sean respetadas las políticas de atención del mismo y evitando que las personas no dejen de ser atendidas.

Diseño y funcionamiento de la solución.

Simulación 01 Supermercado.

Partimos de una idea general en la cual tratamos de analizar qué es lo que sucede en "la realidad" y poder replicarlo en un lenguaje de programación.

Sabemos que de acuerdo a lo habitual lo que sucede dentro de un supermercado es lo siguiente:

Existen diferentes cajas y de igual modo existen clientes, los cuales llegan a formarse a las cajas para ser atendidos, y ellos escogen las cajas de forma aleatoria (generalmente la caja elegida es la que tiene la *cola* menor o la mas en atender rápida)

Esto es lo que se trata de replicar dentro de la simulación 01, se indica cuantas cajas están atendiendo el supermercado, las cuales deben de estar dentro de un rango mayor a 1 y menor a 10 además a cada una de las cajas se le asigna un tiempo de atención, por otro lado, también se establece un tiempo de llegada de los clientes los cuales se van a formar en alguna caja de manera aleatoria.

Adicional a esto mencionamos el estado de la caja, es decir si la caja está atendiendo o está vacía.

Se tiene una condicional, la cual es indicada dentro del planteamiento del problema, esta condición explica que el supermercado deberá atender al menos a 100 clientes y no debe haber clientes formados para poder cerrar, esto es algo importante al momento de buscar una solución, ya que nos hace pensar que debemos llevar un conteo general de los clientes atendidos por todas las cajas además de consultar constantemente cual es el estado de la caja (atendiendo – vacía – cerrada)

Ejemplo:

¿Qué es lo que sucede cuando se utiliza el concepto *Cola*?

Supongamos lo siguiente:

El número de cajas que atendiendo el supermercado son 3

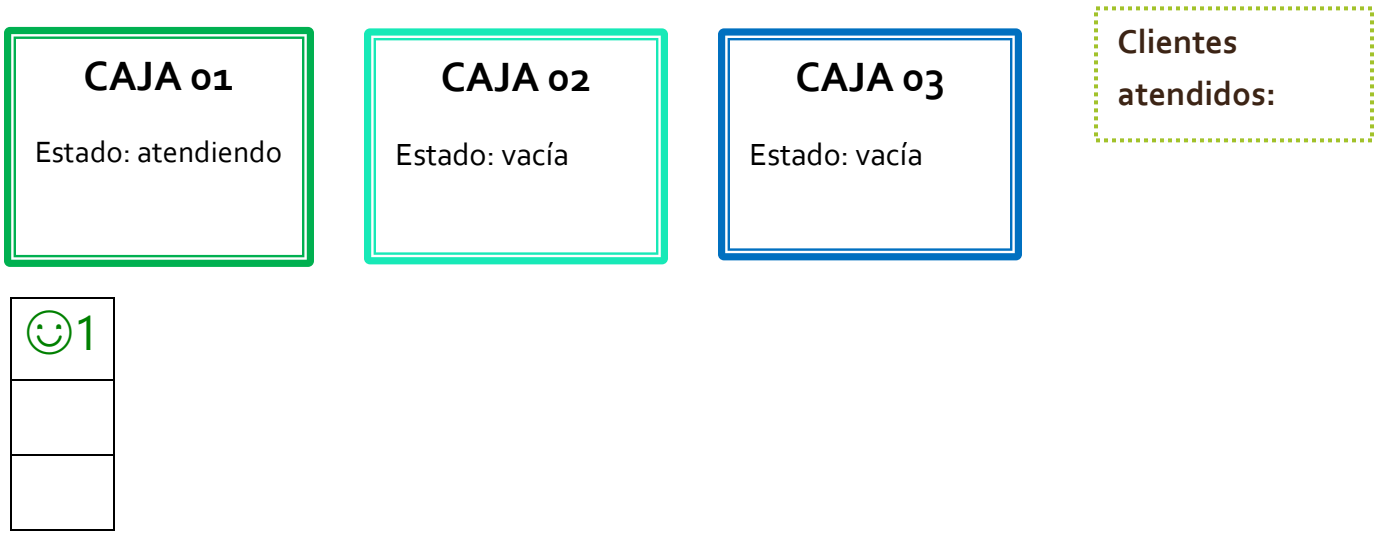
El tiempo de atención cajera 1 es de 5 min, el de la cajera 2 es de 2 min y el de la cajera 3 es de 7 min (tiempos realistas de atención dentro de un supermercado convencional)

El tiempo de llegada de los clientes es de 6 min

Y el supermercado necesita atender a 4 clientes para poder cerrar

(Las suposiciones son a fin de lograr una mejor explicación, más clara y entendible)

Suponemos que llega 1 cliente y se forma en la caja 01.



Suponemos que el cliente 1 se desencola y se atiende.

CAJA 01

Estado: atendiendo

☺1

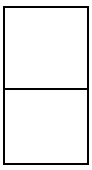
CAJA 02

Estado: vacía

CAJA 03

Estado: vacía

Clientes atendidos:



Suponemos que el cliente ya fue atendido y un minuto después llega un cliente y se forma en la caja 03, se aumenta el número de clientes atendidos.

CAJA 01

Estado: vacía

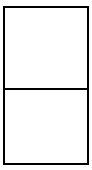
CAJA 02

Estado: vacía

CAJA 03

Estado: vacía

Clientes atendidos: 1



Suponemos que el cliente 2 se desencola y se atiende.

CAJA 01

Estado: vacía

CAJA 02

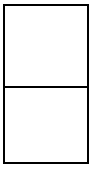
Estado: vacía

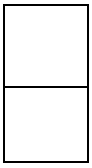
CAJA 03

Estado: vacía

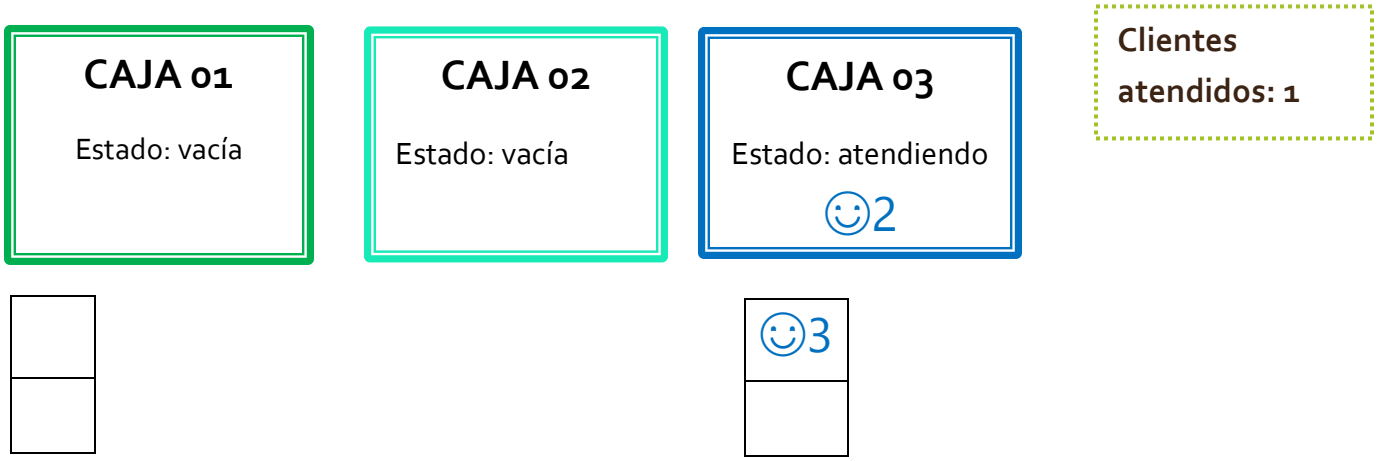
☺2

Clientes atendidos: 1

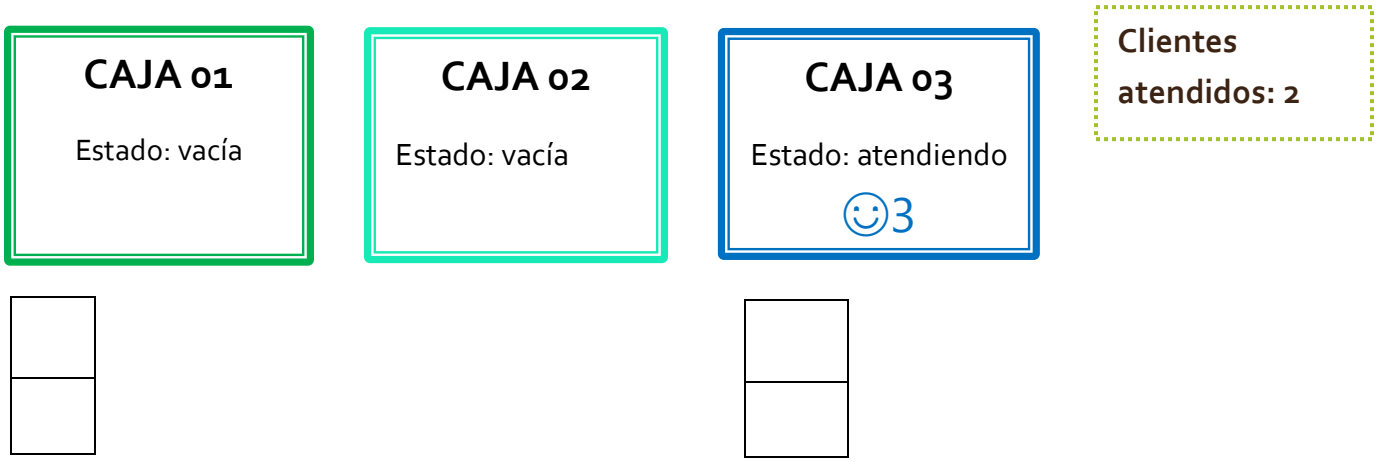




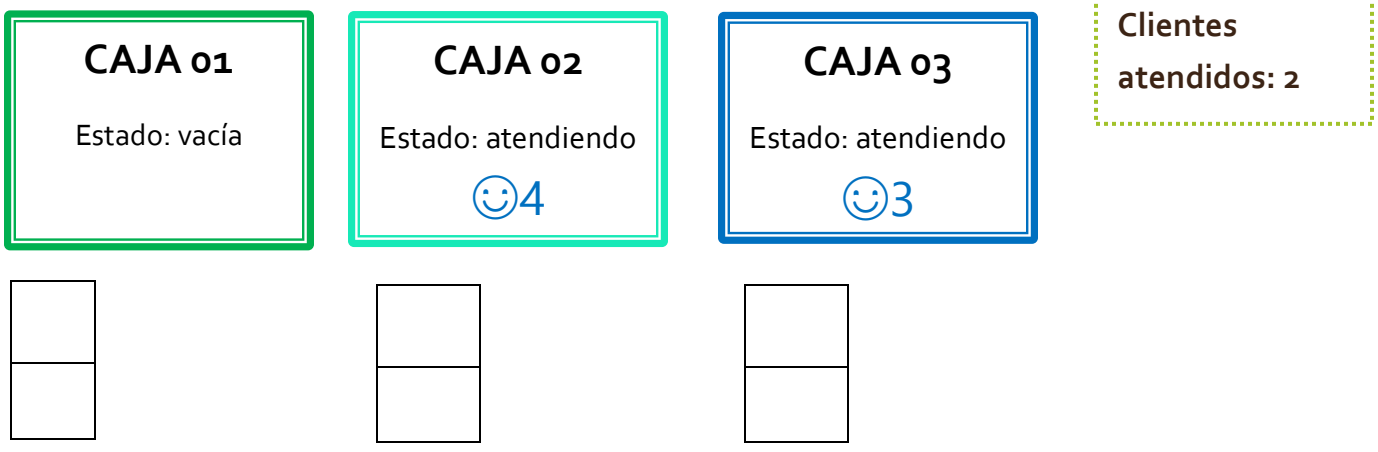
Suponemos que 5 minutos después llega un cliente y se forma en la caja 03.



Suponemos que 2 minutos después se termina de atender al cliente 2 y se comienza a atender al cliente 03, se aumenta el número de clientes atendidos.

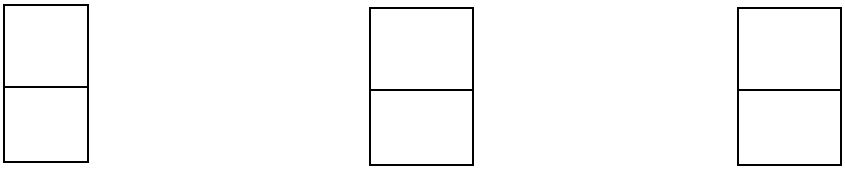


Suponemos que 3 minutos después llega un cliente y se forma en la caja 02 y comienza a ser atendido



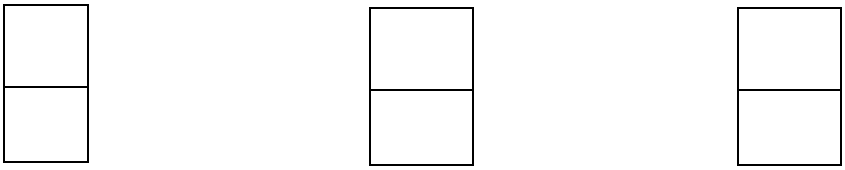
Suponemos que 1 minutos después termina de ser atendido el cliente 03, se aumenta el número de clientes atendidos.

CAJA 01	CAJA 02	CAJA 03	Clientes atendidos: 3
Estado: vacía	Estado: atendiendo ☺4	Estado: vacía	



Suponemos que 1 minutos después termina de ser atendido el cliente 04, se aumenta el número de clientes atendidos.

CAJA 01	CAJA 02	CAJA 03	Clientes atendidos: 4
Estado: vacía	Estado: vacía	Estado: vacía	



Ambas condiciones son cumplidas el número de clientes atendidos es al menos 4 y el estado de las 3 cajas es vacía.

SUPERMERCADO CERRADO

CAJA 01	CAJA 02	CAJA 03	Clientes atendidos: 4
Estado: cerrada	Estado: cerrada	Estado: cerrada	

Simulación 02 Cola de procesos del SO .

Obtención del tiempo total de ejecución

Se especificó que en esta simulación se debe manejar el paso de la cola de espera a ejecución, y una vez que se termina su tiempo restante de ejecución, el proceso pasa a la cola de terminados.

Para darle solución, planteamos decrecer el tiempo de cada proceso en 1 unidad, para esto debíamos tener una cola de tiempos, de modo que a razón de 1 segundo (o quantum de tiempo para despachar los procesos) se desencola 1 elemento de la cola de tiempos, se muestra su tiempo restante actual (tiempo para que finalice), y posteriormente este tiempo se decremento en 1 unidad, de modo que en una posterior impresión el tiempo va a tener una unidad menos. En el caso de que haya llegado a 0, se desencolará permanentemente de la cola tiempo (sin volverse a encolar) junto con los otros elementos, es decir, se desencolara un elemento de nombres, id y actividad (pues son los correspondientes al tiempo del proceso que acaba de finalizar) y se pasarán automáticamente a la cola de terminados y ya no serán introducidos nuevamente en el ciclo que conforma el paso de la cola de espera a ejecución.

Asimismo, se nos plantea que cada que un proceso acabe se imprime su nombre, id, descripción y el tiempo total de ejecución, es decir, el tiempo que paso en espera más su tiempo de ejecución. Para tal efecto propusimos crear una variable de tipo entero que se inicializara en 0 y que fuera incrementándose en una unidad cada vez que pasara el tiempo para despachar los procesos, es decir, cada segundo.

Todo esto podría meterse en un ciclo, repitiéndose hasta que no quedaran elementos en las colas, esto es, siempre que aún quede algún proceso con un tiempo restante mayor o igual a 1 segundo.

Al finalizar el último proceso, es decir, cuando el último tiempo llega a 0, se tiene que en la variable propuesta (tiempoEjecución) se habrá dado un incremento en su valor (a razón de 1 unidad) por cada proceso que se ejecutaba, es decir, cada segundo que pasaba.

Simulacion 03 Banco.

En esta simulación, se ve el funcionamiento del TAD Cola como en la vida real, hay clientes que entran al banco y se forman en una cola y esperan ser atendidos por un cajero.

La simulación cuenta con 3 colas una de usuarios, clientes y preferentes, cada cierto tiempo llegará un visitante y se formará en su respectiva cola, cuando el primer visitante llega, si algún cajero le toca atender lo hará, tomando en cuenta las políticas de atención del banco donde los preferentes siempre tienen que ser atendidos por cualquier cajero disponible, los clientes que

son atendidos por cualquier cajero y siempre tienen que ser atendidos por algún cajero, los usuarios son los menos atendidos pero no pueden pasar más de 5 entre preferentes o clientes sin que un usuarios sea atendido.

Implementación de la solución.

TAD Cola

Todo comienza con la implementación de un TAD cola respetando la teoría explicada en la introducción sobre los tipos de datos abstractos.

