QNAP2: Resumen de usuario

ÍNDICE

1. Introducción

Es un lenguaje de modelización orientado a la construcción y tratamiento de modelos de redes de colas.

Es un lenguaje con una estructura similar al PASCAL, con algún elemento diferenciador. Esto implica que la ejecución de sentencias ira comprendido entre un BEGIN y un END. Además cada sentencia ira finalizada por el carácter ';'.

Proporciona como elementos de resolución:

- Simulador de eventos discretos (Simulación).
- Métodos analíticos exactos basados en la teoría de redes de colas:
  - Algoritmos de convolución.
  - Análisis del valor medio.
- Métodos analíticos aproximados basados en la teoría de redes de colas.
- Métodos analíticos basados en el tratamiento de cadenas de Markov.

Aquí veremos superficialmente la estructura del lenguaje, sin entrar a fondo en todas sus complejidades. Para más información deberá consultarse los manuales de usuario y de referencia.

2. Características del Lenguaje.

Palabras reservadas.- Las siguientes palabras están reservadas y no podrán ser utilizadas como nombres de variables.

ALL	AND	ANY
BEGIN	DO	ELSE
END	FALSE	FOR
FORWARD	GENERIC	GOTO
IF	IN	IS
NIL	NOT	OBJECT
OR	REF	REPEAT
STEP	THEN	TRUE
UNTIL	VAR	WATCHED
WHILE	WITH	

Identificadores.- Se admiten 8 caracteres comenzando por letra y no palabra reservada.

Números.- Las cantidades numéricas podrán ser tanto enteros como reales.

String.- Se considera un string cualquier cadena de caracteres entre comillas dobles `"Cadena de ejemplo"`. La representación del carácter comilla doble es: `" "" "`

Comentario.- Se considera comentario todo lo que va a continuación del símbolo **&** hasta el fin de línea.

Separadores.- Se consideran separadores los caracteres blanco y retorno de carro.

2.1 Expresiones

Están formadas por constantes, variables o llamadas a funciones conectadas por operadores lógicos; aritméticos o por paréntesis.

Las expresiones se resuelven de izquierda a derecha dentro de los paréntesis, de acuerdo a las siguientes reglas de prioridad.

- Negación lógica (**NOT**): más alta prioridad.
- Operador exponencial. (**)
- Operadores de tipo (**::**, **IS**, **IN**)
- Operadores de multiplicación (*, /, **AND**)
- Operadores de adición (+, -, **OR**)
- Operadores relaciónales (=, <>, <, >,<=, >=)

2.2 Sentencias

Pueden ser simples o compuestas, es decir, un grupo de sentencias simples entre un BEGIN y un END.

Se admiten:

- Llamadas a procedimientos y funciones.
- Sentencia GOTO

  ```
  GOTO etiqueta;
  ```

  Salto incondicional a determinada posición, la etiqueta se expresa como

  ```
  etiqueta:
  ```

- Sentencia IF

  ```
  IF expresión THEN sentencia [ELSE sentencia]
  ```

  Se admiten 10 niveles de anidación.

  Al igual que en PASCAL, antes del ELSE no se coloca el ';'

- Sentencia WHILE

  ```
  WHILE expresión DO sentencia;
  ```

  Hace que se repita la sentencia mientras la expresión booleana sea cierta. Si inicialmente la expresión es falsa no se ejecuta la sentencia.

- Sentencia FOR

  ```
  FOR variable:=lista DO sentencia;
  ```

La ejecución de una sentencia FOR hace que se repitan las sentencias que siguen al DO de acuerdo con los valores de la variable de control.

Como sentencias de control están permitidas todas las de listas , son:

STEP, UNTIL, REPEAT, WITH.

La expresión de lista se evalúa al entrar en el lazo FOR.

Ejemplos:

```
FOR N:=1 STEP 1 UNTIL 100  DO...
FOR N:=100 STEP -1 UNTIL 1 DO...
FOR Q:=A,B,C DO...
FOR Q:=ALL QUEUE WITH  MTHRUPUT(QUEUE) > 0. DO...
```

- Sentencia WITH

```
WITH identificador de objeto DO sentencia.
```

Aporta una referencia explícita al objeto que está siendo manipulado.

2.3 Operaciones de Entrada/Salida

El lenguaje QNAP2 incorpora una serie de procedimientos predefinidos para realizar la entrada/salida.

- La entrada se hace con las funciones, GET y GETLN.

GETLN y GET funcionan igual, se diferencian en que GETLN salta a la siguiente línea después de leer, ignorando el resto de la línea.

GET; lee un dato de cada vez.

```
{GET/GETLN} ([fichero,], tipo [, longitud])
```

`fichero`: fichero de donde lee el dato, por defecto FSYSGET.

`tipo`: tipo de dato a leer: (INTEGER, REAL, BOOLEAN, STRING).

`longitud`: Número de caracteres a leer.

- La salida se hace con las funciones PRINT, WRITE y WRITELN.

PRINT y WRITELN saltan a la siguiente línea después de escribir. En cambio, WRITE permanece en la misma línea.

```
PRINT/WRITE/WRITELN [([fichero,] {dato [,formato]} [,...])]
```

`fichero`: especifica la salida, por defecto es FSYSPRINT=FSYSOUTPUT.

- Procedimientos de control.

Los siguientes comandos permiten controlar los ficheros de datos.

