

Trabajo Práctico N° 2. Programación Lógica

Puntuación

Puntaje Total: 100 puntos

Aprobación: 60 puntos

Plazo de entrega: 31/05/2024 - 23:59 hs.

Condiciones de entrega

1. El presente trabajo práctico deberá resolverse en grupo de hasta tres (3) alumnos.
2. Entrega: Se realizará por medio del Campus Virtual de la UTN, en la tarea correspondiente al TP 2. La extensión del archivo será .zip o .rar, de acuerdo al programa de compresión usado. El nombre del archivo se consigue concatenando un prefijo del número del TP con los apellidos de los integrantes por orden alfabético y separados por guiones (por ejemplo: para Nuñez, Acevedo y González, el nombre será: TP2-Acevedo-González-Nuñez.zip). El nombre de archivo no debe contener espacios y debe usar codificación UTF-8. Dentro del archivo de entrega, deben constar los siguientes:
 - Fuentes SWI-Prolog: Se debe entregar un archivo denominado TP2.pl
 - Los casos de prueba se entregarán en un archivo de texto, no deben ser capturas de pantalla. Deberán cubrir diferentes resultados que puedan obtenerse según los predicados solicitados. Se enfatiza que se adjunten casos de prueba que sean claros, válidos y suficientes para poder probar el trabajo. Entregar este archivo con el nombre casos-de-prueba.txt.
 - Archivo de texto (integrantes.txt) con una línea para cada integrante en la cual figure el nombre del alumno/a y su dirección de email.
3. Penalización por entrega fuera de término: Si el trabajo práctico se entrega después de la fecha indicada, y hasta una semana tarde, tendrá una quita de 15 puntos. No serán recibidos trabajos luego de una semana de la fecha de entrega. Los trabajos que se deban rehacer/corregir fuera de la fecha de entrega tendrán una quita de 30 puntos.

Motivación del Problema

Algunos problemas de configuración de dispositivos de hardware tienen relaciones complejas entre los diferentes elementos de la configuración.

Microsoft utilizó un sistema basado en Prolog para configurar los dispositivos de red y los controladores en los sistemas MS Windows NT.¹ El sistema Prolog seleccionado fue el denominado “Small Prolog”, escrito en (K&R) C por Henri de Feraudy y puesto a disposición en dominio público. En las referencias puede encontrar los fuentes de dicho intérprete a los fines de satisfacer tu curiosidad.^{2,3} En la industria se utilizan muchas veces lenguajes de propósito especial, embebidos en sistemas más grandes.

Este trabajo práctico se inspira en esa utilización de Prolog, pero allí termina toda similitud. El enunciado del TP no coincide con el problema de la configuración de la red en MS Windows NT por motivos didácticos, a los fines de simplificar el trabajo. Las referencias históricas son solamente a modo ilustrativo.

Usted va a escribir un programa que ayuda a la configuración de dispositivos de hardware instalados en cierta arquitectura de computadora.

Interrupciones

Los dispositivos de hardware se comunican con la CPU mediante una línea de interrupción que captura la atención de la misma haciendo que una rutina asociada al dispositivo se ejecute a partir de ese momento. La CPU tiene una línea denominada IRQ (Interrupt Request) que puede ser señalada con un valor binario para indicar la necesidad de ocupar la CPU. Como hay una sola línea, existe un chip adicional, denominado controlador de interrupciones, que dispone de varias entradas independientes y una salida conectada al pin de interrupción de la CPU. Cuando alguna de las líneas de entrada del controlador de interrupciones toma un cierto valor binario, eso hace que se produzca la salida que activa la IRQ de la CPU. De esa manera podemos conectar diferentes dispositivos a cada una de las entradas del controlador de interrupciones. El número de la entrada del controlador se usa como índice de un vector de punteros a función (de manera análoga a un vector de funciones en el lenguaje C), donde cada función corresponde al código que se debe ejecutar para el dispositivo que produjo la interrupción.

Algunos dispositivos usuales son: teclado, unidad de disco, puerto serie, tarjeta de sonido, tarjeta de red, scanners y discos conectados mediante una interfaz SCSI, etc.

1

<https://web.archive.org/web/20030218034509/http://www.research.microsoft.com/research/dtg/davidhov/pap.htm> - Using Prolog in Windows NT Network Configuration.

² <https://github.com/CesarBallardini/small-prolog-walnut-creek-original> - Fuentes originales.

<https://github.com/CesarBallardini/small-prolog-walnut-creek-original/tree/gcc-on-gnulinux> - Compilación con GCC en GNU/Linux.