Por tanto tenemos un TAD cola con sus operaciones, las cuales están listas para ser utilizadas dentro del programa principal.

```
#include "TADColaDin.h"
#include <stdio.h>
#include <stdlib.h>

void Initialize(cola * c)
{
	c->frente = NULL;
	c->final = NULL;
	c->num_elem=0;
	return;
}

void Queue(cola * c, elemento e)
{

	nodo * aux;
	aux=malloc(sizeof(nodo));

	if(aux==NULL)
	{
		printf("\nERROR: Desbordamiento de cola");
		exit(1);
	}

	aux->e = e;
	aux->siguiente = NULL;


	if (c->num_elem==0)
	{
		c->frente = aux;
		c->final = aux;
	}
	else
```

```
	{

		c->final->siguiente = aux;
		c->final = aux;
	}

	c->num_elem++;

	return;
}

elemento Dequeue(cola * c)
{
	elemento e;
	nodo *aux;

	if(c->num_elem==0)
	{
		printf("\nERROR: Subdesbordamiento de cola");
		exit(1);
	}
	else
	{
		e = c->frente->e;
		aux=c->frente->siguiente;
		free(c->frente);

		c->num_elem--;
		c->frente = aux;

		if(c->num_elem==0)
		{
			Initialize(c);
		}
	}

	return e;
}

boolean Empty(cola * c)
{
	return (c->num_elem==0) ? TRUE : FALSE;
}

elemento Front(cola * c)
{
	return c->frente->e;
}
```

```
elemento Final(cola * c)
{
	return c->final->e;
}

int Size(cola * c)
{
	return c->num_elem;
}

elemento Element(cola * c, int i)
{
	elemento r;
	nodo *n;
	int j;

	if (i>0&&i<=Size(c))
	{
		n=c->frente;
		for(j=1;j<i;j++)
			n=n->siguiente;
		r=n->e;
	}
	else
	{
		printf("\nERROR: (Element) Se intenta acceder a elemento
inexistente");
		exit(1);
	}
	return r;
}

void Destroy(cola * c)
{
	nodo * aux;

	while(c->frente != NULL)
	{

		aux = c->frente;
		c->frente = c->frente->siguiente;

		free(aux);
	}

	c->num_elem=0;
	c->final = NULL;

	return;
```

```
}
```

Simulación 01 Supermercado.

Uso de funciones y explicación.

```
void hidecursor
```

***Nota:* Aquí se muestra el prototipo de la función, el código completo se encuentra en anexos.**

Esta función es utilizada para poder esconder el cursor en la consola para poder hacer que al momento de ejecutar el programa no ocurra algo que llamamos "parpadeo"

```
void DrawCashier(int x, int y, int IdDrawCashier, char estado, int vel)
```

***Nota:* Aquí se muestra el prototipo de la función, el código completo se encuentra en anexos.**

Esta función es utilizada la dibujar cada una de las cajas, esta función recibe parámetros tales como x, y los cuales son coordenadas para mover el cursor y poder dibujar la caja deseada.

Para poder hacer más atractiva la simulación se utilizaron figuras del código ASCII para poder crear gráficamente la "representación" de una caja.

Esta función además recibe el parámetro *estado e*el cual indica el estado de la caja (atendiendo – vacía – cerrada) y es asi como se imprime con leyenda deseada.

```
oid DrawCashier(int x, int y, int IdDrawCashier, char estado, int vel){
      MoverCursor(x,y);
      printf(" %c%c%c\t",205,205,187);

      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");
            }
```

***Nota:* Aquí se muestra una parte del código, el código completo se encuentra en anexos.**

```
int Cashier(int cant, int largo, char estadoCajeras[], char marketName[], int
llegadaClientes, int tiempoAtencion[], int cantClientesLlegados, int
cantClientesAtendidos)
```

***Nota:* Aquí se muestra el prototipo de la función, el código completo se encuentra en anexos.**

Esta función es utilizada para mostrar en pantalla el nombre de supermercado, el tiempo de llegada de los clientes, el número de clientes que ha llegado y el número de clientes atendidos, esto con la finalidad de que el usuario vea si las condiciones se cumplen para poder cerrar el supermercado a demás a hacer el programa más agradable a la vista.

```
Void Paint(int filaPaint, int Posicion, elemento Cliente, char acaboFila){
```

Esta función es utilizada para dibujar a cada uno de los clientes del supermercado e igualmente se hizo uso del código ASCII para poder representar al cliente como un "■"

```
else{
            MoverCursor(x,y);
            printf("     ");
            MoverCursor(x,y);
            fflush(stdout);
            printf(" %c",254);    }
      return;
}
```

***Nota:* Aquí se muestra una parte del código, el código completo se encuentra en anexos.**

Programa principal (MAIN)

```
int main(void){
      int tiempo, cliente, atendidos, minClientes, fila,columna, alto,
cantCajeras, filaElegida, llegadaClientes;
      elemento aux;
      char marketName[30], fin = 'N';
```

En esta parte es donde se declaran todas las variables locales que son necesarias.

En repetidas ocasiones utilizamos la función `MoverCursor` la cual está dentro del archivo presentacion.h (proporcionado por el profesor Edgardo Adrián Franco Martínez)

Esta función utiliza coordenadas para poder trabajar y es por eso que tiene parámetros similares a este `(40,2);`

Es utilizada para poder tabular las leyendas dentro del programa y hacer que visualmente sea más atractivo para el usuario.

```
MoverCursor(40,2); printf ("*************************************\n");
     MoverCursor(43,4); printf ("Instituto Politecnico Nacional\n");
     MoverCursor(43,5); printf ("Escuela Superior de Computo\n");
```

Posteriormente se pide que se ingresen los siguientes datos.

Nombre del supermercado y el número de cajeras, es aquí donde se procede a hacer una validación del número de cajeras ingresadas para poder si está dentro del rango.

0< cantCajeras<11

```
f(cantCajeras>10 || cantCajeras < 1){
```

Posteriormente se pide que se ingrese el número de clientes mínimos para que el supermercado pueda cerrar (condición importante marcada en el planteamiento del problema)

```
scanf("%d", &minClientes);
```

Declaramos un arreglo para poder guardar los estados de cada cajera y otro arreglo para guardar los tiempos de atención de cada una de las cajeras

```
char estadoCajeras[cantCajeras];
int tiempoAtencion[cantCajeras];
```

Y se declara la cola para almacenar a los clientes.

```
cola filaCajera[cantCajeras];
```

Y a través de un ciclo for se inicializan todas las colas para cada una de las cajeras.

```
for(int i=0; i<cantCajeras; i++)
    Initialize(&filaCajera[i]);
```

Y de la misma forma, a través de un ciclo for guardamos los tiempos de atención de cada una de las cajeras.

Y finalmente se pide que ingresar el tiempo de llegada de los clientes.

A partir de aquí se "inicializa" todo ya que se llama a la función Cashier la cual fue explicada anteriormente y se inicializan las siguientes variables.

```
tiempo = 0; cliente = 0; atendidos = 0;
```

Para decir que aquí es cuando comienza a "trabajar" el supermercado.

Se hace un ciclo que representara el tiempo en que está funcionando el supermercado.

```
while(fin == 'N'){
tiempo = tiempo + 50;
           fin = 'S';
```

Ahora lo que hacemos es iniciar un ciclo desde o hasta el número de cajeras, si la fila no está vacía y la caja no está cerrada se atiende a un cliente y se remueve de la cola (fila), se aumenta el número de clientes atendidos y se modifica el estado de la cajera a atendiendo, en caso contrario se cambia el estado de la cajera a vacía y si ya se cumplieron las condiciones del planteamiento del problema, se cambia el estado de la cajera a cerrada.

```
for(int 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';
}
```

Y nuevamente se llama a la función Cashier para poder imprimir la actualización hecha.

Ahora se redibujan los clientes.

Finalmente se crean números al azar (de 1 a número de cajeras) y se encola un nuevo cliente si es que el supermercado sigue funcionando.

```
                  filaElegida = rand()%cantCajeras;
                  Queue(&filaCajera[filaElegida], aux);
            }
```

***Nota:* Durante la explicación se muestran partes del código, el código completo se encuentra en anexos.**

Simulación 02 Cola de procesos del SO.

Se empezó la codificación para generar la solución al problema especificado, el cual es, simular la cola de procesos de un Sistema Operativo monoproceso, en la cual se maneja el paso de la cola de listos a ejecución y una vez que se termina, a la cola de terminados, mostrando con gráficos de consola todo el proceso de forma agradable al usuario.

La simulación se planteó en las siguientes etapas:

Pedir datos de los procesos

La especificación del problema indicaba que para cada proceso se debía dar su nombre, ID, descripción y tiempo de ejecución.

Esto lo logramos creando una función llamada pedirProceso() la cual pide los datos del proceso, es decir, id, nombre, descripción y tiempo de ejecución, y los va almacenando en colas con la misma etiqueta del dato (hay una cola para nombres, otra para id´s, etc).

Cabe resaltar que desde que se introducen las cadenas (nombre, ID y descripción) se va haciendo un salto o guardando un posible ancho de la columna que tendría el dato en la tabla, esto en el arreglo tam[ ]. Este dato se pasa al final tanto a la función dibujarTabla() como a colaDeProcesos().

Finalmente, llama a las funciones dibujarTabla() (de la librería que se encuentran en anexos tabla.h) y colaDeProcesos().

A continuación se muestra el código:

```
void pedirProceso(){
     int i = 5, j = 0, tam[20], tiempo;
     char id[32], nombre[32], actividad[200], temp[200];
     cola nombres, actividades, ids, tiempos;
     elemento e1;
     Initialize(&nombres);
     Initialize(&actividades);
     Initialize(&ids);
     Initialize(&tiempos);
     do{
          tam[j] = 1; //tam[j] nos indicará el ancho de columna de acuerdo a la
longitud de las cadenas (nombre, id y actividad)
          setbuf(stdin, NULL);
          letreros();
          imprimir("A continuacion introduzca los parametros del proceso que se
le solicitan: ", 4);
          imprimir("Precondiciones (strlen(nombre) y strlen(id)) <= 32,
strlen(descripion) <= 200", 5);
          gotoxy(5,6);
          printf("Introduzca el nombre del proceso: ");
          gets(nombre);
          if(strlen(nombre) >= 10) tam[j] = strlen(nombre)/10 + 1; //el tamaño
del renglón donde se imprimirá es de 10 así que la
          gotoxy(5,7);
     //columa será de ancho igual al número de veces que exceda el 10
          printf("Introduzca el ID del proceso: ");
          gets(id);
```

```
            if(strlen(id) >= 10) tam[j] = strlen(id)/10 + 1;
            gotoxy(5,8);
            printf("Introduzca la descripcion: ");
            gets(actividad);
            i = 9;
            if(strlen(actividad) > 48){//Ciclo para dar saltos de linea durante la
introducción de la cadena en caso de que sea de longitud mayor a
                  i = i + (strlen(actividad) - 48)/80 + 1; //48 (80 - 32 que es la
longitud del letrero "Introduzca ..." + el desplazamiento en x) y que
            }                                                          //así no
se afecte la impresión de los siguientes letreros (que no se encime)
            gotoxy(5,i);
            printf("Introduzca el tiempo de ejecucion (en segs): ");
            scanf("%d", &tiempo);
            if(i > 9){ //Se asume que si i  es mayor a 9, entonces actividad
superaría su espacio de impresión en el renglón
                  tam[j] = strlen(actividad)/50 + 1;
            }
            strcpy(e1.array, nombre); //Se copia el nombre al elemento e1 en su
parte array[]
            Queue(&nombres, e1);    //Se encola el nombre en la cola nombres
            strcpy(e1.array, actividad);
            Queue(&actividades, e1);
            strcpy(e1.array, id);
            Queue(&ids, e1);
            e1.n = tiempo;
            Queue(&tiempos, e1);
            if(otroProceso() == 2) break; //Se pregunta si se desea introducir
otro proceso, en caso de que no, se sale del ciclo
            cleanScreen();
            j++;
            if(j == 19) break; //Si se ha superado el número máximo de datos a
introducir
      }while(1);
      dibujarTabla(tam, &nombres, &ids, &actividades, &tiempos, 18, "espera",
Size(&tiempos)); //Dibuja la tabla con los datos introducidos
      getchar();
      printf("\nPresione cualquier tecla para proceder a la inicializacion de la
Cola de proceso");
      getchar();
      cleanScreen();
      delay_ms(500);
      colaDeProcesos(tam, &nombres, &ids, &actividades, &tiempos); //Llama a la
función que simulará la cola de procesos
      return;
}
```

Dibujar la tabla con los procesos introducidos

El siguiente paso era mostrar los procesos acomodados en la tabla, respetando la correcta visualización, de modo que el usuario pueda corroborar sus datos antes de que se ejecute la simulación de la cola de procesos, esto se hace mediante un llamado a la función dibujarTabla(), la cual se encuentra bien documentada en el anexo.

Esta función se llama inmediatamente después de que se introducen los procesos, pasándole el tamaño a salto de la columna (un array), la coordenada Y donde se empezará a imprimir, en este caso la 18, las colas por referencia y el número de elementos a imprimir, es decir el número de filas, que se especificó como Size(&tiempos) pero pudo haber sido la operación Size() de cualquier cola, pues todas tienen la misma cantidad de elementos. Nota: la razón de que se crearán tantas colas fue por comodidad, pues nos resultó más sencillo comprender la manipulación de esas colas por separado a manipular un array de colas o los elementos del struct elemento.

```
dibujarTabla(tam, &nombres, &ids, &actividades, &tiempos, 18, "espera", Size(&tiempos));
```

Cola de procesos

El último paso era simular la cola de procesos, por lo que se creó la función colaDeProcesos(), en la cual se declaran nuevas colas, de finalización, con la etiqueta análoga, es decir, hay un tiempoFinalizado, nombreFinalizado, etcétera.

Y se propone un bucle, que mientras la cola tiempo tenga elementos (aunque podría preguntarse por las otras colas- nombre, id, etc – pues tienen el mismo número de elementos) se realiza el proceso de paso de la cola de espera a ejecución (para esto no se crea una cola pues simplemente va a mantener un elemento a la vez), ahí se mantienen un segundo, decrementando el contador en 1 unidad al final y, en caso de que el tiempo restante sea 0, se pasa a la cola de terminados, de lo contrario se encola hasta el final de la cola de espera.

Se va mostrando de manera animada la cola de ejecución como una tabla de una fila y las columnas iniciales con los campos de cada proceso. Debajo de esta se muestra el siguiente proceso a ser ejecutado, junto con su tiempo restante, así como el último proceso ejecutado e, igualmente, su tiempo restante.

Y hasta el final se muestra la cola de finalizados, que empieza vacía y en ella se van vaciando los procesos conforme van terminando, quedando llena al final.

Aunque ya se muestra la cola de finalizados, se indica que se terminó la simulación, y que se procederá a mostrar la cola de finalizados, pero de manera que solo salga esta última, junto con los tiempos acumulados de cada proceso (es decir, el tiempo total que transcurrió desde el

inicio de la cola de procesos hasta que se encolaron en terminados, y que es la suma del tiempo de ejecución más el tiempo de espera) y el tiempo total de ejecución de los procesos.