FILASSIGN se usa para asignar un nombre de fichero a un fichero.

```
FILASSIGN (file, nombre);
```

Ejemplo:

```
FILASSIGN (FSYSGET, ”nombre”);
```

OPEN y CLOSE usadas para abrir y cerrar ficheros.

```
OPEN (file [, mode]);
CLOSE (file [, mode]);
```

Donde `mode` puede tomar los siguientes valores con los significados:

1 Abrir un fichero para leer, ya existe.

2 Crear un fichero nuevo.

3 Abrir un fichero ya existente para escribir.

3. Lenguaje de Control

Los programas escritos en lenguaje QNAP2 tienen una estructura definida por unos comandos de control. Los comandos de control vienen expresados entre / / sin espacios de separación entre estos símbolos y el comando.

Los comandos de control posibles son:

/DECLARE/	Declaración de variables e identificadores.
/STATION/	Descripción de características de una estación.
/CONTROL/	Especificación de parámetros de control.
/EXEC/	Bloque de instrucciones de ejecución.
/TERMINAL/	Ejecución interactiva.
/REBOOT/	Relanzamiento de la ejecución algorítmica después de un RESTORE
/RESTART/	Introducción de un nuevo modelo.
/END/	Final del fichero QNAP2.

Cada uno de estos parámetros de control puede aparecer más de una vez dentro de la secuencia del programa.

3.1 Fases de evaluación de las sentencias

- Tiempo de compilación: Compilación de los comandos QNAP2, finaliza al encontrar el EXEC.
- Tiempo de ejecución: Ejecución de las sentencias del EXEC.
- Tiempo de solución: Resolución del modelo.

3.2 Comandos de control

Comenzaremos haciendo un breve resumen de las sentencias de control con sus parámetros más importantes. Para una información más completa debe acudirse al manual de usuario y manual de referencia.

3.2.1 /DECLARE/

Bajo este comando se declaran los datos y variables que forman parte del programa.

Los datos que forman parte del programa pueden ser de varios tipos.

3.2.1.1 Datos de tipo escalar:

Son:

Tipo Dato	Ejemplo de declaración
Enteros	`INTEGER  L1=10;`
Reales	`REAL  R=5.2E4;`
Booleanos	`BOOLEAN  Lbs;`
Strings	`STRING  D;`

3.2.1.2 Datos de tipo array

Consiste en una colección de elementos del mismo tipo. No están permitidos los arrays de arrays.

`REAL C(5);` Array de cinco elementos de tipo real.

`INTEGER D(2,3);` Array de seis elementos de tipo entero, el índice de más a la derecha varía más rápido que el de más a la izquierda.

3.2.1.3 Datos tipo Referencia

Referencia a objetos. Es similar a un puntero de otro tipo de lenguajes.

```
REF tipo_de_objeto  nombre;
```

3.2.1.4 Datos de tipo Objeto

Un objeto es un tipo de dato que puede definir el usuario, puede ser algo como lo siguiente.

```
/DECLARE/
     OBJECT CENTRO (PROB, NLINES);
          & declaración de sus atributos.
          INTEGER NLINES;      & Número de líneas
          REAL PROB (NLINES); & Probabilidades
          REAL TLINE;          & Tiempo de servicio promedio
     END;
```

Más adelante podría escribirse:

```
/DECLARE/
     CENTRO ((0.5, 0.2, 0.2, 0.1), 4);
```

En general sirve para definir nuevos tipos de datos. La expresión general para definir un objeto es:

```
[tipo del objeto] OBJECT identificador  [(parámetros)];
     declaración de atributos;
END;
```

Ejemplo.-

```
CUSTOMER OBJECT PACKET;
     INTEGER BYTES;
     STRING HEADER;
     REF CLASS CIRCUIT;
END;
```

3.2.1.5 Etiquetas

Dentro del comando /DECLARE/ aparte de los tipos de datos vistos, también es posible definir otra serie de elementos. Entre ellos tenemos las etiquetas que son nombres empleados para marcar el lugar a donde se dirigen los saltos, se declaran.

```
LABEL  identificador de etiqueta usado posteriormente
```

3.2.1.6 Procedimientos.

Son subrutinas de código introducido por el usuario. Existen tres formas de declararlos, de las cuales la más normal es:

```
PROCEDURE identificador [(parámetros [,...])];
[declaración de parámetros y variables locales];
BEGIN
     Sentencias;
END;
```

Los parámetros pueden pasarse por valor, en cuyo caso cualquier modificación dentro del procedimiento no los afectará, o por referencia (Se les antepone la palabra [VAR]) en cuyo caso si son afectados por las modificaciones realizadas dentro del procedimiento.

El procedimiento puede llamarse desde cualquier parte ejecutable del programa.

Ejemplo.-

```
/DECLARE/
     QUEUE Q;
     REAL SUM 0.0; T=3.0;
     PROCEDURE DELAY (DELTA, D);
     VAR REAL DELTA
     REAL D
     BEGIN
          D:=2*D;
          CST(D);
```

```
            DELTA:=DELTA D;
      END;

/STATION/
      NAME=Q;
      SERVICE=DELAY (SUM,T);
      PRINT (SUM,T); &6.0, 3.0
```

3.2.1.7 Funciones

Las funciones son similares a los procedimientos.

Se diferencian:

- La palabra reservada es FUNCTION en lugar de PROCEDURE.
- La función va precedida del tipo de resultado que devuelve.
- Dentro de la función, el valor devuelto debe asignarse a la variable predefinida RESULT.

Ejemplo.-

```
/DECLARE/
      BOOLEAN FUNCTION ISREADY (RQ);
      REF QUEUE RQ;      & Variable parámetro
      REF CUSTOMER RC;   & Variable local.
      BEGIN
            RC:=RQ.FIRST;
            IF(RC=NIL)
            THEN RESULT:=TRUE
            ELSE RESULT:=NOT RC.BLOCKED;
      END;
```

3.2.2 Objetos predefinidos

En el lenguaje QNAP2, existen una serie de objetos predeclarados que son:

QUEUE .-.	Cola.
CUSTOMER .-	Cliente.
CLASS .-	Clase.
FLAG .-	Flag.