Finalmente regresa el control al main, desde donde se pregunta si se desea introducir otro proceso. En caso de que si, borra la pantalla y se inicia nuevamente el programa, en caso contrario termina el programa.

```
void colaDeProcesos(int anchoColumnas[], cola *nombre, cola *id, cola *actividad,
cola *tiempo){
      char nom[32], ids[32];
      int t;
      cola nombreFinalizado, idFinalizado, actividadFinalizado, tiempoFinalizado;
      elemento e1;
      int i, j, desplazamiento = 0, tiempoTotal = 0, tiempoEjecucion = 0;
      Initialize(&nombreFinalizado);
      Initialize(&idFinalizado);
      Initialize(&actividadFinalizado);
      Initialize(&tiempoFinalizado);
      i = 0;
      while(Empty(tiempo) == FALSE){ //Mientras la cola tiempo (aunque podría ser
cualquiera de las que se recibieron, ya que tienen el mismo tamaño)
            letreros();                              //no este vacía
            dibujarTabla(anchoColumnas, nombre, id, actividad, tiempo, 10,
"ejecucion", 1);//Dibuja la tabla que contiene al proceso en ejecución
            while(Empty(tiempo) == FALSE){ //Algoritmo de encolado y desencolado
para paso de cola de espera a ejecución cuando transcurre un quantum
                  cleanScreen();
                  letreros();
                  dibujarTabla(anchoColumnas, nombre, id, actividad, tiempo, 10,
"ejecucion", 1);//Dibuja la tabla que contiene al proceso en ejecución
                  dibujarTabla(anchoColumnas, &nombreFinalizado, &idFinalizado,
&actividadFinalizado, &tiempoFinalizado, 28, "finalizados",
Size(&nombreFinalizado));
                  if(i != 0){
                        e1 = Element(nombre, Size(tiempo));
                        strcpy(nom, e1.array);
                        e1 = Element(id, Size(tiempo));
                        strcpy(ids, e1.array);
                        e1 = Element(tiempo, Size(tiempo));
                        t = e1.n;
                        imprimir("Ultimo proceso Ejecutado: ", 14);
                        gotoxy(0,15);
                        printf("Nombre = %s \t ID = %s \tTiempo para que finalice
= %d", nom, ids, t);
                        imprimir("Proximo proceso a ejecutarse: ", 18);
                        if(Size(nombre) > 1){ //En caso de que haya más de un
elemento en la cola
```

```
                        e1 = Element(nombre, 2);//Se muestra el segundo visto
desde el frente
                        strcpy(nom, e1.array);//porque es el próximo a ser
ejecutado
                        e1 = Element(id, 2);
                        strcpy(ids, e1.array);
                        e1 = Element(tiempo, 2);
                        t = e1.n;
                    }
                    else{ //Si solo hay un elemento en la cola se obtiene el
frente, pues
                        e1 = Front(nombre); //ese va a ser el siguiente, y a
su vez fue
                        strcpy(nom, e1.array); //el anterior, esto pasa cuando
ya solo
                        e1 = Front(id); //queda un proceso en ejecución
                        strcpy(ids, e1.array);
                        e1 = Front(tiempo);
                        t = e1.n;
                    }
                    gotoxy(0,19);
                    printf("Nombre = %s \t ID = %s \tTiempo para que finalice
= %d", nom, ids, t);
                }
                e1 = Dequeue(tiempo);
                gotoxy(72,10);
                printf("%ds", e1.n);
                delay_ms(1000);
                if(e1.n > 0){ //Si el tiempo de ese proceso aun no acaba (t == 0
es el criterio de finalización)
                    e1.n--;
                    Queue(tiempo, e1); //Encola el nuevo tiempo
                    Queue(nombre, Dequeue(nombre)); //Encola (lleva al final) el
nombre que se encontraba al principio
                    Queue(id, Dequeue(id));
                    Queue(actividad, Dequeue(actividad));
                }
                else{
                    e1.n = tiempoEjecucion;
                    Queue(&tiempoFinalizado, e1); //Se encola el tiempoEjecucion
de ese proceso en tiempoFinalizado
                    Queue(&nombreFinalizado, Dequeue(nombre)); //Se desencola el
nombre de la cola de ejecución y se encola en finalizado
                    Queue(&idFinalizado, Dequeue(id));
                    Queue(&actividadFinalizado, Dequeue(actividad));
                    break;
                }
                tiempoEjecucion++;
                i++;
```

```
            }
            dibujarTabla(anchoColumnas, &nombreFinalizado, &idFinalizado,
&actividadFinalizado, &tiempoFinalizado, 28, "finalizados",
Size(&nombreFinalizado));
            if(Empty(nombre) == TRUE){ //Una vez que se vacía la cola de ejecución
                  printf("\n\tEl SO ha ejecutado todos los procesos en la cola.
\n\tPresione cualquier tecla para continuar ...");
                  getchar();
            }
      }
      cleanScreen();
      letreros();
      dibujarTabla(anchoColumnas, &nombreFinalizado, &idFinalizado,
&actividadFinalizado, &tiempoFinalizado, 10, "finalizados",
Size(&tiempoFinalizado));
      printf("\n\t\tEl tiempo total de ejecucion fue de %ds \n", tiempoEjecucion);
//Se indica el tiempo (tiempo de proceso 1 + proceso 2 + ... + proceso n)
      Destroy(&nombreFinalizado); //Destruye las colas pensando en un posible
nuevo proceso
      Destroy(&idFinalizado);
      Destroy(&actividadFinalizado);
      Destroy(&tiempoFinalizado);
      Destroy(nombre);
      Destroy(id);
      Destroy(actividad);
      Destroy(tiempo);
      return;
}
```

Simulacion 03.

Al comienzo de la simulación se declaran las variables a ocupar y se utiliza una función que solicita el número de cajas que atenderán el banco, posteriormente se solicita el tiempo de llegada de los clientes, usuarios y preferentes, para después solicitar el tiempo de llegada de cada tipo de visitante y el tiempo de atención de cada cajero.

Cuando comienza el ciclo del banco la condición de tener funcionando el banco es que, si las colas no están vacías o no es hora de cerrar, el banco sigue trabajando. Cada ciclo el programa revisa si le toca atender a algún cajero, gracias a un for que va desde o hasta el último cajero, si le toca atender al cajero, siempre intentara atender a preferentes verificando que haya preferentes formados y el cajero este vacío y que no hayan pasado más de 3 preferentes seguidos, si no intentara atender a un cliente si es que hay clientes formados el cajero esta vacío y no han pasado más de dos clientes seguidos por último si no se cumplió alguna condición anterior intentara atender a un usuario si es que hay en cola y ya han pasado 5 visitantes entre preferentes y clientes,

sin embargo decidimos agregar dos condiciones más, para la atención de clientes, si hay clientes formados y aún no han llegado tanto usuarios como preferentes, así evitando que con solo la primera condición se quedaran atascados y para atender a algún usuario se consideró que si hay usuarios formados y el cajero esta vacío entonces que pase directamente un usuario ya que como son if en cascada seria el último intento de atención indicando que tanto clientes como preferentes no han llegado al banco. Por último, también consideramos si los preferentes fueran los únicos que llegasen entonces se atenderían directamente, respetando las políticas del banco.

Para encolar a los visitantes en su respectiva cola, consideramos que, si el tiempo transcurrido modulo tiempo de cada visitante un elemento en la parte tipoC, tipoP y tipoU se le asignara el valor de un auxiliar que tiene el número de cliente a encolar, y al elemento en su parte n se le asignara un número de identificador de visitante, con los tres tipos podemos distinguir al U1 del P1.

Al final para la impresión de las colas tomamos en cuenta un tiempo de impresión y una condición, si el tamaño de cola es superior a los 20 elementos, entonces comienza un ciclo de 1 al 20 para imprimir los primeros 20 en la cola y otro ciclo que empieza de 1 al tamaño de cola imprimiendo solo un entero, el tamaño de cola menos 20 ya que los primeros 20 se imprimieron en el anterior for, en este segundo caso solo se imprimirá "Y (tamaño de cola – 20) mas", si el tamaño de cola es inferior a 20 elementos solo se inicia un for desde 1 al tamaño de cola imprimiendo el elemento en la posición del ciclo. Esta implementación se realizó para la impresión de las 3 colas.

Cada cierto tiempo la pantalla se va limpiando para evitar que queden residuos de impresiones, eso se deben los pantallazos cada 5 segundos, este valor se puede modificar con limpiar

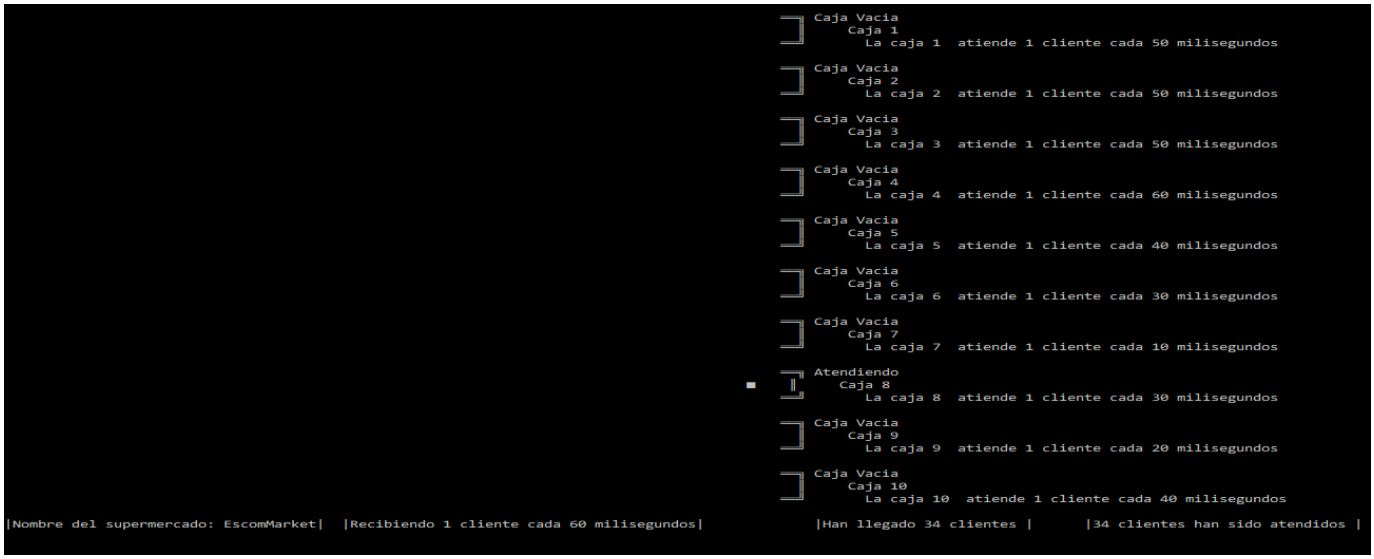
Cuando haya pasado el tiempo de cerrar, los clientes ya no llegarán a formarse en la fila, sin embargo, los clientes ya formados se atenderán intentando respetar las políticas de atención. Al cerrar el banco y finalizar la simulación se destruyen las colas y se imprime en pantalla, el banco ha cerrado, vuelve otro dia.

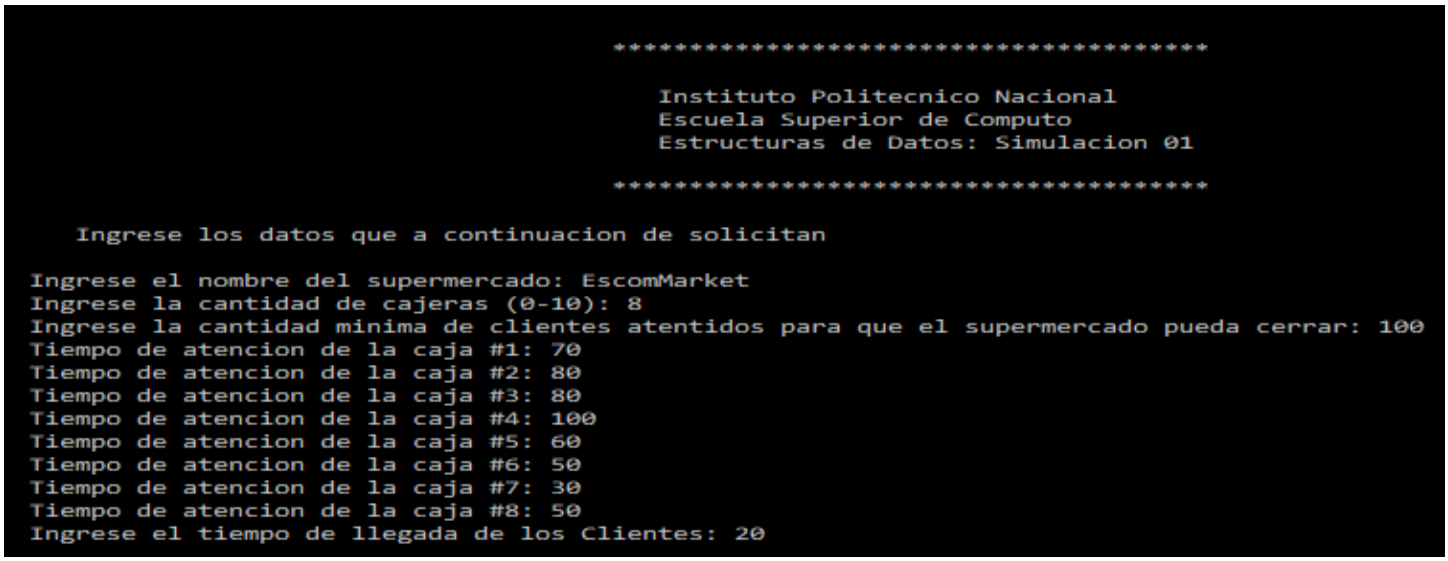
Funcionamiento

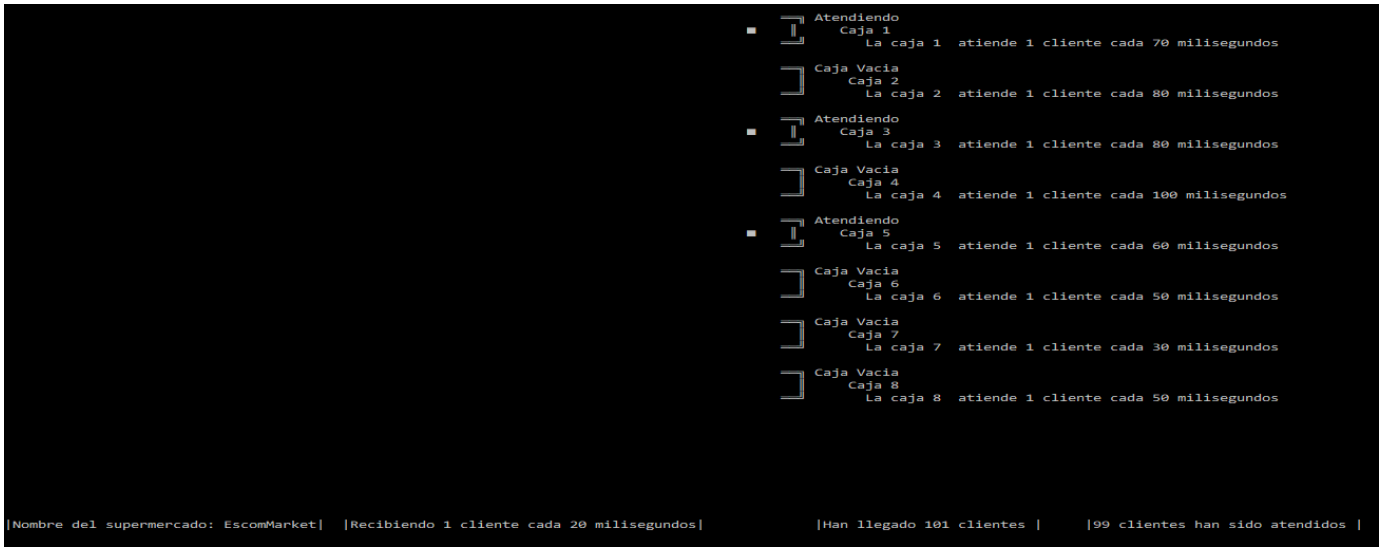
Simulación 01.

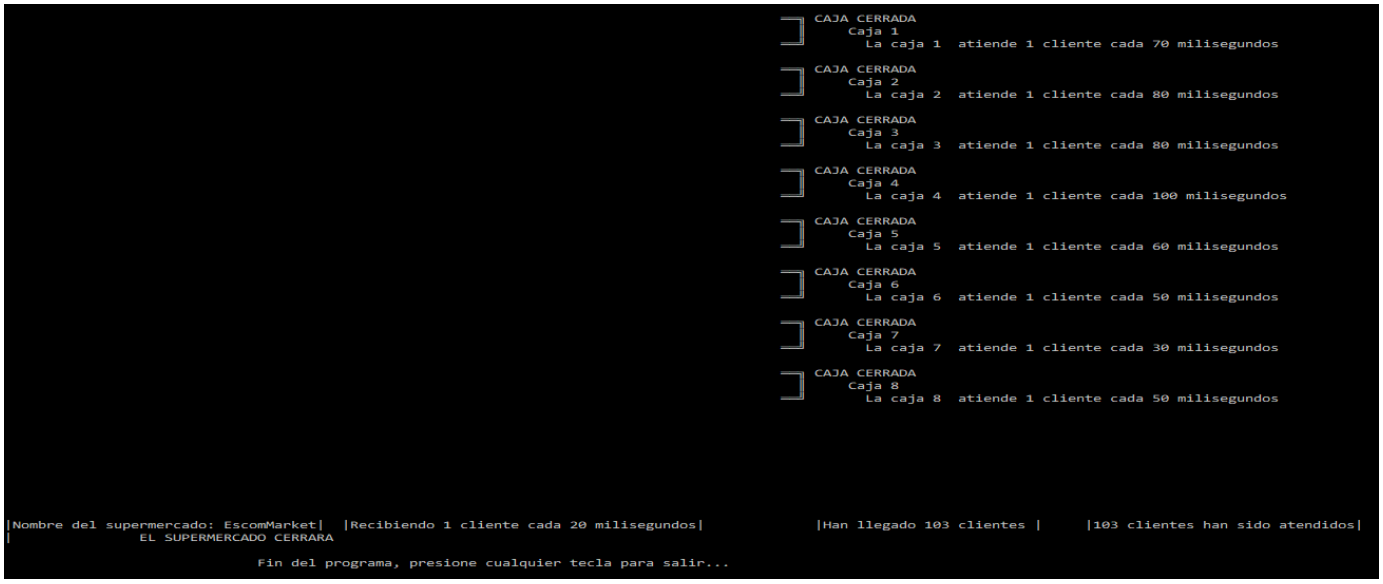
```
                                    ****************************************

                                       Instituto Politecnico Nacional
                                       Escuela Superior de Computo
                                       Estructuras de Datos: Simulacion 01

                                    ****************************************

    Ingrese los datos que a continuacion de solicitan

Ingrese el nombre del supermercado: EscomMarket
Ingrese la cantidad de cajeras (0-10): 10
Ingrese la cantidad minima de clientes atentidos para que el supermercado pueda cerrar: 50
Tiempo de atencion de la caja #1: 50
Tiempo de atencion de la caja #2: 50
Tiempo de atencion de la caja #3: 50
Tiempo de atencion de la caja #4: 60
Tiempo de atencion de la caja #5: 40
Tiempo de atencion de la caja #6: 30
Tiempo de atencion de la caja #7: 10
Tiempo de atencion de la caja #8: 30
Tiempo de atencion de la caja #9: 20
Tiempo de atencion de la caja #10: 40
Ingrese el tiempo de llegada de los Clientes: 60
```