Otros objetos también definidos pero de uso más específico son:

TIMER .-	Temporizador.
EXCEPTION .-	Código a ejecutar cuando ocurra alguna excepción en

sistemas UNIX.

FILE .- Fichero.

Cada uno de estos objetos tiene a su vez, declarados una serie de parámetros o campos que es interesante conocer.

Además, es posible añadir atributos a los objetos de los primeros cuatro tipos, para ello, sólo es necesario declararlos. Por ejemplo:

```
QUEUE INTEGER H;
```

Con la declaración anterior, añadimos al tipo cola un nuevo campo entero H. Todos los objetos de tipo cola declarados a partir de este momento dispondrán además de sus campos predefinidos del atributo H.

Los objetos pueden formar parte a su vez de otros objetos. Por ejemplo:

```
OBJECT NODE (N,T);
      INTEGER N;
      QUEUE CPU;
      QUEUE DISK (N);
      REAL T;
END;
```

Vamos a ver seguidamente los atributos más importantes de los cuatro primeros tipos de objetos. Para una descripción más completa de todos los atributos, habrá que acudir tanto al manual de usuario como al manual de referencia.

3.2.2.1 Atributos del tipo Cola.

Los principales atributos del tipo cola son:

CAPACITY .- Variable que informa sobre la capacidad de la cola.

```
[queue.] capacity [(clase)];
```

Devuelve el valor (-1) si la capacidad de la cola no está limitada (infinita).

MULT .- Variable que informa sobre la multiplicidad o número de servidores en la cola, devuelve el valor (-1) en el caso de que tengamos un servidor infinito.

Los siguientes atributos sólo son accesibles cuando el método de funcionamiento elegido para el QNAP2 sea la simulación.

FIRST .- Referencia al primer cliente en una cola.

LAST .- Referencia al último cliente de una cola.

NB .- Número de clientes en la cola (esperando y recibiendo servicio).

NBIN .- Número de clientes que han entrado en la cola hasta el momento.

NBINSERV .- Número de clientes que están recibiendo servicio.

NBOUT .- Número de clientes que han salido de la cola hasta el momento.

VALUE .- Número de servidores libres.

3.2.2.2 Atributos del tipo cliente

Seguidamente se describen los atributos más importantes del tipo cliente, la mayoría de los cuales sólo son accesibles durante la simulación. Recordar, que el tipo cliente es el que recibe servicio en las diferentes colas.

ACTIVETIME .- Tiempo en el que el cliente está activo.

BLOCKED .- Devuelve el estado del cliente, bloqueado o no. Simulación.

BLOCKTIME .- Devuelve tiempo que el cliente ha permanecido bloqueado. Simulación.

CHANGEDATE .- Devuelve la fecha de la última modificación del estado de un cliente; activo, esperando, bloqueado.

ENTERDATE .- Fecha de llegada a la estación actual. Simulación.

RESPTIME .- Devuelve el tiempo empleado por un cliente en una estación.

STARTED .- Devuelve la fecha de comienzo de servicio a un cliente.

CCLASS .- Referencia a la clase del cliente. Simulación.

CPRIOR .- Devuelve el nivel de prioridad del cliente. Simulación.

CQUEUE .- Referencia la estación que contiene al cliente. Simulación.

FATHER .- Referencia al padre del cliente.

NEXT .- Referencia al siguiente cliente en la cola.

PREVIOUS .- Referencia al cliente previo en la cola.

SON .- Referencia al último cliente creado por el cliente.

3.2.2.3 Atributos del tipo clase

No se especifican aquí los atributos del tipo clase, al no emplearse. Para obtener información sobre los atributos de este tipo dirigirse al manual de usuario o manual de referencia.

3.2.2.4 Atributos del tipo flag

Para el tipo FLAG los valores predefinidos más importantes son:

STATE .-Variable booleana que toma el valor TRUE si el flag está en SET o FALSE si el flag está en RESET.

LIST .- Referencia del último cliente esperando por el flag.

3.2.3 /STATION/

Este comando de control, permite describir una o varias estaciones de servicio en el modelo. Puede usarse para la definición inicial de una lista de estaciones así como las sucesivas alteraciones de sus características. Admite varios parámetros, de los cuales veremos los más importantes:

NAME .- Debe ser el primer parámetro del comando /STATION/. Define tantas estaciones como colas haya en la lista de colas asociada a este parámetro y cada una de ellas debe estar identificada en el comando de declaración.

```
NAME={lista de colas};
```

Ejemplos.-

```
/DECLARE/ QUEUE CPU, DM(10);
/STATION/ NAME=CPU;


/STATION/
     NAME=DK(1 STEP 1 UNTIL 5);
     TRANSIT=CPU;
     SERVICE=EXP(0.1);
```

TYPE .- Parámetro que define el tipo de estación. Se utiliza:

```
TYPE=tipo de estación;
```

Puede tomar los siguientes valores:

- SERVER estación de tipo servidor; a su vez puede ser:

SINGLE	1 servidor.
MULTIPLE (n)	n servidores.
INFINITE	infinitos servidores.

- RESOURCE define una estación de tipo recurso; puede ser:

SINGLE	Recurso no compartido.
MULTIPLE(n)	El recurso puede ser compartido por n clientes.
INFINITE	El recurso está siempre disponible.