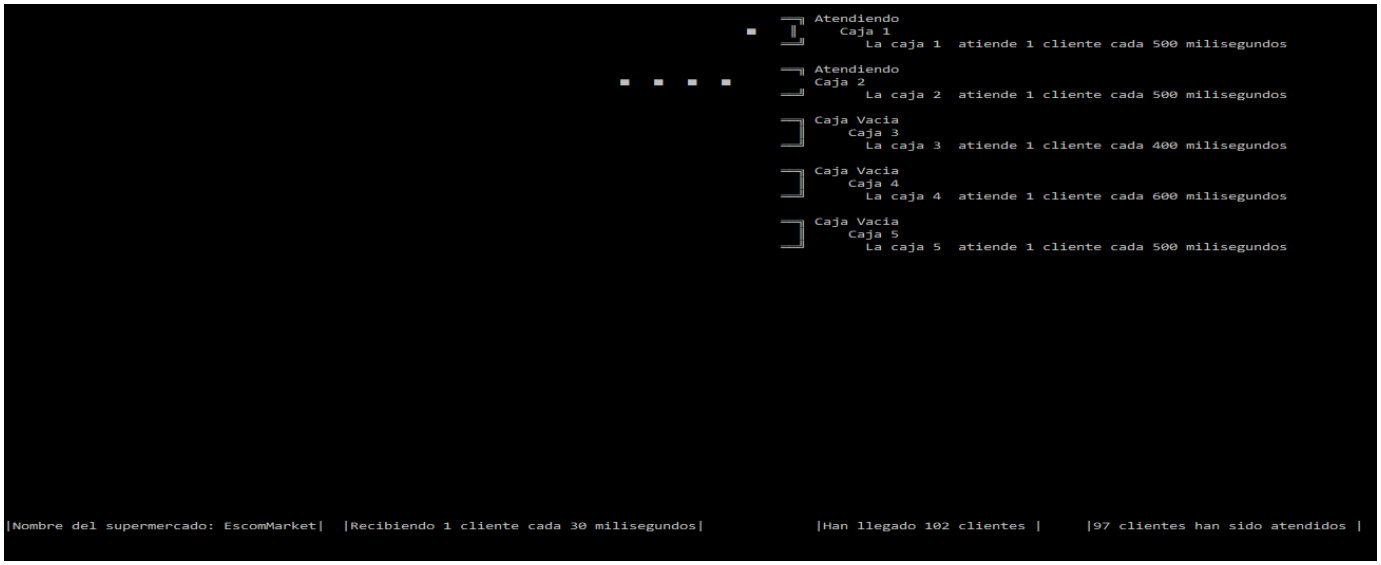
```
                                                    Caja Vacia
                                                        Caja 1
                                                          La caja 1  atiende 1 cliente cada 50 milisegundos

                                                    Caja Vacia
                                                        Caja 2
                                                          La caja 2  atiende 1 cliente cada 50 milisegundos

                                                    Caja Vacia
                                                        Caja 3
                                                          La caja 3  atiende 1 cliente cada 50 milisegundos

                                                    Caja Vacia
                                                        Caja 4
                                                          La caja 4  atiende 1 cliente cada 60 milisegundos

                                                    Caja Vacia
                                                        Caja 5
                                                          La caja 5  atiende 1 cliente cada 40 milisegundos

                                                    Caja Vacia
                                                        Caja 6
                                                          La caja 6  atiende 1 cliente cada 30 milisegundos

                                                    Caja Vacia
                                                        Caja 7
                                                          La caja 7  atiende 1 cliente cada 10 milisegundos

                                                    Caja Vacia
                                                        Caja 8
                                                          La caja 8  atiende 1 cliente cada 30 milisegundos

                                                    Caja Vacia
                                                        Caja 9
                                                          La caja 9  atiende 1 cliente cada 20 milisegundos

                                                    Caja Vacia
                                                        Caja 10
                                                          La caja 10  atiende 1 cliente cada 40 milisegundos
|Nombre del supermercado: EscomMarket|  |Recibiendo 1 cliente cada 60 milisegundos|          |Han llegado 15 clientes |      |15 clientes han sido atendidos |
```


Caja Vacia
Caja 1
La caja 1 atiende 1 cliente cada 50 milisegundos
Caja Vacia
Caja 2
La caja 2 atiende 1 cliente cada 50 milisegundos
Caja Vacia
Caja 3
La caja 3 atiende 1 cliente cada 50 milisegundos
Caja Vacia
Caja 4
La caja 4 atiende 1 cliente cada 60 milisegundos
Caja Vacia
Caja 5
La caja 5 atiende 1 cliente cada 40 milisegundos
Caja Vacia
Caja 6
La caja 6 atiende 1 cliente cada 30 milisegundos
Caja Vacia
Caja 7
La caja 7 atiende 1 cliente cada 10 milisegundos
Atendiendo
Caja 8
La caja 8 atiende 1 cliente cada 30 milisegundos
Caja Vacia
Caja 9
La caja 9 atiende 1 cliente cada 20 milisegundos
Caja Vacia
Caja 10
La caja 10 atiende 1 cliente cada 40 milisegundos
|Nombre del supermercado: EscomMarket| |Recibiendo 1 cliente cada 60 milisegundos| |Han llegado 34 clientes | |34 clientes han sido atendidos |


****
Instituto Politecnico Nacional
Escuela Superior de Computo
Estructuras de Datos: Simulacion 01
****
Ingrese los datos que a continuacion de solicitan
Ingrese el nombre del supermercado: EscomMarket
Ingrese la cantidad de cajeras (0-10): 8
Ingrese la cantidad minima de clientes atentidos para que el supermercado pueda cerrar: 100
Tiempo de atencion de la caja #1: 70
Tiempo de atencion de la caja #2: 80
Tiempo de atencion de la caja #3: 80
Tiempo de atencion de la caja #4: 100
Tiempo de atencion de la caja #5: 60
Tiempo de atencion de la caja #6: 50
Tiempo de atencion de la caja #7: 30
Tiempo de atencion de la caja #8: 50
Ingrese el tiempo de llegada de los Clientes: 20


Atendiendo
Caja 1
La caja 1 atiende 1 cliente cada 70 milisegundos
Caja Vacia
Caja 2
La caja 2 atiende 1 cliente cada 80 milisegundos
Atendiendo
Caja 3
La caja 3 atiende 1 cliente cada 80 milisegundos
Caja Vacia
Caja 4
La caja 4 atiende 1 cliente cada 100 milisegundos
Atendiendo
Caja 5
La caja 5 atiende 1 cliente cada 60 milisegundos
Caja Vacia
Caja 6
La caja 6 atiende 1 cliente cada 50 milisegundos
Caja Vacia
Caja 7
La caja 7 atiende 1 cliente cada 30 milisegundos
Caja Vacia
Caja 8
La caja 8 atiende 1 cliente cada 50 milisegundos
|Nombre del supermercado: EscomMarket| |Recibiendo 1 cliente cada 20 milisegundos| |Han llegado 101 clientes | |99 clientes han sido atendidos |





Simulación 02.

Prueba:

Proceso 1:

Nombre	ID	Descripción	Tiempo
main.c	C001	Programa en C	12s

Proceso 2:

Nombre	ID	Descripción	Tiempo
main.cpp	CPP01	Programa en C++	21s

Tiempo final esperado: 33s

```
                        Instituto Politecnico Nacional
                                    ESCOM
                        Cola de procesos de Sistema Operativo

   A continuacion introduzca los parametros del proceso que se le solicitan:
 Precondiciones (strlen(nombre) y strlen(id)) <= 32, strlen(descripion) <= 200
     Introduzca el nombre del proceso: main.c
     Introduzca el ID del proceso: C001
     Introduzca la descripcion: Programa en C
     Introduzca el tiempo de ejecucion (en segs): 12

Desea introducir otro proceso(s|S,n|N)?:
```

```
                        Instituto Politecnico Nacional
                                    ESCOM
                        Cola de procesos de Sistema Operativo

   A continuacion introduzca los parametros del proceso que se le solicitan:
 Precondiciones (strlen(nombre) y strlen(id)) <= 32, strlen(descripion) <= 200
     Introduzca el nombre del proceso: main.cpp
     Introduzca el ID del proceso: CPP01
     Introduzca la descripcion: Programa en C++
     Introduzca el tiempo de ejecucion (en segs): 21

Desea introducir otro proceso(s|S,n|N)?: n


                                  Cola de espera
┌-------------------------------------------------------------------------------------┐
|  Nombre  |   ID    |                  Descripcion                      | Tiempo     |
|-------------------------------------------------------------------------------------|
|main.c    |C001     |Programa en C                                      |12s         |
|-------------------------------------------------------------------------------------|
|main.cpp  |CPP01    |Programa en C++                                    |21s         |
└-------------------------------------------------------------------------------------┘
Presione cualquier tecla para proceder a la inicializacion de la Cola de proceso
```

```


                                Cola de ejecucion
┌-------------------------------------------------------------------------------------┐
|  Nombre  |   ID    |                  Descripcion                      | Tiempo     |
|-------------------------------------------------------------------------------------|
|main.cpp  |CPP01    |Programa en C++                                    |20s         |
└-------------------------------------------------------------------------------------┘


                           Ultimo proceso Ejecutado:
Nombre = main.c          ID = C001      Tiempo para que finalice = 10


                          Proximo proceso a ejecutarse:
Nombre = main.c          ID = C001      Tiempo para que finalice = 10



                                Cola de finalizados
┌-------------------------------------------------------------------------------------┐
|  Nombre  |   ID    |                  Descripcion                      | Tiempo     |
└-------------------------------------------------------------------------------------┘
```

```
                                Cola de ejecucion
┌──────────┬──────────┬──────────────────────────────────────────────────────────┬──────────┐
│  Nombre  │    ID    │                        Descripcion                       │  Tiempo  │
├──────────┼──────────┼──────────────────────────────────────────────────────────┼──────────┤
│main.cpp  │CPP01     │Programa en C++                                           │0s        │
└──────────┴──────────┴──────────────────────────────────────────────────────────┴──────────┘

                            Ultimo proceso Ejecutado:
Nombre = main.cpp          ID = CPP01      Tiempo para que finalice = 0

                           Proximo proceso a ejecutarse:
Nombre = main.cpp          ID = CPP01      Tiempo para que finalice = 0


                                Cola de finalizados
┌──────────┬──────────┬──────────────────────────────────────────────────────────┬──────────┐
│  Nombre  │    ID    │                        Descripcion                       │  Tiempo  │
├──────────┼──────────┼──────────────────────────────────────────────────────────┼──────────┤
│main.c    │C001      │Programa en C                                             │24s       │
├──────────┼──────────┼──────────────────────────────────────────────────────────┼──────────┤
│main.cpp  │CPP01     │Programa en C++                                           │33s       │
└──────────┴──────────┴──────────────────────────────────────────────────────────┴──────────┘
        El SO ha ejecutado todos los procesos en la cola.
        Presione cualquier tecla para continuar ...
```

```
                          Instituto Politecnico Nacional
                                      ESCOM
                          Cola de procesos de Sistema Operativo


                                Cola de finalizados
┌──────────┬──────────┬──────────────────────────────────────────────────────────┬──────────┐
│  Nombre  │    ID    │                        Descripcion                       │  Tiempo  │
├──────────┼──────────┼──────────────────────────────────────────────────────────┼──────────┤
│main.c    │C001      │Programa en C                                             │24s       │
├──────────┼──────────┼──────────────────────────────────────────────────────────┼──────────┤
│main.cpp  │CPP01     │Programa en C++                                           │33s       │
└──────────┴──────────┴──────────────────────────────────────────────────────────┴──────────┘
                        El tiempo total de ejecucion fue de 33s

Desea introducir otro proceso(s|S,n|N)?:
```

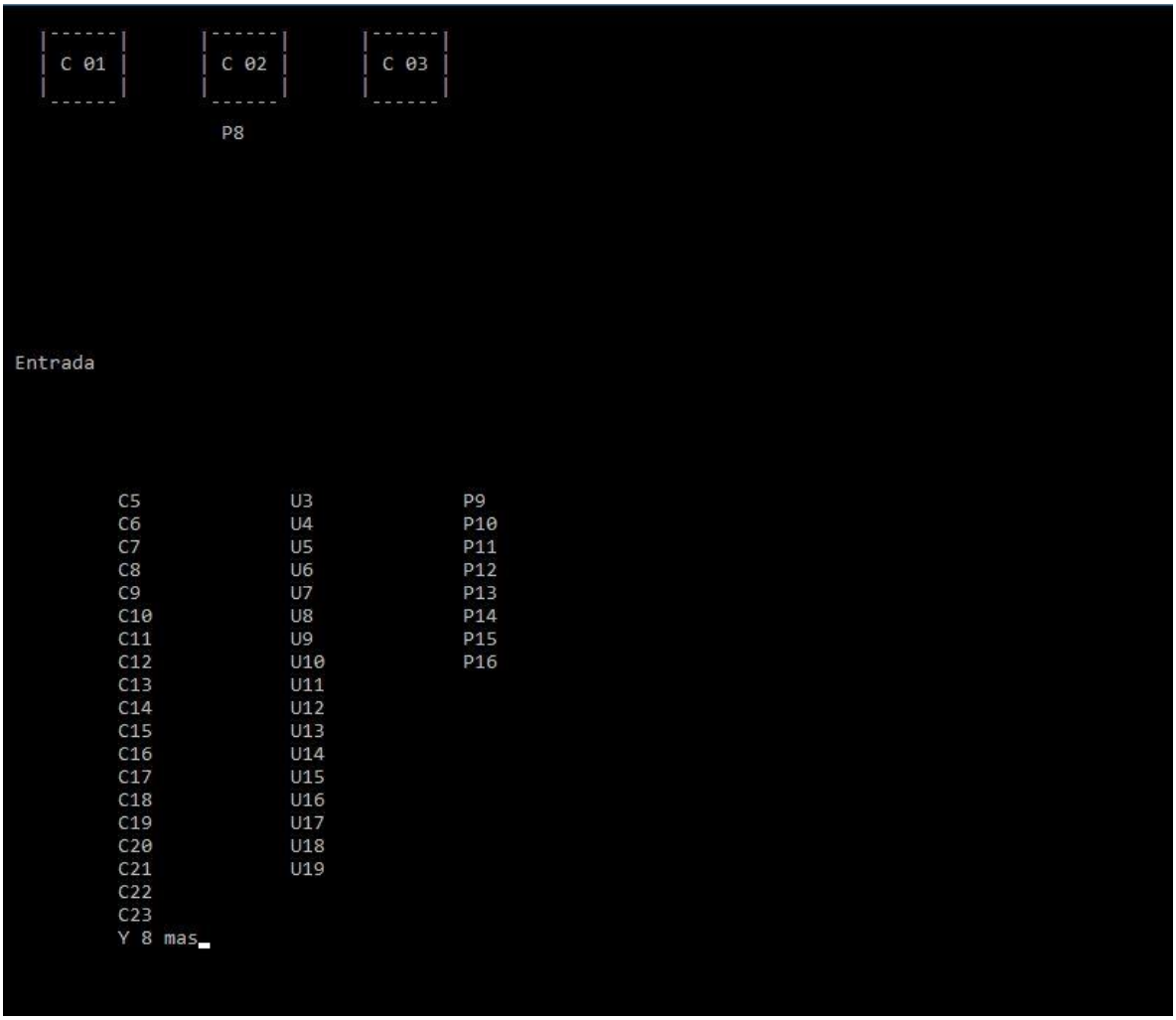
Simulacion 03.