- SEMAPHORE estación de tipo semáforo, a su vez pueden ser:

SINGLE	valor inicial=1.
MULTIPLE (n)	valor inicial del contador (n>=0).

- SOURCE define una estación tipo fuente, es decir, generadora de clientes.
- MULTIPLE (n) usado solo considera la estación de tipo SERVER y n servidores.

Si en la definición de una estación se omite el parámetro TYPE, éste toma por defecto los valores SERVER, SINGLE.

SCHED .- Este parámetro describe la gestión de los clientes en la cola de la estación, algunos de los valores más frecuentes que puede tomar son los siguientes valores:

- FIFO .- Los clientes serán servidos según el orden de llegada. Valor por defecto.
- LIFO ó FILO .- Los clientes se sirven en el orden inverso al de llegada, es decir, el último en llegar será el primero en servirse.

- PRIOR .- Los clientes se ordenan en la cola de acuerdo a su prioridad relativa. La más alta prioridad primero, prioridades iguales según sea FIFO ó LIFO.
- PREMPT .- El cliente que está siendo servido es adelantado por la llegada de un cliente de más prioridad. El cliente adelantado continuará siendo servido cuando le vuelve a tocar el turno de servicio.
- QUANTUM .- Asignación de quanto. Cuando se asigna un servidor a los clientes durante una longitud fija de tiempo, definida por un número real asociado

```
SCHED= QUANTUM(0.2);
```

- PS .- Procesador compartido. El tiempo de servicio se reparte por igual entre todas las tareas a servir.

PRIOR .- Fija el nivel de prioridad de un cliente que entra a la cola. Puede ser diferente para cada clase de cliente. Sin parámetros se aplica a todas las clases. Se usa:

```
PRIOR [(lista de clases)]=entero;
```

Cuanto mayor sea el valor entero, mayor será la prioridad asignada.

Ejemplo.-

```
/DECLARE/  QUEUE UC, UK;
           INTEGER PX;
           CLASS X,Y,Z;
/STATION/  NAME=UC;
           PRIOR (X,Y)=PX; & Para las clase X e Y
           PRIOR=2;        & Para el resto de clases
/STATION/  NAME=UK;
           SCHED=PRIOR, PREEMPT;
           PRIOR (Y)=1;
           PRIOR (Z)=3;
           & La clase X mantiene la prioridad que
tuviera.
```

QUANTUM .- Este parámetro especifica la asignación del cuanto en cada estación. Este valor se usa sólo si con el parámetro SCHED se ha elegido el valor QUANTUM. Se usa:

```
QUANTUM [(lista de clases)]=real;
```

Cuando se especifica una clase, el valor del quantum se aplica a esa clase. En caso de no especificar ninguna clase, el valor del quanto se aplica a todas las clases.

Ejemplo.-

```
/DECLARE/  QUEUE UC;
```

```
            REAL QT;
            CLASS X,Y,Z;
/STATION/   NAME=UC;
            SCHED=QUANTUM(0.3);
            QUANTUM (X)=QT;
```

INIT .- Este parámetro especifica el número de clientes de cada clase que están en la estación considerada en el instante inicial. Si no se especifica la clase, se aplica por igual a todas las clases.

Las estaciones de tipo RESOURCE y SEMAPHORE no deben contener clientes en el instante inicial.

Se usa el parámetro:

```
INIT [(lista de clases)]=entero;
```

Ejemplo.-

```
/DECLARE/   QUEUE A,B;
            CLASS X,Y;
            INTEGER N;
/STATION/   NAME=A;
            INIT (X)=2;
            INIT (Y)=1;
/STATION/   NAME=B;
            INIT=N;
```

SERVICE .- Especifica el servicio requerido por los clientes que están en las estaciones consideradas. Puede ser distinto para cada clase. Este parámetro se ignora en las estaciones de tipo RESOURCE y SEMAPHORE.

Un servicio se compone de peticiones de trabajo y de operaciones de manipulación de objetos. Las peticiones y los objetos están relacionados por todo tipo de instrucciones algorítmicas y funciones para describir su comportamiento en una estación de servicio. Se usa:

```
SERVICE [(lista de clases)]=sentencia;
```

Este parámetro tiene un funcionamiento diferente según se trabaje en resolución analítica o simulación. En resolución analítica el servicio recibido en una estación ha de estar restringido a una única sentencia. En el caso de la simulación el servicio puede estar formado por múltiples sentencias comprendidas entre un BEGIN y un END.

Existen servicios predefinidos y limitados a una única sentencia. Algunos de los más frecuentes se verán a continuación. Para una completa descripción de todos los tipos de servicio habrá de referirse al manual de usuario y manual de referencia.

CST .- Constante.

```
CST(real);
```

EXP .- Exponencial. El valor es la media de la distribución.

```
EXP(real);
```

HEXP .- Hiperexponencial. Los parámetros son la media y el resultado de dividir la varianza entre la media al cuadrado.

```
HEXP(real 1, real 2);
```

ERLANG .- Erlang.

```
ERLANG(real, entero);
```

UNIFORM .- Distribución uniforme. Los parámetros son los extremos del intervalo de variación.

```
UNIFORM(real 1, real 2);
```

RINT .- Distribución uniforme con valores enteros.

```
RINT (entero 1, entero 2);
```

COX .- Ley de Cox.

NORMA L.- Normal. Los parámetros son la media y la desviación estándar.

```
NORMAL (real 1, real 2);
```

TRANSIT .- Este parámetro describe las reglas de routing de los clientes cuando completan el servicio. El cliente se mueve de una estación a otra de acuerdo a unas probabilidades. Las probabilidades de transición pueden venir dadas para cada estación de destino y para cada clase por un grupo de tres elementos. Cada grupo consta de:

- Una o varias estaciones de destino.
- Una o varias clases de destino.
- Una o varias probabilidades de transición. (relativa ó absoluta).

En el caso de sistemas abiertos existe la cola predefinida OUT, donde son enviados todos los clientes que abandonan el sistema (Los clientes son destruidos).

Una estación de tipo fuente crea clientes de varias clases que pueden rotar por diferentes estaciones según probabilidades dadas.

Las probabilidades son absolutas cuando su suma es 1.0 en otro caso el QNAP2 las toma como relativas y las normaliza convirtiéndolas en absolutas.

El formato general de la orden es:

```
TRANSIT[(lista de clases)] {lista de colas[, lista de
clases], probabilidad}  [...];
```

Ejemplo.-

```
/DECLARE/  QUEUE A,B,C;
/STATION/  NAME=A;
           SERVICE=EXP (5);
           TRANSIT=A, 0.5, B, 0.3, C; & Prob. absolutas.
/DECLARE/  QUEUE A, B, C;
```

```
            CLASS X, Y, Z;
            REAL PA, PB;
/STATION/   NAME=A;
            SERVICE=EXP(5);
            TRANSIT=B;
            TRANSIT(X)=A, PA, B, Y, PB, C, Z;
            TRANSIT(Y)=A, Z, 1, C, Y, 2;
/DECLARE/   QUEUE CPU, DISK(3);
            REAL PROB(3);
/STATION/   NAME=CPU;
            TRANSIT=DISK (1 STEP 1 UNTIL 3), PROB, OUT;
```

CAPACITY .- Especifica el número máximo de clientes permitidos en la estación. Esta capacidad puede ser diferente para cada clase. Si se especifica una clase, el número asociado a este parámetro se refiere al número máximo de clientes de esa clase. En caso de no especificar lista de clases la capacidad se refiere a toda la estación. La capacidad por defecto es infinita.

La capacidad incluye al cliente que está siendo servido, así como los que esperan en la cola. Se usa:

```
CAPACITY[(lista de clases)]=entero;
```

COPY .- Este parámetro sirve para definir una o varias estaciones idénticas copiando los parámetros de una estación a otra. Se usa

```
COPY=cola;
```

Ejemplo.-

```
/DECLARE/   QUEUE DK0, DK1, DK2, CPU;
            CLASS X, Y;
/STATION/   NAME = DK0;
            TRANSIT = CPU;
            SERVICE = EXP(10);
            SCHED = LIFO, PREEMPT;
            PRIOR(X) = 2;
            PRIOR(Y) = 1;
            CAPACITY = 10;
/STATION/   NAME = DK1;
            COPY = DK0;
```

3.2.4 /CONTROL/

El comando /CONTROL/ sirve para variar el control de la ejecución del programa, permite alterar los valores por defecto. Seguidamente se verán los parámetros más frecuentes que acepta este comando, para una información más completa dirigirse al manual de usuario y manual de referencia.

CLASS .- Especifica para qué clases se obtienen valores estadísticos. Se usa:

```
CLASS=lista de colas;
```

Ejemplo.-

```
CLASS=ALL QUEUE;          & Para todas las colas y
clases
CLASS=NIL;           & Para ninguna (defecto)
```

MARGINAL .- Establece probabilidades marginales para una cola, por defecto calcula la probabilidad hasta el nivel 5. Es decir, calcula las probabilidades de que haya 0 clientes en la cola, 1 cliente, 2, ... así hasta el nivel especificado, por defecto 5 clientes en el sistema. Se usa:

```
MARGINAL={sublista de colas [, entero]} [, ...];
```

OPTION .- Permite realizar el control de la impresión de resultados posterior a la ejecución del programa. El programa QNAP2 después de ejecutar el modelo genera un fichero de resultados, este fichero puede estar formado por diferentes textos según se establece con este parámetro. Las opciones que aparecen subrayas son los valores por defecto. Las opciones posibles son:

```
OPCIÓN     SOURCE;    & Se muestra el programa fuente.
           NSOURCE;
           RESULT;    & Aparecen los resultados finales.
           NRESULT;
           TRACE;     & Aparecen los valores de traza.
           NTRACE;
           DEBUG;     &  Resultados  del  depurador  de
líneas.
           NDEBUG;
           VERIF;     & Valores de pruebas en ejecución.
           NVERIF;
```

UNIT .- Parámetro que permite especificar la asignación de unidades de entrada salida. Permite redireccionar las entradas y salidas estándar cuando es necesario. Se usa:

```
UNIT=unidad (fichero);
```

Las diferentes unidades pueden ser:

```
OUTPUT     Salida estándar.
PRINT      Salida de impresora estándar.
INPUT      Entrada estándar.
GET        Entrada de donde se leen datos.
LIBR       Entrada de librerías.
TRACE      Salida de resultados de traza.
```

Ejemplo.-

```
FILEASSIGN (F, "nombre.dat");
OPEN (F,1);
```

```
UNIT=GET(F);
```

TSTART .- Define el tiempo en el que deben comenzar las medidas en el modelo de simulación para obtener estadísticas. Por defecto se toma el valor 0. Se usa:

```
TSTART=REAL;
```

TMAX .- Este parámetro define la duración máxima de una simulación expresada en unidades de tiempo del modelo. La duración puede venir definida por una cantidad o una expresión. Por defecto TMAX toma el valor TMAX=0. Se usa:

```
TMAX=real;
```

RANDOM .- Generador de números aleatorios. El generador se inicializa al comienzo de cada ejecución. El valor inicial o semilla del generador de números aleatorios puede cambiarse con RANDOM, por defecto este valor es 413. El valor puede ser entero constante o expresión. Se usa:

```
RANDOM=entero;
```

ACCURACY .- Sirve para fijar el intervalo de confianza de trabajo. Se usa:

```
ACCURACY={sublista colas [, sublista de clases]} [,
...];
```

Ejemplos.-

```
ACCURACY=NIL;
```

Es el defecto, no son necesarios intervalos de confianza.

```
ACCURACY=ALL QUEUE, ALL CLASS;
```

Intervalos de confianza para todas las colas y clases.