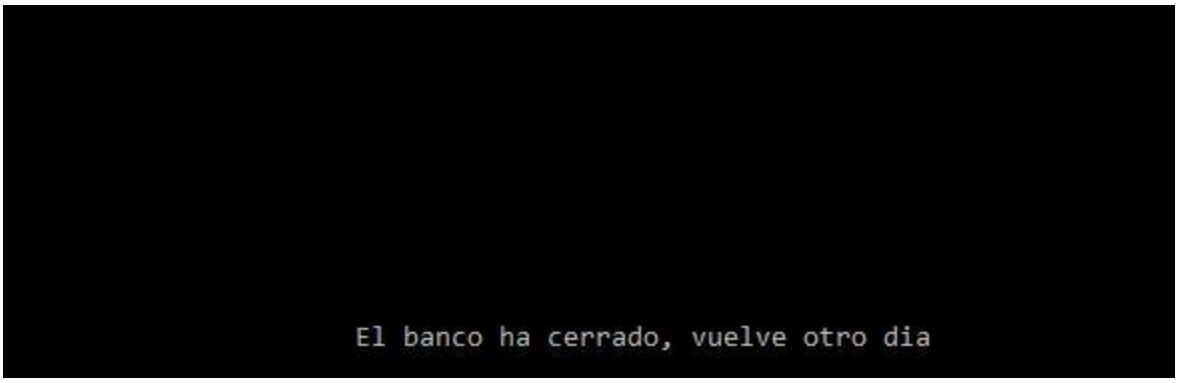
```
                          Instituto Politecnico Nacional
                                      ESCOM
                                Simulacion de banco

                                Alanis Ramirez Damian
                              Mendieta Torres Alfonso Ulises
                              Oledo Eneiquez Gilberto Irving

    Introduzca numero de cajeros:
```

```
                        Instituto Politecnico Nacional
                                    ESCOM
                               Simulacion de banco

                                Alanis Ramirez Damian
                              Mendieta Torres Alfonso Ulises
                              Oledo Eneiquez Gilberto Irving

    Introduzca numero de cajeros: 10

        Usted selecciono 10 cajeros, presione cualquier tecla para continuar ...
```

```
Ingrese el tiempo de llegada de clientes (ms): 120
Ingrese el tiempo de llegada de usuarios (ms): 260
Ingrese el tiempo de llegada de preferentes (ms): 380
Ingrese el tiempo de atencion del cajero 1 220
Ingrese el tiempo de atencion del cajero 2 360
Ingrese el tiempo de atencion del cajero 3 140
Ingrese el tiempo de atencion del cajero 4 260
Ingrese el tiempo de atencion del cajero 5 230
Ingrese el tiempo de atencion del cajero 6 820
Ingrese el tiempo de atencion del cajero 7 990
Ingrese el tiempo de atencion del cajero 8 860
Ingrese el tiempo de atencion del cajero 9 480
Ingrese el tiempo de atencion del cajero 10 220
```

```
 ------   ------   ------   ------   ------   ------   ------   ------   ------   ------
| C 01 | | C 02 | | C 03 | | C 04 | | C 05 | | C 06 | | C 07 | | C 08 | | C 09 | | C 10 |
 ------   ------   ------   ------   ------   ------   ------   ------   ------   ------
  P6                C13      C12                                          U11



Entrada


      C13         U12         P6
```





Errores detectados

Simulacion 01.

Un error detectado en la simulación 01 fue el tiempo en milisegundos, tuvimos problemas ya que al poner 10 en tiempo de llegada de los clientes este 10 no era tomado como milisegundos, y el programa tardaba mucho en hacer llegar un cliente lo cual estaba mal, finalmente pudimos resolver el problema, el cual estaba en un ciclo while y simplemente aumentamos el tiempo en tiempo + 50.

```
while(fin == 'N'){
            tiempo = tiempo + 50;
            fin = 'S';
```

Y el problema quedo corregido pero sin duda fue un gran problema analizar porque el tiempo no era correcto al momento de iniciar el programa.

Otro error detectado y el cual no pudo ser corregido fue que al estar formados muchos clientes en una sola fila (el tiempo de atención era lento y el tiempo de llegada era rápido) si las figuras que representaban los clientes llegaban el borde del marco de la consola comenzaban a empalmarse con otras figuras y no se podía apreciar cómo eran atendidos los clientes.

Simulación 02.

Hasta ahora no se han detectado errores, salvo una observación, y es que pareciera que cuando el tiempo llega a 0 segundos todavía lo ejecuta 1 segundo más, sin embargo eso parece ser una transición de ejecución a la cola de terminados, pues al final el tiempo total es congruente con lo que se esperaba al principio.

Simulacion 03.

La limpieza de pantalla es esencial cuando una cola avanza más rápido que otro, a veces cuando un preferente P220 se imprime en caja si la caja atiende un usuario U50 se sobrescribirá y se imprimirá U500, intentamos respetar al máximo las políticas de atención de clientes sin embargo hay veces que un cliente no es atendido siempre. La atención de siempre usuarios por una caja aún no se completa correctamente.

Posibles Mejoras

Simulación 01.

La primera posible mejora es sin duda resolver el problema de empalme explicado anteriormente, otra posible mejora sin duda seria en la parte del código, ya que pensamos como equipo que quizá la implementación quedo un poco confusa en cuanto el código y sin duda es algo que nos gustaría mejorar para que el código fuera mucho más entendible.

Simulación 02.

Podría reducirse el número de colas que se manipulan si simplemente se añaden variables dentro de la estructura elemento, o si se crea un arreglo de colas. Esto llega a hacer que las funciones se vean muy aparatosas, al recibir un gran número de argumentos, por lo que esta podría ser una mejora que convendría para simplificar el código. Aunque con la desventaja de

que se debe manipular el struct elemento de cada archivo .h de la implementación estática y dinámica.

Simulacion 03.

Para el mejor seguimiento de las políticas del banco se puede tomar las cajeras como colas, y así poder saber si está vacía y asignarle un número de conteo en el elemento y mencione exactamente cuántos clientes, preferentes o usuarios se van atendiendo y llevar el conteo más exacto. Para tomar la condición de 5 (usuarios y preferentes) y 1(Clientes).

Anexos

Aquí se muestran los códigos completos con la documentación requerida.

TADColaDin.c

```c
/*
IMPLEMENTACION DE LA LIBRERIA DEL TAD COLA ESTATICA (TADColaEst.h)
AUTOR: Edgardo Adrián Franco Martínez (C) Febrero 2017
VERSIÓN: 1.6

DESCRIPCIÓN: TAD cola o Queue.
Estructura de datos en la que se cumple:
Los elementos se insertan en un extremo (el posterior) y
la supresiones tienen lugar en el otro extremo (frente).

OBSERVACIONES: Hablamos de una Estructura de datos dinámica
cuando se le asigna memoria a medida que es necesitada,
durante la ejecución del programa.

COMPILACIÓN PARA GENERAR EL CÓDIGO OBJETO: gcc -c TADColaDin.c
*/

//LIBRERAS
#include "TADColaDin.h"
#include <stdio.h>
#include <stdlib.h>

/*
void Initialize(cola *c);
Descripción: Inicializar cola (Iniciar una cola para su uso)
Recibe: cola *c (Referencia a la cola "c" a operar)
Devuelve:
Observaciones: El usuario a creado una cola y c tiene la referencia a ella,
si esto no ha pasado se ocasionara un error.
*/
void Initialize(cola * c)
{
      c->frente = NULL;
      c->final = NULL;
      c->num_elem=0;
      return;
}

/*
void Queue(cola * c, elemento e);
Descripción: Recibe una cola y agrega un elemento al final de ella.
```

```
Recibe: cola *c (Referencia a la cola "c" a operar) elemento e (Elemento a
encolar)
Devuelve:
Observaciones: El usuario a creado una cola y ha sido inicializada y c tiene la
referencia a ella,
si esto no ha pasado se ocasionara un error.
*/
void Queue(cola * c, elemento e)
{
	//Apuntador a elemento
	nodo * aux;

	//Crear un bloque de memoria para un elemento y mantener su referencia en
aux
	aux=malloc(sizeof(nodo));

	//Si malloc no pudo devolver un bloque de memoria indicar el error
	if(aux==NULL)
	{
		printf("\nERROR: Desbordamiento de cola");
		exit(1);
	}

	//Introducir el elemento al bloque referenciado por aux
	aux->e = e;

	//Colocar el NULL a el apuntador siguiente del nuevo elemento
	aux->siguiente = NULL;

	//Si la cola esta vacia, los apuntadores de la cola apuntarán al nuevo
elemento
	if (c->num_elem==0)
	{
		c->frente = aux;
		c->final = aux;
	}
	//Si la cola ya tiene elementos
	else
	{
		//El elemento del final apuntará al nuevo elemento
		c->final->siguiente = aux;
		//El final de la cola apuntará al nuevo elemento
		c->final = aux;
	}
	//Incrementar el número de elementos en la cola
	c->num_elem++;

	return;
}
```

```
/*
elemento Dequeue(cola * c);
Descripción: Recibe una cola y devuelve el elemento que se encuentra al
frente de esta, quitándolo de la cola.
Recibe: cola *c (Referencia a la cola "c" a operar)
Devuelve: elemento (Elemento desencolado)
Observaciones: El usuario a creado una cola y la cola tiene elementos, si no
se genera un error y se termina el programa.
*/
elemento Dequeue(cola * c)
{
	elemento e;	//Elemento a retornar
	nodo *aux; //Apuntador auxiliar

	//Si la cola esta vacia (Subdesbordamiento de cola)
	if(c->num_elem==0)
	{
		printf("\nERROR: Subdesbordamiento de cola");
		exit(1);
	}
	//Si la cola tiene elementos
	else
	{
		//Almacenar el elemento a retornar
		e = c->frente->e;

		//Guardar la dirección del siguiente nodo
		aux=c->frente->siguiente;

		//Destruir el bloque de memoria del elemento al frente
		free(c->frente);

		//Decrementar el número de eleme en la cola
		c->num_elem--;

		//El nuevo frente de la cola es aux (Siguiente del frente)
		c->frente = aux;

		//Si la cola ha quedado vacia se inicializa (c->frente=NULL, c-
>final=NULL)
		if(c->num_elem==0)
		{
			Initialize(c);
		}
	}

	//Retornar al elemento desencolado
	return e;
```

```
}


/*
boolean Empty(cola * c);
Descripción: Recibe una cola y TRUE si la cola esta vacia y FALSE en caso
contrario
Recibe: cola *c (Referencia a la cola "c" a verificar)
Devuelve: boolean TRUE O FALSE
Observaciones: El usuario a creado una cola y la cola fue correctamente
inicializada
*/
boolean Empty(cola * c)
{
	return (c->num_elem==0) ? TRUE : FALSE;
}

/*
elemento Front(cola * c);
Descripción: Recibe una cola y devuelve el elemento que se encuentra al frente de
esta.
Recibe: cola *c (Referencia a la cola "c")
Devuelve: elemento del frente de la cola
Observaciones: El usuario a creado una cola,la cola fue correctamente
inicializada, esta
tiene elementos de lo contrario devolvera ERROR. *No se modifica el TAD
*/
elemento Front(cola * c)
{
	return c->frente->e;
}

/*
elemento Final(cola * c);
Descripción: Recibe una cola y devuelve el elemento que se encuentra al final de
esta.
Recibe: cola *c (Referencia a la cola "c")
Devuelve: elemento del final de la cola
Observaciones: El usuario a creado una cola,la cola fue correctamente
inicializada, esta
tiene elementos de lo contrario devolvera ERROR. *No se modifica el TAD
*/
elemento Final(cola * c)
{
	return c->final->e;
}

/*
int Size(cola * c);
```

```
Descripción: Recibe una cola y devuelve el número de elemento que se encuentran
en esta.
Recibe: cola *c (Referencia a la cola "c")
Devuelve: int (Tamaño de la cola)
Observaciones: El usuario a creado una cola,la cola fue correctamente
inicializada, esta
*No se modifica el TAD
*/
int Size(cola * c)
{
	return c->num_elem;
}

/*
void Element(cola * c, int i);
Descripción: Recibe una cola y un número de elemento de 1 al tamaño de la cola y
retorna el elemento de esa posición.
Devuelve: elemento en la posicion i de la cola
Observaciones: El número i debera estar entre 1 y el tamaño de la cola, si esta
es vacia o mas pequeña se provoca un error.
*/
elemento Element(cola * c, int i)
{
	elemento r;
	nodo *n;
	int j;
	//Si el elemento solicitado esta entre 1 y el tamaño de la cola
	if (i>0&&i<=Size(c))
	{
		//Obtener el elemento en la posición i
		n=c->frente;
		for(j=1;j<i;j++)
			n=n->siguiente;
		r=n->e;
	}
	else
	{
		printf("\nERROR: (Element) Se intenta acceder a elemento
inexistente");
		exit(1);
	}
	return r;
}

/*
void Destroy(cola * c);
Descripción: Recibe una cola y la libera completamente.
Recibe: cola *c (Referencia a la cola "c" a operar)
Devuelve:
```

```
Observaciones: La cola se destruye y se inicializa.
*/
void Destroy(cola * c)
{
	nodo * aux;	//Apuntador auxiliar a elemento

	//Mientras el apuntador del frente de la cola no sea NULL
	while(c->frente != NULL)
	{
		//Guardar la referencia al frente
		aux = c->frente;

		//El nuevo frente es el siguiente
		c->frente = c->frente->siguiente;

		//Liberar el antiguo frente de memoria
		free(aux);
	}

	//El número de elementos en la cola es 0
	c->num_elem=0;

	//Colocar el frente (ya quedo en NULL según como se fue destruyendo) y final
inicializado
	c->final = NULL;

	return;
}
```

Simulación 01.

Presentacion.h

```
/*
Autor: Edgardo Adrián Franco Martínez
Versión 1.0 (25 de Septiembre 2012)
Descripción: Cabecera de la libreria para recrear presentaciones más agradables
al usuario en el modo consola

Observaciones: La implementación de esta libreria es distinta si se trata de
Windows o Linux, ya que requerirá de funciones no ANSI C
*/
//DECLARACIÓN DE FUNCIONES
void MoverCursor( int x, int y );	//Función para mover el cursor de escritura
de pantalla, simulación de la función gotoxy() que se tenia en borland 3.0 en la
libreria conio.h
```

```
void EsperarMiliSeg(int t);           //Función para esperar un tiempo en
milisegundos, simulación de la función delay() que se tenia en borland 3.0 en la
libreria conio.h

void BorrarPantalla(void);            //Función para borrar la pantalla de la
consola, simulación de la función clrscr() que se tenia en borland 3.0 en la
libreria conio.h
```