TEST .- Especifica una sentencia que se ejecutará durante la simulación al final de cada intervalo dado por el parámetro PERIOD. Puede ser una sentencia o un bloque algorítmico. Se usa:

```
TEST=sentencia;
```

El defecto es TEST=;

PERIOD .- Determina cada cuanto tiempo se ejecuta la activación de la secuencia de TEST. Se usa:

```
PERIOD=real;
```

Ejemplo.-

```
/DECLARE/  QUEUE A, B, C;
/CONTROL/  PERIOD=200;
           TMAX=2000.;
           TEST=IF A.NBOUT>500 THEN STOP;
/EXEC/     SIMUL;
```

TRACE .- Controla el tiempo de comienzo y final de traza durante una simulación. El primer valor es el tiempo de comienzo, y el segundo es el de finalización. Por defecto este valor coincide con TMAX. Se usa:

```
TRACE=real [, real]  [, string];
```

El valor de traza por defecto es TRACE=0.0, 0.0; No se produce traza. El valor del string indica el formato de salida de los datos, en 132 columnas o 80 columnas ("L132" y "L80"). Por defecto 132 columnas.

NMAX .- Incrementa el número máximo de clases disponibles en un modelo. Este parámetro debe especificarse antes de la primera declaración de una cola o de una clase y no se puede cambiar posteriormente. Se usa:

```
NMAX=entero;
```

El valor por defecto de NMAX es de 20.

ENTRY .- Establece una serie de sentencias a ejecutar antes de iniciar el análisis del modelo. Durante este proceso, se pueden inicializar variables. Se usa:

```
ENTRY=sentencia;
```

EXIT .- Describe una secuencia de instrucciones que se ejecutarán una vez resuelto el modelo. Se usa:

```
EXIT=sentencia;
```

Ejemplo.-

```
/DECLARE/  QUEUE A, B, C;
           INTEGER MA, MB, MC;
           REAL PA, PB, PC;
           REAL LRESPONS(20);
/STATION/  NAME=A;
           TRANSIT=B, PB, C, PC, A;
/CONTROL/  ENTRY=BEGIN
                     PRINT("MA, MB, MC", MA, MB, MC);
                     PB:=MB; PB:=PB/(MA+MB+MC);
                     PC:=MC; PC:=PC/(MA+MB+MC);
                END;
           EXIT=BEGIN
                     LRESPONS(N):=MRESPONS(C);
                     PRINT (MRESPONS (A)+MRESPONS(B));
                     PRINT ("FIN DEL ANALISIS");
                END;
/EXEC/  BEGIN
           MA:=10;
```

```
            MB:=20;
            MC:=30;
            SOLVE;
            END;
/EXEC/  FOR N:=STEP 1 UNTIL 20 DO SOLVE;
```

ALIAS .- Se usa para añadir nuevos nombres a palabras claves ya existentes. El primer identificador es el alias dado al segundo identificador. Se restringe a las palabras reservadas.

```
ALIAS={(identificador, identificador)}, [, ...];
```

Ejemplo.-

```
/CONTROL/  ALIAS=(NOMBRE, NAME), (GESTION, SCHED);
```

No están permitidos los alias de alias.

3.2.5 /EXEC/

A partir de este comando se introduce la parte ejecutable del modelo. **Si la parte ejecutable está formada por más de una sentencia, el comando /EXEC/ deberá comenzarse con un BEGIN y finalizarse con un END.**

Para la resolución del modelo se emplean diferentes métodos, son:

- SOLVE [("Keyword")]; .- Resolución analítica.

  Keyword especifica el método, si se omite , se elige automáticamente el más adecuado.

- MARKOV; .- Resolución de cadenas de Markov.
- SIMUL; .- Simulación de eventos discretos.

3.3 Utilidades

Dentro de este punto vamos a describir algunas funciones ya implementadas en el lenguaje QNAP2 que nos permiten la realización de operaciones complejas, veremos principalmente las herramientas matemáticas, la manipulación de objetos, los procesos de sincronización y la obtención de resultados. Para un tratamiento más extenso de estos puntos así como de otros no cubiertos aquí deben remitirse al manual de usuario y manual de referencia del lenguaje QNAP2.

3.3.1 Herramientas Matemáticas

A continuación se presenta un resumen de las herramientas matemáticas que se pueden utilizar dentro del lenguaje QNAP2.

- **ABS** Devuelve el valor absoluto de un entero o un real. Se llama:

```
ABS(entero/real);
```

- **ACOS** Función trigonométrica arcocoseno, su dominio de definición es entre [-1, 1]. Se llama:

  ```
  ACOS(real);
  ```

- **ASIN** Función trigonométrica arcoseno, su dominio de definición es entre [-1, 1]. Se llama:

  ```
  ASIN(real);
  ```

- **ATAN** Función trigonométrica arcotangente, el valor devuelto se expresa en radianes. Se llama:

  ```
  ATAN(real);
  ```

- **COS** Función trigonométrica coseno, el argumento debe dársele en radianes. Se llama:

  ```
  COS(real);
  ```

- **DISCRETE** Representación de un número aleatorio entre una lista de elementos homogéneos de acuerdo con una tabla de probabilidades. Se llama:

  ```
  DISCRETE(list/array, list/tabla reales [,entero]);
  ```

  Primera lista.- Lista de valores a elegir.
  Segunda lista.- Lista de probabilidades. Si la suma de probabilidades es distinta de 1, entonces los valores se normalizan a 1.
  El tercer parámetro, opcional, representa el número de valores a tener en cuenta, por defecto es la dimensión del primer argumento.