Simulacion01.c

```
/*************************************************************************
* PROGRAMA: Simulacion01.c                                               *
* AUTORES:                                                               *
* - Oledo Enriquez Gilberto Irving                                       *
* - Alanís Ramírez Damián                                                *
* - Mendieta Torres Alfonso Ulises                                       *
*                                                                        *
* VERSIÓN: 1.5                                                           *
*                                                                        *
* DESCRIPCIÓN: Programa que simula la atención de clientes en un supermerc-  *
* ado, el cual debera atender al menos 100 clientes para poder cerrar y no   *
* tener perdidas, por lo tanto, si ya se atendieron a mas de 100 personas y  *
* no hay gente formada en las filas, la tienda piede cerra, mientras la      *
* tienda                                                                 *
* no cierre los clientes pueden seguir llegando a comprar.               *
*                                                                        *
* Compilación: cd (ruta_archivos)                                        *
* gcc -o Simulacion01 Simulacion01.c TADCola(Est|Din|Cir).o presentacionWin.o *
*************************************************************************/

/*
Aqui se incluyen todas las librerias necesarias para poder hacer uso
de todas las funciones que conforman el programa principal
NOTA: Se puede incluir "TADColaEst.h" - "TADColaCir.h"
*/
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <time.h>
#include <string.h>
#include "TADColaDin.h"
#include "presentacion.h"


//Definición de constantes

#define WidthConsole 90 //Se define el ancho de consola
#define TimetoDraw 0 //Se define el tiempo para dibujar
#define ClientTime 5//Se define el tiempo para dibujar a un cliente
```

```c
//Funcion utilizada para erradicar el "parpadeo"
void hidecursor(void){
 HANDLE consoleHandle = GetStdHandle(STD_OUTPUT_HANDLE);
   CONSOLE_CURSOR_INFO info;
   info.dwSize = 100;
   info.bVisible = FALSE;
   SetConsoleCursorInfo(consoleHandle, &info);
}
/* Nombre de la funcion: DrawCashier
Descripción: Esta funcion dibuja una caja del supermercado
Recibe: 2 enteros(x, y) que son las coordenadas del cursor para comenzar a
dibujar; un entero(IdDrawCashier)que es el numero de la caja; un caracter(Letra
que define el estado actual de la caja), y un entero
(velocidad) que es la velocidad de atencion de cada cajera,
velocidad en que la cajera atendera los clientes
Devuelve: No tiene ningun retorno */
void DrawCashier(int x, int y, int IdDrawCashier, char estado, int vel){ //Se
define la funcion para dibujar una caja
      MoverCursor(x,y); //Se posiciona el cursor antes de comenzar a dibujar la
caja
      printf(" %c%c%c\t",205,205,187);//Se dibuja una parte de la caja (NOTA: Se
utilizaron simbolos del ASCII)
      //Estado (A=Atendiendo)- (F=Cerrada)
      if(estado == 'A'){ //Se crea una condicional para saber si la caja esta
atendiendo
            MoverCursor(x+7, y);//Funcion encontrada en "presentacion".h
            printf("                                  ");
            MoverCursor(x+7, y);//Funcion encontrada en "presentacion".h
            printf("Atendiendo");//Se imprrime si la caja esta atendiendo
      }
      else
            if(estado == 'F'){//Se crea una condicional para saber si la caja esta
cerrada
                  MoverCursor(x+7, y);//Funcion encontrada en "presentacion".h
                  printf("                                  ");
                  MoverCursor(x+7, y);//Funcion encontrada en "presentacion".h
                  printf("CAJA CERRADA\n");//Se imprime si la caja esta cerrada
            }
            else{ //La caja esta abierta pero esta vacia
                  MoverCursor(x+7, y);//Funcion encontrada en "presentacion".h
                  printf("                                  ");
                  MoverCursor(x+7, y);//Funcion encontrada en "presentacion".h
                  printf("Caja Vacia");
            }
      MoverCursor(x+2,y+1);//Funcion encontrada en "presentacion".h
      printf(" %c\t    Caja %d",186, IdDrawCashier+1);//Se dibuja la parte central
de la caja (NOTA: Se utilizaron simbolos del ASCII)
      MoverCursor(x,y+2);//Funcion encontrada en "presentacion".h
```

```
	printf(" %c%c%c\t",205,205,188);//Se dibuja la ultima parte de la caja
(NOTA: Se utilizaron simbolos del ASCII)
	printf("La caja %d  atiende 1 cliente cada %d
milisegundos\n",IdDrawCashier+1, vel);// Se muestra informacion individual sobre
cada caja
	//(Numero de caja, cual es el tiempo que tarda en atender a cada cliente)
}
/* Nombre de la funcion: Cashier
Descripción: Dibuja el numero de cajas indicado e imprime en pantalla: Nombre de
supermercado,
El tiempo de llegada de los clientes, El numero de clientes que ha llegado y el
numero de clientes atendidos
Recibe: La cantidad de cajas que se van a dibujar, el espacio que tendra la fila
de cada cajera,
el estado de cada una de las cajas(El estado se definio anteriormente),
el nombre del supermercado, la velocidad de llegada de los clientes(ingresado
desde el teclado), la velocidad en que cada cajera atendera a los clientes, la
cantidad de clientes que han
llegado y finalmente la cantidad de clientes atendidos.
Devuelve: Un entero que representa una posicion abajo de lo que se dibujo
anteriormente */
int Cashier(int cant, int largo, char estadoCajeras[], char marketName[], int
llegadaClientes, int tiempoAtencion[], int cantClientesLlegados, int
cantClientesAtendidos){
	int i;
	MoverCursor(0,40);
	printf("
");
	MoverCursor(0,41);//Funcion encontrada en "presentacion".h
	printf("|Nombre del supermercado: %s|\t|Recibiendo 1 cliente cada %d
milisegundos|\t\t|Han llegado %d clientes |\t|%d clientes han sido atendidos\t|",
marketName, llegadaClientes, cantClientesLlegados, cantClientesAtendidos);
	if(cantClientesAtendidos>100){ //Condicional que presenta la especificacion
del problema
		printf("\t\tEL SUPERMERCADO CERRARA\n");
	}
	for(i=0; i<cant; i++){
		DrawCashier(largo+1, i*4+1, i, estadoCajeras[i], tiempoAtencion[i]);
	}
	return i*4+5;
}
/* Nombre de la funcion: Paint
Descripcion: Dibuja cada uno de los clientes del supermercado
Recibe: la fila en la que se dibujara, la posiscion, el cliente y un caracter que
describe si
la fila se ha terminado.
Devuelve: No tiene ningun retorno */
void Paint(int filaPaint, int Posicion, elemento Cliente, char acaboFila){
	char temp[4];//Creamos un arreglo auxiliar de caracteres
```

```
	itoa(Cliente.n,temp,10);// Convierte a cadena el entero que esta dentro de la estructura elemento
	int x = WidthConsole - ClientTime*Posicion - 3; //cordenada en x donde se dibujara
	int y = filaPaint * 4 + 2; //cordenada en y donde se dibujara
	if(acaboFila == 'S'){ //Se crea una condicional para comprobar el espacio disponible en la cola
		for(int i=0; i<x; i++){//Se limpia la cola mediante un ciclo for
			MoverCursor(i,y);//Recorrer el cursor desde el inicio hasta el final de la cola
			printf(" ");
		}
	}
	else{
		MoverCursor(x,y);
		printf("     ");
		MoverCursor(x,y);
		fflush(stdout);
		printf(" %c",254);//Se impre al cliente en la cola seleccionada (NOTA: Se utilizaron simbolos del ASCII)
	}
	return;
}

//Programa principal (Main)

int main(void){
	//Se definenen todas las varibles y solo son de dos tipos int y char
	int tiempo, cliente, atendidos, minClientes, fila,columna, alto, cantCajeras, filaElegida, llegadaClientes;
	elemento aux;
	char marketName[30], fin = 'N'; //Fin es utilizado para declarar el estado donde el supermercado cerrara
system("cls");//Se limpia la pantalla antes de comenzar el programa
	//Con ayuda de MoverCursor
	MoverCursor(40,2); printf ("****************************************\n");
	MoverCursor(43,4); printf ("Instituto Politecnico Nacional\n");
	MoverCursor(43,5); printf ("Escuela Superior de Computo\n");
	MoverCursor(43,6); printf ("Estructuras de Datos: Simulacion 01\n");
	MoverCursor(40,8); printf ("****************************************\n\n");
	MoverCursor(5,10);
	printf ("Ingrese los datos que a continuacion de solicitan\n\n");
	printf("  Ingrese el nombre del supermercado: ");
	scanf("%s", &marketName);
	printf("  Ingrese la cantidad de cajeras (0-10): ");
	scanf("%d", &cantCajeras);

	if(cantCajeras>10 || cantCajeras < 1){ //Condicion de la especificacion del problema
```

```
          //el numero de cajeras solo puede ser mayor o igual a 1 y menor o
igual a 10
          printf("ERROR: Cantidad de cajeras no valida\n"); //Si el numero de
cajeras no esta dentro de este rango se
          //dice que la cantidad de cajeras no es valida
          exit(0);
     }

     printf("  Ingrese la cantidad minima de clientes atentidos para que el
supermercado pueda cerrar: ");
     scanf("%d", &minClientes);//Cantidad minima de clientes atendidos para que
el supermercado pueda cerra

     char estadoCajeras[cantCajeras];// Arreglo para almacenear el estado de las
cajeras
     int tiempoAtencion[cantCajeras];//Arreglo para almacenar la velocidad a la
que atenderan las cajeras
     cola filaCajera[cantCajeras];// Se crean las filas (colas) donde se
almacenaran a los clientes
     for(int i=0; i<cantCajeras; i++)
          Initialize(&filaCajera[i]);//Se inicializan todas las filas (colas)

     for(int i=0; i<cantCajeras; i++){
          printf("  Tiempo de atencion de la caja #%d: ", i+1);
          scanf("%d", &tiempoAtencion[i]);//Se guardan los tiempos de antencion
de cada una de las cajas en milisegundos (Multiplos de 10)
     }
     printf("  Ingrese el tiempo de llegada de los Clientes: ");
     scanf("%d", &llegadaClientes);//Se guardan los tiempos de llegada de los
clientes

     alto = Cashier(cantCajeras, WidthConsole, estadoCajeras, marketName,
llegadaClientes, tiempoAtencion, cliente, atendidos);
     tiempo = 0; cliente = 0; atendidos = 0;//Inicializamos las variables

     system("cls");//Se limpia la pantalla
     hidecursor();//Ocultamos el cursor para que no parpadee
     srand(time(NULL));//Se generan numeros aleatorios

     while(fin == 'N'){//Se hace un ciclo que representara el tiempo en que esta
funcionando el supermercado
          tiempo = tiempo + 50;//Se aumenta en 50 el tiempo cada que entre en el
ciclo (Se tuvo que hacer así porque de otra forma no era congruente)
          fin = 'S';//El supermecado cerrara

          for(int i=0; i<cantCajeras; i++){//Se inicia un ciclo desde 0 hasta el
numero de cajeras
               if((!Empty(&filaCajera[i])) && estadoCajeras[i] != 'F'){//Si la
fila no esta vacia y la caja no esta cerrada
```

```
                    if(tiempo % tiempoAtencion[i] == 0){//Si ya se ha atendido
un clinete
                        aux = Dequeue(&filaCajera[i]);//Se remueve el cliente
de la fila

                        atendidos++;//Se aumenta en uno el contador de
clientes antendidos
                    }
                    estadoCajeras[i] = 'A'; //Se cambia el estado de la caja a
Atendiendo
               }
               else{

                    estadoCajeras[i] = 'Z';//Se cambia el estado de la caja a
vacia
                    if(atendidos>minClientes)//Si ya se cumplio con el minimo de
clientes atendidos
                        estadoCajeras[i] = 'F'; //Se cambie el estado de la
caja a cerrada
               }
               //Se dibuja la simulacion actualizandola cada vez que entre al
ciclo
               Cashier(cantCajeras,WidthConsole, estadoCajeras, marketName,
llegadaClientes, tiempoAtencion, cliente, atendidos);
          }

          //Se dibujan los clientes
          for(int i=0; i<cantCajeras; i++){//Se inicia un ciclo que vaya desde 0
hasta el numero de cajeras
               //Se incia un ciclo que vaya desde 0 hasta el numero de cajeras
               for(int j=0; j<=Size(&filaCajera[i]); j++){

                    if(j == Size(&filaCajera[i])){
                        Paint(i,j-1,aux, 'S');
                    }
                    else{
                        aux = Element(&filaCajera[i],j+1);
                        Paint(i, j, aux, 'N');
                    }
               }
          }
          //Cuando se ha antedido a un cliente y el supermecado sigue abierto
          if((tiempo % llegadaClientes == 0) && atendidos<=minClientes){
               cliente++;//Se aumenta la cantidada de clientes
               aux.n = cliente;//Se almacenan al cliente dentro del elemento
auxiliar para poder encolarlo
               filaElegida = rand()%cantCajeras;//Asignmos la fila
aleatoriamente al cliente
```

```
                Queue(&filaCajera[filaElegida], aux);//Encolamos en la fila(cola)
al cliente
           }



           //Sleep(TimetoDraw); // Se aplica sleep para "dormir al programa con
el tiempo TimetoDraw"

           EsperarMiliSeg(1);
           for(int a=0; a<cantCajeras; a++){//Se verifica que todas las cajas
esten funcionando
                if(estadoCajeras[a] != 'F')
                     fin = 'N';//Se verifica que todas las cajas esten
funcionando

                MoverCursor(0, alto+5); //Se mueve el cursor para poder terminar
en programa sin que se empalmen los textos
                //printf("%c\n", fin);

           }
     }

     MoverCursor(30,44);//Se mueve el cursor para poder terminar en programa sin
que se empalmen los textos
     printf("Fin del programa, presione cualquier tecla para salir...");
     setbuf(stdin, NULL);
     getchar();
     return 0;
}
```

Simulacion 02.

Tabla.h

```
/*************************************************************************

LIBRERIA: tabla.h
AUTORES:

- Oledo Enriquez Gilberto Irving

- Alanís Ramírez Damián
- Mendieta Torres Alfonso Ulises
VERSIÓN: 2.4

DESCRIPCIÓN: Las funciones de esta librería permiten generar la tabla
con los datos que se introducen por el usuario para cada proceso (nom-
```

```
 bre, id, descripción, tiempo, etc) de n renglones. Cada renglón es ca-
 paz de ajustar dinámicamente su altura, en función de cuantos espacios
 se requeririan para imprimir el dato completo sin que este se encime o
 desaparezca parte del mismo al imprimir el dato del siguiente campo
 sobre este.


**********************************************************************/
#include <stdio.h>
#include <conio.h>
#include "gotoxy.h"

/*
void imprimirParedes(int coordenada Y)
Función que imprime una pared (un caracter |) marcando el final de cada campo en
el eje x,
es decir, es el delimitador de las columnas.
Argumentos:
- int coordenadaY (es la coordenada de la consola en la que se imprimirá la
pared)
Retorna: void
*/
void imprimirParedes(int coordenadaY){
     int posicion[5] = {0, 11, 21, 71, 79}, i; //posicion[] es para imprimir a lo
largo del eje X (   |      |            |      |)
     for(i = 0; i < 5; i++){
          gotoxy(posicion[i], coordenadaY);
          printf("%c", 179);
     }
     return;
}

/*
void imprimirRenglon(char cadena[], int coordenada Y, int coordenadaX, int tipo)
Función que imprime el dato en la tabla, de tal manera que si la longitud de la
cadena
a imprimir supera el espacio horizontal por defecto (es decir, 9 espacios para el
tipo 0 y
49 para el tipo 1, el cual es la descripción, ya que este cuenta con 49 espacios
sobre el
eje x para poder imprimirse, es la columna más amplia).
Si se sobrepasa se continua imprimiendo el dato pero con un desplazamiento, es
decir, salta
a la siguiente posición en y y se continua imprimiendo.
Argumentos:
- char cadena[] (es el dato a imprimir en su campo: nombre, id o descripción)
- int coordenadaY (es la coordenada y de la consola en la que se encuentra el
caracter a imprimir)
```

```
- int coordenadaX (es la coordenada x de la consola en la que se encuentra el caracter a imprimir)
- int tipo (indica si se trata de un dato con un ancho de columna permitido de 9 espacios por renglón-
id y nombre- o uno de 49 espacios por renglón - descripción -)
Retorna: desplazamiento (retorna el desplazamiento que sufrió el renglón para poder imprimirse correctamente)
*/
int imprimirRenglon(char cadena[], int coordenadaY, int coordenadaX, int tipo){
	int j, desplazamiento, x[4];
	if(tipo == 1){ //Ayuda a indicar si se trata de una columna tipo 1 (de 49 espacios) o de una tipo 0 (de 9 espacios)
		x[0] = 49;
		x[1] = 98;
		x[2] = 147;
		x[3] = 196;
		if(strlen(cadena) < 50) desplazamiento = 0; //Si la cadena se puede imprimir en menos de 50 espacios entonces no habrá salto de renglón
	}
	else{
		x[0] = 9;
		x[1] = 19;
		x[2] = 29;
		x[3] = 39;
		if(strlen(cadena) < 10) desplazamiento = 0;
	}
	for(j = 0; j < strlen(cadena); j++){ //Mini algoritmo para la impresión de la descripción completa
		if(j == x[0] || j == x[1] || j == x[2] || j == x[3]){
			imprimirParedes(coordenadaY);
			if(j == x[0]){ //Si la longitud que va hasta ahora de la cadena es igual a 9
				imprimirParedes(coordenadaY + 1); //Imprime las paredes en el siguiente, sin cerrar todavía el renglón con una linea horizontal
				gotoxy(coordenadaX, coordenadaY + 1);
				printf("%c", cadena[j]);
				desplazamiento = 1; //El desplazamiento ya vale 1, pues ya se tuvo que aumentar la altura del renglón para darle cabida al dato
			}
			if(j == x[1]){
				imprimirParedes(coordenadaY + 2);
				gotoxy(coordenadaX, coordenadaY + 2);
				printf("%c", cadena[j]);
				desplazamiento = 2;
			}
			if(j == x[2]){
				imprimirParedes(coordenadaY + 3);
				gotoxy(coordenadaX, coordenadaY + 3);
				printf("%c", cadena[j]);
```

```
                        desplazamiento = 3;
                  }
                  if(j == x[3]){
                        imprimirParedes(coordenadaY + 4);
                        gotoxy(coordenadaX, coordenadaY + 4);
                        printf("%c", cadena[j]);
                        desplazamiento = 4;
                  }
            }
            else{
                  printf("%c", cadena[j]); //En caso de que se encuentre dentro del
campo de impresión se imprime normalmente el caracter
            }
      }
      return desplazamiento;
}

/*
void imprimirDatos(cola *nombre, cola *id, cola *actividad, cola *tiempo, int
desplazamiento, int numeroItem, int y)
Función que se encarga de la impresión de la tabla con los datos, ya que aquí es
donde se obtienen los datos directamente de las colas
que se tienen como argumentos y se pasan a la función imprimirRenglon().
Se apoya del desplazamiento generado por la función imprimirRenglon() para ir
imprimiendo adecuadamente cada renglón sin que se encime.
Al final el desplazamiento definitivo lo determinará el dato que más se haya
desplazado. Por ejemplo, si se tiene un nombre de tamaño 12
entonces este se imprime del 1 al 9 en el renglón 1 y luego se desplaza 1 unidad
en la coordenada Y y se imprimen los caracteres restantes
en el siguiente renglón. Pero si se tiene una descripción de 100 caracteres,
entonces esta ya sufrió 2 saltos o desplazamientos, por lo que
el desplazamiento total de esa fila va a ser el del dato que sufrió el mayor
desplazamiento, en este caso, la descripción.
Argumentos:
- cola *nombre (puntero a cola, servirá para pasarle la cola nombre a la función)
- cola *id (puntero a cola, servirá para pasarle la cola id a la función)
- cola *actividad (puntero a cola, servirá para pasarle la cola actividad - la
cual contiene las descripciones - a la función)
- cola *tiempo (puntero a cola, servirá para pasarle la cola tiempo a la función)
- int desplazamiento (es un desplazamiento de acarreo, es decir, que se va
generando con cada iteración, inicialmente es 0 en la función que llama
a imprimirDatos())
- int numeroItem (da el elemento a obtener para su impresión, viene dado por el
contador que se encuentra en la función dibujarTabla(), que es la
que llama a esta función)
- int y (es la coordenada Y en la cual se imprimiran los datos, a esta se le
sumará el desplazamiento de acarreo)
Retorna: void
*/
```

```
void imprimirDatos(cola *nombre, cola *id, cola *actividad, cola *tiempo, int
desplazamiento, int numeroItem, int y){
	int i, j, desplazamientoNombre, desplazamientoId, desplazamientoActividad;
	elemento e1;
	char tabla[6] = {218, 179, 196, 191, 217, 192};
	gotoxy(0, y + desplazamiento);
	putchar(tabla[1]);
	gotoxy(1, y + desplazamiento);
	e1 = Element(nombre, numeroItem); //guarda en e1 el elemento numeroItem
	desplazamientoNombre = imprimirRenglon(e1.array, y + desplazamiento, 1, 0);
//Se imprime el nombre, guardando su desplazamiento en
	gotoxy(11, y + desplazamiento); //desplazamientoNombre,
	putchar(tabla[1]);
	e1 = Element(id, numeroItem);
	desplazamientoId = imprimirRenglon(e1.array, y + desplazamiento, 12, 0);
	gotoxy(21, y + desplazamiento);
	putchar(tabla[1]);
	e1 = Element(actividad, numeroItem);
	desplazamientoActividad = imprimirRenglon(e1.array, y + desplazamiento, 22,
1);
	gotoxy(71, y + desplazamiento);
	printf("%c",tabla[1]);
	e1 = Element(tiempo, numeroItem);
	gotoxy(72, y + desplazamiento);
	printf("%ds", e1.n); //Se imprime el tiempo
	gotoxy(79, y + desplazamiento);
	putchar(tabla[1]);
	if(desplazamientoActividad >= desplazamientoNombre &&
desplazamientoActividad >= desplazamientoId){ //En caso de que el
		desplazamiento = desplazamiento + desplazamientoActividad;
	//desplazamiento de la actiniad sea el mayor de todos
	}
	else{
		if(desplazamientoNombre >= desplazamientoActividad &&
desplazamientoNombre >= desplazamientoId){ //En caso de que el
			desplazamiento = desplazamiento + desplazamientoNombre;
		//desplazamiento del nombre sea el mayor de todos
		}
		else desplazamiento = desplazamiento + desplazamientoId; //Por
descarte se tiene que si el control llega hasta aquí es porque
	}
	//el desplazamiento del id es el mayor
	gotoxy(0, y + 1 + desplazamiento); //Se imprimen las esquinas inferiores de
la tabla
	putchar(tabla[5]);
	gotoxy(79, y + 1 + desplazamiento);
	putchar(tabla[4]);
	gotoxy(1, y + 1 + desplazamiento);
```

```
	for(i = 0; i < 78; i++) printf("-"); //Al final de cada renglón imprime la linea horizontal para indicar su fin
}

/*
void dibujarEncabezados(int y, char titulo[])
Función que imprime la parte de arriba de la tabla, es decir, el encabezado, el cual es
 ---------------------------------------------------------------------------
 |Nombre      |     ID     |          Descripción            | Tiempo  |
 ---------------------------------------------------------------------------
 Se considero pertinente hacerlo una función ya que será un dato estático que se imprimirá
 constantemente.
 Argumentos:
- int y (es la coordenada de la consola en la que se posicionará)
- char titulo[] (es la etiqueta de la cola, ejemplo: listos, finalizados, etc)
Retorna: void
*/
void dibujarEncabezados(int y, char titulo[]){
	int i;
	char tabla[6] = {218, 179, 196, 191, 217, 192};

	gotoxy((80-(strlen("Cola de") + strlen(titulo)))/2, y); //Va a la coordenada x en la que se imprima centrada la cabecera
	printf("Cola de %s", titulo);
	gotoxy(0, y + 1);
	putchar(tabla[0]); //Imprime la tabla con las etiquetas de cada columna (nombre, id, descripción y tiempo), es decir, los campos
	for(i = 0; i < 78; i++) printf("-");
	gotoxy(79, y + 1);
	putchar(tabla[3]);
	gotoxy(0, y + 2);
	putchar(tabla[1]);
	gotoxy(3, y + 2);
	printf("Nombre");
	gotoxy(11, y + 2);
	putchar(tabla[1]);
	gotoxy(15, y + 2);
	printf("ID");
	gotoxy(21, y + 2);
	putchar(tabla[1]);
	gotoxy(40, y + 2);
	printf("Descripcion");
	gotoxy(71, y + 2);
	putchar(tabla[1]);
	gotoxy(73, y + 2);
	printf("Tiempo");
	gotoxy(79, y + 2);
```

```
	putchar(tabla[1]);
	gotoxy(0, y + 3);
	putchar(tabla[5]);
	gotoxy(79, y + 3);
	putchar(tabla[4]);
	gotoxy(1, y + 3);
	for(i = 0; i < 78; i++) printf("-");
	return;
}

/*
void dibujarTabla(int anchoColumnas[], cola *nombre, cola *id, cola *actividad,
cola *tiempo, int y, char titulo[], int tam)
Función que se encarga de llamar a las otras funciones esenciales para dibujar la
tabla, además, es la que tiene el contador
que va recorriendo los elementos encolados, y que proceden de las colas en las
que se almacenaron los datos introducidos por
el usuario, y los va imprimiendo de manera coherente (sin encimarse ni
recortarse). Es importante observar que se tiene un
desplazamiento de acarreo o ancho de columna que viene desde el mismo programa
colaDeProcesosSO.c, y que depende de si el
dato desde un inicio se sabe que va a rebasar el espacio para su impresión.
Argumentos:
- int anchoColumnas[] (arreglo que contiene los desplazamientos en concreto para
cada dato conforme fueron introducidos)
- cola *nombre (puntero a cola, servirá para pasarle la cola nombre a la función)
- cola *id (puntero a cola, servirá para pasarle la cola id a la función)
- cola *actividad (puntero a cola, servirá para pasarle la cola actividad - la
cual contiene las descripciones - a la función)
- cola *tiempo (puntero a cola, servirá para pasarle la cola tiempo a la función)
- int y (coordenada y en la que se posicionará)
- char titulo[] (indica que tipo de cola será, de ejecución, de finalizado)
- int y (es la coordenada Y en la cual se imprimiran los datos, a esta se le
sumará el desplazamiento de acarreo)
- int tam (indica el numero de iteraciones a realizar, depende del numero de
procesos introducidos por el usuario, es decir, si en
colDeProcesosSO el usuario introdujo 7 datos, entonces se deberá realizar este
ciclo 7 veces, ya que cada ciclo imprime un dato)
Retorna: void
*/
void dibujarTabla(int anchoColumnas[], cola *nombre, cola *id, cola *actividad,
cola *tiempo, int y, char titulo[], int tam){
	int desplazamiento = 0, i, j;
	elemento e1;
	dibujarEncabezados(y - 4, titulo);
	for(i = 0; i < tam; i++){
		imprimirDatos(nombre, id, actividad, tiempo, desplazamiento, i + 1,
y);
```

```
            desplazamiento = desplazamiento + anchoColumnas[i] + 1; //En el
desplazamiento en y influye el desplazamiento de acarreo (el que
      } //venía desde que el usuario introdujo los datos), y el propio
desplazamiento anterior más 1
      return;
}
```

coladeProcesosSO.c

```
/*****************************************************************************
* PROGRAMA: coladeProcesosSO.c                                               *
* AUTOR: Alanís Ramírez Damián                                               *
* VERSIÓN: 1.2                                                               *
*                                                                            *
* DESCRIPCIÓN: Programa que simula la cola de procesos de un SO monopro-*
* cesador, en la cual existen procesos con su nombre, ID, descripción y *
* tiempo de ejecución.                                                       *
* Cada proceso entra en la cola de pendientes y el que este en el tope  *
* entrará en ejecución. Cada proceso terminado se encolará en la cola de*
* terminados y se mostrará al final sus datos, así como el tiempo total *
* en la cola: t = tiempoProceso + tiempoColaPendientes.                      *
*                                                                            *
* Compilación: cd (ruta_archivos)                                            *
* gcc -o colaDeProcesosSO colaDeProcesosSO.c presentacion(Win|Lin).c    *
* TADCola(Est|Din|Circ).c                                                    *
*****************************************************************************/
//LIBRERÍAS
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include "lib\gotoxy.h"
#include "lib\TADColaDin.h"
#include "lib\tabla.h"

//DECLARACIÓN DE FUNCIONES
void imprimir(char const *cadena, int y);
void letreros();
int otroProceso();
void colaDeProcesos();
void pedirProceso();

//MAIN
int main(){
      do{
            cleanScreen();
            pedirProceso();
      }while(otroProceso() == 1);
      getchar();
      return 0;
```

```
}

//DEFINICIÓN DE FUNCIONES

/*
void imprimir()
Función que imprime una cadena centrada y en la coordenada y especificada.
Argumentos:
- char const *cadena (una cadena a imprimir)
- int y (la coordenada y donde se va a imprimir)
Retorna: void
*/
void imprimir(char const *cadena, int y){
      gotoxy((80-strlen(cadena))/2, y);
      puts(cadena);
      return;
}

/*
void letreros()
Función que imprime los letreros de presentación del
programa.
Argumentos: void
Retorna: void
*/
void letreros(){
      imprimir("Instituto Politecnico Nacional", 0);
      imprimir("ESCOM", 1);
      imprimir("Cola de procesos de Sistema Operativo", 2);
      return;
}

/*
int otroProceso()
Función que pregunta al usuario si desea introducir otro proceso (ya sea
durante la función pedirProceso() o al final cuando debe decidir si reiniciar
el programa para simular otra cola de procesos, o salir del programa).
Argumentos: void
Retorna: 1 si selecciona 's'|'S' y 2 si selecciona 'n'|'N'
*/
int otroProceso(){
      char sn;
      printf("\nDesea introducir otro proceso(s|S,n|N)?: ");
      setbuf(stdin, NULL);
      scanf("%c", &sn);
      tolower(sn);
      return(sn == 's')?1:2;
}
```

```
/*
void colaDeProcesos(int anchoColumnas[], cola *nombre, cola *id, cola *actividad,
cola *tiempo)
Función que simula la cola de procesos del SO, mostrando gráficamente (con
interfaz con caracteres
ASCII) la cola de ejecución y la cola de terminados, en las cuales aparecen
nombre, id, descripción y
tiempo de ejecución de cada proceso.
Al final muestra la cola de terminados y el tiempo total de ejecución.
Argumentos:
- int anchoColumnas[] (es un arreglo que indica el ancho de la columna a imprimir
en
función del cálculo que se hace en la función pedir proceso)
- cola *nombre, cola *id, cola *actividad, cola *tiempo (se pasan por referencia)
Retorna: void
*/
void colaDeProcesos(int anchoColumnas[], cola *nombre, cola *id, cola *actividad,
cola *tiempo){
      char nom[32], ids[32];
      int t;
      cola nombreFinalizado, idFinalizado, actividadFinalizado, tiempoFinalizado;
      elemento e1;
      int i, j, desplazamiento = 0, tiempoTotal = 0, tiempoEjecucion = 0;
      Initialize(&nombreFinalizado);
      Initialize(&idFinalizado);
      Initialize(&actividadFinalizado);
      Initialize(&tiempoFinalizado);
      i = 0;
      while(Empty(tiempo) == FALSE){ //Mientras la cola tiempo (aunque podría ser
cualquiera de las que se recibieron, ya que tienen el mismo tamaño)
            letreros();                              //no este vacía
            dibujarTabla(anchoColumnas, nombre, id, actividad, tiempo, 10,
"ejecucion", 1);//Dibuja la tabla que contiene al proceso en ejecución
            //dibujarTabla(anchoColumnas, &nombreFinalizado, &idFinalizado,
&actividadFinalizado, &tiempoFinalizado, 19, "finalizados",
Size(&nombreFinalizado));
            while(Empty(tiempo) == FALSE){ //Algoritmo de encolado y desencolado
para paso de cola de espera a ejecución cuando transcurre un quantum
                  cleanScreen();
                  letreros();
                  dibujarTabla(anchoColumnas, nombre, id, actividad, tiempo, 10,
"ejecucion", 1);//Dibuja la tabla que contiene al proceso en ejecución
                  dibujarTabla(anchoColumnas, &nombreFinalizado, &idFinalizado,
&actividadFinalizado, &tiempoFinalizado, 28, "finalizados",
Size(&nombreFinalizado));
                  if(i != 0){
                        e1 = Element(nombre, Size(tiempo));
                        strcpy(nom, e1.array);
                        e1 = Element(id, Size(tiempo));
```

```
                        strcpy(ids, e1.array);
                        e1 = Element(tiempo, Size(tiempo));
                        t = e1.n;
                        imprimir("Ultimo proceso Ejecutado: ", 14);
                        gotoxy(0,15);
                        printf("Nombre = %s \t ID = %s \tTiempo para que finalice
= %d", nom, ids, t);
                        imprimir("Proximo proceso a ejecutarse: ", 18);
                        if(Size(nombre) > 1){
                              e1 = Element(nombre, 2);
                              strcpy(nom, e1.array);
                              e1 = Element(id, 2);
                              strcpy(ids, e1.array);
                              e1 = Element(tiempo, 2);
                              t = e1.n;
                        }
                        else{
                              e1 = Front(nombre);
                              strcpy(nom, e1.array);
                              e1 = Front(id);
                              strcpy(ids, e1.array);
                              e1 = Front(tiempo);
                              t = e1.n;
                        }
                        gotoxy(0,19);
                        printf("Nombre = %s \t ID = %s \tTiempo para que finalice
= %d", nom, ids, t);
                  }
                  e1 = Dequeue(tiempo);
                  gotoxy(72,10);
                  printf("%ds", e1.n);
                  delay_ms(1000);
                  if(e1.n > 0){ //Si el tiempo de ese proceso aun no acaba (t == 0
es el criterio de finalización)
                        e1.n--;
                        Queue(tiempo, e1); //Encola el nuevo tiempo
                        Queue(nombre, Dequeue(nombre)); //Encola (lleva al final) el
nombre que se encontraba al principio
                        Queue(id, Dequeue(id));
                        Queue(actividad, Dequeue(actividad));
                  }
                  else{
                        e1.n = tiempoEjecucion;
                        Queue(&tiempoFinalizado, e1); //Se encola el tiempoEjecucion
de ese proceso en tiempoFinalizado
                        Queue(&nombreFinalizado, Dequeue(nombre)); //Se desencola el
nombre de la cola de ejecución y se encola en finalizado
                        Queue(&idFinalizado, Dequeue(id));
                        Queue(&actividadFinalizado, Dequeue(actividad));
```

```
                    break;
               }
               tiempoEjecucion++;
               i++;
          }
          //imprimir("Ultimo proceso Ejecutado: ", 14);
          //gotoxy(10,15);
          //printf("Nombre = %s \t ID = %s \tTiempo para que finalice = %d",
nom, ids, t);
          dibujarTabla(anchoColumnas, &nombreFinalizado, &idFinalizado,
&actividadFinalizado, &tiempoFinalizado, 28, "finalizados",
Size(&nombreFinalizado));
          if(Empty(nombre) == TRUE){ //Una vez que se vacía la cola de ejecución
               printf("\n\tEl SO ha ejecutado todos los procesos en la cola.
\n\tPresione cualquier tecla para continuar ...");
               getchar();
          }
     }
     cleanScreen();
     letreros();
     dibujarTabla(anchoColumnas, &nombreFinalizado, &idFinalizado,
&actividadFinalizado, &tiempoFinalizado, 10, "finalizados",
Size(&tiempoFinalizado));
     printf("\n\t\tEl tiempo total de ejecucion fue de %ds \n", tiempoEjecucion);
//Se indica el tiempo (tiempo de proceso 1 + proceso 2 + ... + proceso n)
     Destroy(&nombreFinalizado); //Destruye las colas pensando en un posible
nuevo proceso
     Destroy(&idFinalizado);
     Destroy(&actividadFinalizado);
     Destroy(&tiempoFinalizado);
     Destroy(nombre);
     Destroy(id);
     Destroy(actividad);
     Destroy(tiempo);
     return;
}


/*
void pedirProceso()
Función que pide los datos del proceso, es decir, id, nombre, descripción y
tiempo de
ejecución, y los va almacenando en colas con la misma etiqueta del dato (hay una
cola
para nombres, otra para idÂ´s, etc). Asimismo, llama a las funciones
dibujarTabla() (de
la librería tabla.h) y colaDeProcesos.
Argumentos: void
Retorna: void
```

```
*/
void pedirProceso(){
      int i = 5, j = 0, tam[20], tiempo;
      char id[32], nombre[32], actividad[200], temp[200];
      cola nombres, actividades, ids, tiempos;
      elemento e1;
      Initialize(&nombres);
      Initialize(&actividades);
      Initialize(&ids);
      Initialize(&tiempos);
      do{
            tam[j] = 1; //tam[j] nos indicará el ancho de columna de acuerdo a la
longitud de las cadenas (nombre, id y actividad)
            setbuf(stdin, NULL);
            letreros();
            imprimir("A continuacion introduzca los parametros del proceso que se
le solicitan: ", 4);
            imprimir("Precondiciones (strlen(nombre) y strlen(id)) <= 32,
strlen(descripion) <= 200", 5);
            gotoxy(5,6);
            printf("Introduzca el nombre del proceso: ");
            gets(nombre);
            if(strlen(nombre) >= 10) tam[j] = strlen(nombre)/10 + 1; //el tamaño
del renglón donde se imprimirá es de 10 así que la
            gotoxy(5,7);
      //columa será de ancho igual al número de veces que exceda el 10
            printf("Introduzca el ID del proceso: ");
            gets(id);
            if(strlen(id) >= 10) tam[j] = strlen(id)/10 + 1;
            gotoxy(5,8);
            printf("Introduzca la descripcion: ");
            gets(actividad);
            i = 9;
            if(strlen(actividad) > 48){//Ciclo para dar saltos de linea durante la
introducción de la cadena en caso de que sea de longitud mayor a
                  i = i + (strlen(actividad) - 48)/80 + 1; //48 (80 - 32 que es la
longitud del letrero "Introduzca ..." + el desplazamiento en x) y que
            }                                                          //así no
se afecte la impresión de los siguientes letreros (que no se encime)
            gotoxy(5,i);
            printf("Introduzca el tiempo de ejecucion (en segs): ");
            scanf("%d", &tiempo);
            if(i > 9){ //Se asume que si i  es mayor a 9, entonces actividad
superaría su espacio de impresión en el renglón
                  tam[j] = strlen(actividad)/50 + 1;
            }
            strcpy(e1.array, nombre); //Se copia el nombre al elemento e1 en su
parte array[]
            Queue(&nombres, e1);    //Se encola el nombre en la cola nombres
```

```
            strcpy(e1.array, actividad);
            Queue(&actividades, e1);
            strcpy(e1.array, id);
            Queue(&ids, e1);
            e1.n = tiempo;
            Queue(&tiempos, e1);
            if(otroProceso() == 2) break; //Se pregunta si se desea introducir
otro proceso, en caso de que no, se sale del ciclo
            cleanScreen();
            j++;
            if(j == 19) break; //Si se ha superado el número máximo de datos a
introducir
      }while(1);
      dibujarTabla(tam, &nombres, &ids, &actividades, &tiempos, 18, "espera",
Size(&tiempos)); //Dibuja la tabla con los datos introducidos
      getchar();
      printf("\nPresione cualquier tecla para proceder a la inicializacion de la
Cola de proceso");
      getchar();
      cleanScreen();
      delay_ms(500);
      colaDeProcesos(tam, &nombres, &ids, &actividades, &tiempos); //Llama a la
función que simulará la cola de procesos
      return;
}
```

Simulacion 03.