- **EXPO** Función exponencial. Se llama:

  ```
  EXPO(real);
  ```

- **FIX** Devuelve la parte entera de un número real, pero en formato real. Se llama:

  ```
  FIX(real);
  ```

- **INTREAL** Devuelve la parte entera de un número real, pero en formato entero. Se llama:

  ```
  INTREAL(real);
  ```

- **INTROUND** Devuelve el entero más próximo a un número real. Se llama:

  ```
  INTROUND(real);
  ```

- **LOG** Función logaritmo natural o neperiano. Se llama:

  ```
  LOG(real);
  ```

- **LOG10** Función en base 10. Se llama:

  ```
  LOG10(real);
  ```

- **MAX** Devuelve el máximo entre varios valores. Un entero si todos son enteros, real si uno o más son reales. Se llama:

  ```
  MAX(entero1/real1, entero2/real2, ...);
  ```

- **MIN** Devuelve el mínimo entre varios valores. Un entero si todos son

enteros, real si uno o más son reales. Se llama:

```
MIN(entero1/real1, entero2/real2, ...);
```

- **MOD** Módulo entre dos enteros. Devuelve un valor entero resto de la división entre los argumentos. Se llama:

  ```
  MOD(entero1, entero2);
  ```

- **REALINT** Esta función devuelve el valor real del entero que se le pasa como argumento. Se llama:

  ```
  REALINT(entero);
  ```

- **SIN** Función trigonométrica seno, el argumento debe dársele en radianes. Se llama:

  ```
  SIN(real);
  ```

- **SQRT** Función raíz cuadrada, el argumento debe ser positivo o nulo. Se llama:

  ```
  SQRT(real);
  ```

- **TAN** Función trigonométrica tangente, el argumento debe dársele en radianes. Se llama:

  ```
  TAN(real);
  ```

3.3.2 Manejo de Objetos

Existen funciones que permiten manipular objetos, algunas que pueden ser útiles y que utilizaremos son:

- **NEW** Creación dinámica de un nuevo elemento. Se utiliza:

  ```
  NEW (type [, lista de parámetros]);
  ```

  Devuelve una referencia al nuevo objeto creado.

  Type es el tipo del objeto creado, los siguientes son parámetros del objeto si tiene.

  Por defecto el hijo tiene la misma clase y prioridad que su padre.

  Ejemplo de creación de un cliente.

  ```
  /DECLARE/ REF C;
  /STATION/ SERVICE=BEGIN
                  .....
          NEW (CUSTOMER); & crea un nuevo cliente.
  ```

  Un cliente creado de esta forma no está en ninguna cola, debe ser transferido a una estación mediante un TRANSIT

  Asociado a lo anterior existen dos atributos ya vistos de cliente:

  FATHER .- Devuelve la dirección del padre del cliente.

  SON .- Devuelve la dirección del cliente hijo.

- **DISPOSE** Borrado de Objetos. Se usa:

```
DISPOSE (ref. objeto);
```

Permite borrar objetos creados dinámicamente con la función NEW.

Un objeto creado con /DECLARE/ no puede borrarse. Los objetos del tipo escalar no pueden borrarse. Los clientes se borran con una transición al dispositivo **OUT**.

- **REFSON** Accede a un hijo de un cliente. Se utiliza:

  ```
  REFSON (cliente, entero);
  ```

  Devuelve la referencia al cliente hijo definido por el primer argumento; el número de hijo buscado viene dado por el segundo argumento. Los hijos se numeran de más viejo (1) a más joven.

- **TYPENAME** Accede al tipo de un objeto. Se utiliza:

  ```
  TYPENAME (ref. objeto[, entero]);
  ```

  Resuelve una cadena que es el nombre de tipo del objeto especificado.

3.3.3 Procesos de Sincronización

Algunos de los procedimientos que veremos son:

- **BLOCK** Bloqueo y desbloqueo de la estación. Se usa:

  UNBLOCK `BLOCK (cola1, cola2,....);` Bloquea estaciones.

  ISBLOCK `UNBLOCK (cola1, cola2,....);` Desbloquea estaciones.

  `ISBLOCK (cola);` Nos permite comprobar el estado de la estación. Devuelve TRUE si está bloqueada y FALSE en otro caso.

  El tiempo de bloqueo se le añadirá sólo al cliente que recibe servicio cuando se produce el bloqueo.

- **FREE** Desbloqueo de un cliente. Se utiliza:

  ```
  FREE (customer [, cola/flag]);
  ```

  Este procedimiento cancela las condiciones de espera del cliente referenciado por el primer argumento.

- **MOVE** Transición del primer cliente de una cola a otra. Se utiliza:

  ```
  MOVE(cola1, cola2);
  ```

  Sólo se puede utilizar en simulación. No es posible enviar un cliente bloqueado fuera de la red.

- **TRANSIT** Transición de un cliente a una estación. Se utiliza:

  ```
  TRANSIT([cliente], cola, [clase] [, entero]);
  ```

  Mueve el cliente actual o el especificado, de la estación actual a la indicada por el segundo parámetro. Puede producirse un cambio de clase si el cliente procede de una estación SOURCE. El entero representa el nivel de prioridad asignado.