Banco.c

```
/*Instituto Piolitécnico Nacional: ESCOM          *
 *    Estructura de datos                          *
 *    Autores: Alanis Ramirez Damian               *
 *                Mendieta Torres Alfonso Ulises   *
 *                Oledo Enriquez Gilberto Irving   *
 *               Octubre 2017                      *
 *    version: 1.3                                 *
 ************************************************/

/*Descripcion, programa que siula la atencion de un banco, cumpliento con sus
politicas de atencion de visitantes*/
// Para compilar en terminal  gcc -o banco banco.c TADColaEst|Din.c
// Para compilar en terminal  gcc -o banco banco.c TADCOlaEst|Din.o

#include <stdio.h>
#include <windows.h>
#include "TADColaDin.h"
#include "dibujarCajas.h"
```

```
int vacio(int q);

#define Tiempobase 10 //Tiempo base base del problema
#define impresion 20 // cada 20 se imprimira los nuevo usuarios
#define limpiar 300 // cada 1000 ms se limpiara la pantalla

void main()
{
	elemento e, e1, e2; // elementos utilizados e para desencolar e1 para encolar y e2 para imprimir las colas
                                    // se considero 3 elementos ya que no imprimia bien si solo era un elemento utilizado

	cola preferentes;
	Initialize(&preferentes);
	cola usuarios;
	Initialize(&usuarios);
	cola clientes;
	Initialize(&clientes);
	int i, cajeras, Tiempocliente[3]; // i auxiliar para ciclos for, cajeras guarda el entero de la funcion cajeras,
                                // El Tiempocliente se utilizó para guardar el tiempo de llegada de 0 = Clientes, 1 = Usuarios 2 = Preferentes
	unsigned int base_time = 0;	// auxiliar utilizado para saber si llega o algún cajero atiende algún cliente



	system("cls");
	/* pedirDatos regresa un entero con el número de cajeros solicitados, imprime centradamente el cartel de
	presentacion del programa*/
	cajeras = pedirDatos();

	system("cls");
	// 0 = Clientes, 1 = Usuarios, 2 = Preferentes;
	printf("Ingrese el tiempo de llegada de clientes (ms): ");
	scanf("%d", &Tiempocliente[0]);

	printf("Ingrese el tiempo de llegada de usuarios (ms): ");
	scanf("%d", &Tiempocliente[1]);

	printf("Ingrese el tiempo de llegada de preferentes (ms): ");
	scanf("%d", &Tiempocliente[2]);

	int Tiempocajero[cajeras];	//Es el tiempo de atención de cada cajero de 0 a los cajeros solicitados
```

```
	int cajerovacio[cajeras];		//auxiliares para saber si el cajero de 0 - 10 está vacio

	int c = 0, p = 0, u = 0;		//Auxiliares para conteo de los clientes(c), usuarios(u) y preferentes(p)
	int Pa = 0, Ua = 0;			//Auxiliares para cumplir las políticas de atención

 //Pedimos el tiempo de cada cajero
	for(i=0; i < cajeras; i++)
	{
		printf("Ingrese el tiempo de atencion del cajero %d", i+1);
		scanf("%d", &Tiempocajero[i]);
	}
	fflush(stdout);
	system("cls");
	/* Funcion cajas, es un void que imprime en pantalla los cajeros solicitados se imprimen despues de pedir los datos*/
	Cajas(cajeras);

	// cuando empieza el while significa que el banco emepezó la jornada de trabajo
	while((!Empty(&usuarios) || !Empty(&preferentes) || !Empty(&clientes)) || base_time<=10000) // mientas hya clientes y aun no es hpra de cerrar
	{
		Sleep(Tiempobase);
		base_time++;

		/*atencion de clientes, en cada ciclo revisará si algun cajero esta disponible para atender, se considera que los preferentes se atienden primero, si el cajero esta
		vacio si aun no pasan 3 preferentes consecutivos o si hay preferentes formados y aun no han llegado algún usuario o cliente, en caso de que no se atienda a algun preferente
		entonces se atendera a un cliente si hay clientes en cola y aun no pasan dos consecutivos o hay clientes en cola y aun no han llegado preferentes o usuarios, por último los usuarios
		son los que menos atienden pero no pueden pasar 5 de los demas sin ser atendidos*/

		//Atencion de los clientes <---------------------------
		for(i=0; i < cajeras; i++)
		{
			if(base_time % Tiempocajero[i] == 0)
			{
				fflush(stdout);
				cajerovacio[i] = 0;
				if((Pa < 3 && !Empty(&preferentes)&& vacio(cajerovacio[i]) == 1)|| (!Empty(&preferentes) && Empty(&usuarios) && Empty(&clientes)))
```

```
                    {
                        e1 = Dequeue(&preferentes);// Se obtienen un elemento
de la cola de preferentes para ser atendido
                        Pa++;
                        cajerovacio[i] = e1.n; //se toma en cuenta con fines
estéticos que el cajero este ocupado
                        PasarAcaja(i); //PasarAcaja una funcion que retorna un
void, mueve el cursor abajo del cajero que le toca atender
                        printf("P%d", e1.tipoP);  // el número de preferente
atendido se encuentra en tipoP
                    }
                    else
                        if((!Empty(&clientes) && Ua < 2 &&
vacio(cajerovacio[i]) == 1) || (!Empty(&clientes) && Empty(&usuarios) &&
Empty(&preferentes)))
                        {
                            e1 = Dequeue(&clientes);
                            Ua++;
                            cajerovacio[i] = e1.n;
                            PasarAcaja(i);
                            printf("C%d", e1.tipoC); // el número de cliente
atendido se encuentra en tipoC
                        }
                        else
                            if ((!Empty(&usuarios) && Pa+Ua >= 5) ||
(!Empty(&usuarios) && vacio(cajerovacio[i]) == 1))
                            {
                                e1 = Dequeue(&usuarios);
                                PasarAcaja(i);
                                printf("U%d", e1.tipoU); // el número de
usuario atendido se encuentra en tipoU
                                Pa = 0;//Esto pone a los contadores de
preferentes y clientes en 0, indicando que un usuario ha pasado y que pueden
pasar 5 entre clientes y preferentes
                                Ua = 0;
                            }
                }
            }
            //Fin del ciclo de atencion


            // Tiempocliente[0] = Clientes, Tiempocliente[1] = Usuarios,
Tiempocliente[2] = Preferentes;
            // Encolamos a los visitantes <----------------------------------
            /*Se toma en cuenta las 3 colas y el timepo de llegada de cada
visitante*/
            if(base_time<=10000) // si el tiempo base == 10000 entonces el banco
ya cerró
            {
```

```
                fflush(stdin);
                if(base_time % Tiempocliente[0] == 0)
                {
                      c++; // se guarda el número de cliente empezando en 1 y se
suma 1 cada vez que un cliente llega
                      e.tipoC = c;// tipoC es una parte del elemento para guardar
el número de cliente y asi distinguirse de preferentes y usuarios
                      e.n = 0; // en la parte n del elemento se agrega un número 0
de clientes
                      Queue(&clientes, e);
                }

                if (base_time % Tiempocliente[1] == 0)
                {
                      u++;// se guarda el número de usuario empezando en 1 y se
suma 1 cada vez que un usuario llega
                      e.tipoU = u;// tipoU es una parte del elemento para guardar
el número de usuario y asi distinguirse de preferentes y clientes
                      e.n = 1;// en la parte n del elemento se agrega un número 1
de usuarios
                      Queue(&usuarios, e);
                }

                if (base_time % Tiempocliente[2] == 0)
                {
                      p++;// se guarda el número de preferente empezando en 1 y se
suma 1 cada vez que un preferente llega
                      e.tipoP = p;// tipoP es una parte del elemento para guardar
el número de preferente y asi distinguirse de clientes y usuarios
                      e.n = 2;// en la parte n del elemento se agrega un número 2
de preferentes
                      Queue(&preferentes, e);
                }
          }
          //Fin del encolamiento

          //impresion de las colas<--------------------------------
          if(base_time % impresion == 0)
          {
                if(Size(&preferentes)>=20)// Si el número de elementos en cola es
mayor a 20
                {
                      for(i=1; i< 20; i++)//conociendo que el tamaño de cola es
superior a 20 se puede hacer el ciclo de impresion de esos 20 elementos
                      {
                            e2 = Element(&preferentes, i);
                            gotoxy(40, 20+i);
                            printf("P%d", e2.tipoP);
```

```
                    }
                    for(i=1; i<= Size(&preferentes); i++)//Para que la pantalla
no se baje se imprimira el tamaño de cola - 20 ya impresos
                    {
                          gotoxy(40,40);
                          printf("Y %d mas", (Size(&preferentes)-20));
                    }
              }
              else//en caso de que el tamaño de cola sea inferior a 20 entonces
se da comienzo al ciclo de impresion completo, sin embargo cuando llegue a 20
                    //cambiara a la condicion de tamaño de cola mayor a 20
              {
                    for(i=1; i<= Size(&preferentes); i++)
                    {
                          e2 = Element(&preferentes, i);
                          gotoxy(40, 20+i);
                          printf("P%d", e2.tipoP);
                    }
              }

              if(Size(&usuarios)>=20)// Si el número de elementos en cola es
mayor a 20
              {
                    for(i=1; i< 20; i++)//conociendo que el tamaño de cola es
superior a 20 se puede hacer el ciclo de impresion de esos 20 elementos
                    {
                          e2 = Element(&usuarios, i);
                          gotoxy(25, 20+i);
                          printf("U%d", e2.tipoU);

                    }
                    for(i=1; i<= Size(&usuarios); i++)//Para que la pantalla no
se baje se imprimira el tamaño de cola - 20 ya impresos
                    {
                          e2 = Element(&usuarios, i);
                          gotoxy(25,40);
                          printf("Y %d mas", (Size(&usuarios)-20));
                    }
              }
              else//en caso de que el tamaño de cola sea inferior a 20 entonces
se da comienzo al ciclo de impresion completo, sin embargo cuando llegue a 20
                    //cambiara a la condicion de tamaño de cola mayor a 20
              {
                    for(i=1; i<= Size(&usuarios); i++)
                    {
                          e2 = Element(&usuarios, i);
                          gotoxy(25, 20+i);
                          printf("U%d", e2.tipoU);
                    }
```

```
                }

                if(Size(&clientes)>=20)// Si el número de elementos en cola es
mayor a 20
                {
                      for(i=1; i< 20; i++)//conociendo que el tamaño de cola es
superior a 20 se puede hacer el ciclo de impresion de esos 20 elementos
                      {
                            e2 = Element(&clientes, i);
                            gotoxy(10, 20+i);
                            printf("C%d", e2.tipoC);

                      }
                      for(i=1; i<= Size(&clientes); i++)//Para que la pantalla no
se baje se imprimira el tamaño de cola - 20 ya impresos
                      {
                            e2 = Element(&clientes, i);
                            gotoxy(10,40);
                            printf("Y %d mas", (Size(&clientes)-20));
                      }
                }
                else//en caso de que el tamaño de cola sea inferior a 20 entonces
se da comienzo al ciclo de impresion completo, sin embargo cuando llegue a 20
                      //cambiara a la condicion de tamaño de cola mayor a 20
                {
                      for(i=1; i<= Size(&clientes); i++)
                      {
                            e2 = Element(&clientes, i);
                            gotoxy(10, 20+i);
                            printf("C%d", e2.tipoC);
                      }
                }
          }
          //Fin del algoritmo para imprimir las colas

          if(base_time % limpiar == 0)// Cada cierto tiempo se limpia la
pantalla para borrar a los clientes ya atendidos, si es que se necesitara
          {
                system("cls");
                Cajas(cajeras);
          }
     }
     /*Al finalizar la simulacion se procede a destruir las colas, más util
cuando se utiliza el TADColaDin*/
     Destroy(&preferentes);
     Destroy(&usuarios);
     Destroy(&clientes);
     system("cls");
```

```
	imprimirCentrado("El banco ha cerrado, vuelve otro dia", 10);//mensae de
indicacion de que cerro el banco == finalizacion del programa
	return;
}

int vacio(int q) // Retorna un entero, simulacion de un booleano con un entero,
para saber si el valor es 0 regresa 1 y viceversa
{
	int band1;
	if(q==0)
		band1 = 1;
	else
		band1 = 0;
	return band1;
}
```

Conclusiones

Oledo Enriquez Gilberto Irving

Finalmente puedo concluir que esta práctica fue se mucha ayuda ya que de principio las simulaciones parecían muy complicadas pero una vez que analizas lo que tienes que hacer es ahí donde comienzan a surgir ideas de cómo llegar a una solución y poco a poco saber que tanto la parte visual como la implementación puede quedar mejor de como se había pensado en un inicio.

Entendí en funcionamiento de una cola y por qué es que es obedece el concepto FIFO y que diariamente tenemos contacto con este tipo de estructuras de datos, quizá no de forma totalmente orientada a programación pero si en acciones diarias como el simple hecho de comprar un boleto de metro.

Aprendí cómo es que del lado programador podemos hacer uso del TAD Cola sin necesidad de sabes cómo fue implementado simplemente respetando la especificación de esta estructura de datos.

Por ultimo considero que la práctica fue muy buena para poder entender cuáles son las aplicaciones de las estructuras de datos y como un problema tan cotidiano puede ser resuelto de una manera eficiente con ayuda de estas.

Alanís Ramírez Damián

Con esta práctica percibí una gran mejora en mi capacidad de pensar en estrategias para solucionar los problemas planteados. Aprendí a implementar el tipo de dato abstracto (TAD)

conocido como cola, el cual cuenta con sus propiedades, como que se realizan inserciones por el final y se extrae por el frente, análogo a una cola o fila de la vida real; y operaciones específicas, mismas que nos ayudan a simplificar procesos y a poder realizar procesos que de otra manera serían muy complicados o muy ineficientes.

Aprendí tanto a estar del lado implementador, pudiendo definir por medio de un lenguaje de programación las operaciones elementales de una cola, como a ponerme del lado usuario, en este último caso, limitándome a emplear las operaciones propias del TAD cola, sin modificarlas o volverme a preocupar por la implementación del TAD en absoluto.

Finalmente, cada práctica me ayuda a comprender la infinidad de soluciones posibles que pueden darse a un problema, y lo importante que son las estructuras de datos, pues aunque existan muchas soluciones, considero que una que tenga estructuras de datos bien aplicadas es inherentemente eficiente.

Mendieta Torres Alfonso Ulises

Las colas se viven dia a dia, incluso también para la computación, se trata de una estructura de datos útil para ciertas simulaciones y esencial para organizar los procesos del sistema operativo, con las colas de prioridad se puede hacer un ordenamiento por jerarquía.

Bibliografía

[1] E. Quevedo, R. López y A. Asencio, Universidad de Valladolid Departamento de informática Campus de Segovia, [En línea]. Available: http://www.iuma.ulpgc.es/users/jmiranda/docencia/programacion/Tema4_ne.pdf. [Último acceso: 05 Octubre 2017].

[2] A. D. Roldan, SIDECI, [En línea]. Available: http://sedici.unlp.edu.ar/handle/10915/33386. [Último acceso: 05 Octubre 2017].

[3] I. Y. Villalobos, EcuRed, [En línea]. Available: https://www.ecured.cu/Cola_(Estructura_de_datos). [Último acceso: 05 Octubre 2017].

[4] E. . A. Franco Martinez. [En línea]. Available: http://www.eafranco.com/docencia/estructurasdedatos/files/03/Tema03.pdf. [Último acceso: 05 Octubre 2017].