Sólo se puede emplear en simulación. No es posible enviar un cliente bloqueado fuera de la red.

Este es el modo de manejar clientes en otros lugares del sistema o los clientes creados dinámicamente.

Ejemplo.-

```
C:=NEW(CUSTOMER);
TRANSIT(C, COLA1);
```

- **PRIOR** Actualiza el nivel de prioridad de un cliente. Se utiliza:

  ```
  PRIOR([customer, ] integer);
  ```

  Modifica la prioridad del cliente actual o del especificado. Sólo se puede emplear en simulación.

- **SET** **RESET** Establece un flag a estado a nivel alto o nivel bajo. Se utiliza:

  `SET (flag);` Coloca el flag en estado alto.

  `RESET (flag);` Coloca el flag en estado bajo.

  Sólo se puede emplear en simulación.

- **WAIT** **WAITAND** **WAITOR** Define las condiciones de espera por el estado de los flags. Se usa:

  ```
  WAIT ([cliente, ] flag);
  WAITAND ([cliente, ]flag1, flag2, ...);
  ```

  Ejecuta la operación AND entre flags, ha de esperar por todos.

  ```
  WAITOR ([cliente, ]flag1, flag2, ...);
  ```

  Ejecuta la operación OR entre flags, espera por el primero.

  Se permanece en estado de espera mientras los flags estén a nivel bajo. La condición de espera desaparece cuando los flags pasan a nivel alto. Sólo se pueden utilizar en simulación.

3.3.4 Funciones de presentación de resultados

Seguidamente se presenta un resumen de las funciones implementadas por el lenguaje que generan resultados estadísticos.

- **CBLOCKED** Devuelve valores reales, correspondientes al intervalo de confianza del tiempo de bloqueo. Se llama:

  ```
  CBLOCKED(cola [,clase]);
  ```

- **CBUSYPCT** Devuelve valores reales, correspondientes al intervalo de confianza para el porcentaje de ocupación de los servidores de la estación. Se llama:

  ```
  CBUSYPCT(cola [,clase]);
  ```

- **CCUSTNB** Devuelve valores reales, correspondientes al intervalo de confianza para el número medio de clientes en la estación. Se llama:

```
CCUSTNB(cola [,clase]);
```

- **CRESPONSE** Devuelve valores reales, correspondientes al intervalo de confianza para el tiempo medio de respuesta. Se llama:

  ```
  CRESPONSE(cola [,clase]);
  ```

- **CSERVICE** Devuelve valores reales, correspondientes al intervalo de confianza para el tiempo medio de servicio. Se llama:

  ```
  CSERVICE(cola [,clase]);
  ```

- **CUSTNB** Devuelve un entero que representa el número medio de clientes en el momento actual en una estación. Se llama:

  ```
  CUSTNB(cola [,clase]);
  ```

- **MAXCUSTNB** Esta función devuelve un entero que representa el número máximo de clientes en una estación. Se llama:

  ```
  MAXCUSTNB(cola [,clase]);
  ```

- **MBLOCKED** Devuelve un valor real que corresponde al tiempo medio de bloqueo de la estación. Se llama:

  ```
  MBLOCKED(cola [,clase]);
  ```

- **MBUSYPCT** Devuelve un valor real que corresponde al porcentaje medio de ocupación de la estación. El porcentaje de ocupación corresponde a la proporción de tiempo durante el cual el servidor ha sido solicitado. Se llama:

  ```
  MBUSYPCT(cola [,clase]);
  ```

- **MCUSTNB** Esta función devuelve un valor real correspondiente al número medio de clientes en la estación. Se llama:

  ```
  MCUSTNB(cola [,clase]);
  ```

- **MRESPONSE** Devuelve un valor real correspondiente al valor medio del tiempo de respuesta de la estación. El tiempo de respuesta representa el lapso entre la entrada en la cola y la salida después de recibir servicio. Se llama:

  ```
  MRESPONSE(cola [,clase]);
  ```

- **MSERVICE** Devuelve un valor real correspondiente al valor medio del tiempo de servicio de la estación. Se llama:

  ```
  MSERVICE(cola [,clase]);
  ```

- **MTHRUPUT** La función devuelve un valor real correspondiente al valor medio del throughput de la estación. Se llama:

  ```
  MTHRUPUT(cola [,clase]);
  ```

- **OUTPUT** Petición de representación de los resultados estándar de QNAP2.

- **PCUSTNB** Esta función devuelve un valor real que representa la probabilidad para la estación de contener un número de clientes igual al primer parámetro. Debe solicitarse mediante el comando MARGINAL. Se

llama:

```
PCUSTNB(entero, cola [,clase]);
```

- **PMXCUSTNB** Devuelve un entero que representa el número máximo de clientes en la estación durante el último periodo de simulación. Se llama:

  ```
  PMXCUSTNB(cola [,clase]);
  ```

- **SERVNB** Esta función devuelve un valor entero que representa el número de clientes servidos en la estación especificada durante una simulación. Se llama:

  ```
  SERVNB(cola [,clase]);
  ```

- **SONNB** Devuelve un valor entero que representa el número de hijos del cliente dado. Se llama:

  ```
  SONNB(cliente);
  ```

- **VCUSTNB** La función devuelve un valor real que corresponde a la varianza del número medio de clientes en la estación. Se llama:

  ```
  VCUSTNB(cola [,clase]);
  ```

- **VRESPONSE** Esta función devuelve un valor real que corresponde a la varianza del tiempo medio de respuesta de la estación especificada. Se llama:

  ```
  VRESPONSE(cola [,clase]);
  ```