<https://github.com/a-yiorgos/smallprolog/tree/master/junk> - Ejecutable para MS DOS y el DOS Extender.

³ <https://www.cs.cmu.edu/Groups/AI/html/faqs/lang/prolog/prg/part1/faq-doc-18.html> - Is Prolog really used in Windows NT?

Entonces, cuando el disco que está conectado a la línea 14 genera una señal, la CPU recibe la interrupción, busca en el vector de controladores de dispositivos la dirección de la función que hay en la posición 14, y salta a esa dirección para ejecutar el código provisto por el fabricante del disco (o el proveedor del sistema operativo). Los datos se transfieren entre el disco y la memoria, y se termina la interrupción. A continuación la CPU retoma el trabajo que estaba haciendo cuando fue interrumpida.

Esta explicación es también una simplificación, pues no estamos involucrando chips más avanzados como los que gestionan DMA (Direct Memory Access), cuyas operaciones no requieren interrumpir la CPU y se hacen en paralelo con las operaciones de la misma.

Puede encontrar información adicional en las referencias.⁴


A los fines del TP, vamos a usar la siguiente tabla:

IRQ	Función	¿Disponible para cambiar?
0	System Timer	no disponible
1	Keyboard Controller	no disponible
2	usada por algunas tarjetas EGA antiguas	disponible
3	COM2, COM4	disponible
4	COM1, COM3	disponible
5	LPT2, sound card	disponible
6	Floppy disk	disponible
7	LPT1	disponible
8	Real Time Clock	no disponible
9	redirect to irq2, network card	no disponible
10	SCSI, network cards	disponible
11	SCSI	disponible
12	sin uso	disponible
13	Math Coprocessor	disponible

⁴ <https://www.pchell.com/hardware/irqs.shtml>
https://brainbell.com/tutors/A+/Hardware/Interrupt_Request.htm

14	Primary IDE controller, PS/2 mouse	disponible
15	Secondary IDE Controller, network card	disponible

Tabla 1. Asignación de IRQ a dispositivos

 1. Escriba el predicado **hardware_irq/3** donde se registra el hecho que una línea de IRQ está disponible o no disponible para ser usada, y quien la usa, por ejemplo:

```
hardware_irq(0, system_timer, no_disponible).
hardware_irq(1, keyboard_controller, no_disponible).
```

Complete la lista hasta la IRQ 15. Note que los nombres son solamente descripciones que no utilizará en el programa, por lo cual puede escogerlos con libertad. Se recomienda usar átomos como tipo de dato para los nombres a lo largo del TP.

Puertos


Por otro lado, ciertas configuraciones de IRQ y dirección de Entrada/Salida (IO Address) se denominan usualmente “puertos”. Los estándar son:

Puerto	IO Address	IRQ
COM1	0x3F8	4
COM2	0x2F8	3
COM3	0x3E8	4
COM4	0x2E8	3
LPT1	0x378	7
LPT2	0x278	5

Tabla 2. Configuraciones de puertos estándar.

Los puertos COM se destinan a comunicaciones seriales, habitualmente interfaces RS-232, RS-422, módem de diversas características, etc.

Los puertos LPT (Line Printer) se destinan a la comunicación en paralelo con dispositivos de impresión de líneas de alta velocidad.

 2. Escriba el predicado **standard port/3** que describe el nombre, la dirección de IO y la IRQ asociada a los puertos de la tabla que figura más arriba. Ejemplo:

`standard_port(com1, 0x3F8, 4).`

Direcciones de Entrada/Salida (IO Addresses)

En el espacio de direcciones de memoria que la CPU puede leer/escribir, hay ciertas áreas designadas para la comunicación con los dispositivos de hardware. Esas direcciones se mapean sobre el hardware de los dispositivos. Entonces, retomando el ejemplo del disco, cuando la tarjeta controladora de hardware del disco ha leído un bloque, tiene esa información en un área especial, produce una interrupción, lo cual da la oportunidad a la CPU de ejecutar el driver de software del disco, y este lee desde la dirección de I/O el bloque de interés (cuando existe DMA, las cosas son diferentes pero esta aproximación vale para el TP). Hay también un área de memoria en algunos dispositivos que sirve para escribir comandos y para leer el estado del mismo.

Algunas referencias si desea ampliar estos conceptos están en el pie de página.⁵


Las direcciones de IO que usaremos en el TP se describen en la siguiente tabla:

Rangos de direcciones libres:

Inicio	Fin
0x100	0x1FF
0x220	0x26F
0x270	0x39F
0x3B0	0x3DF
0x3E8	0x4FF

Tabla 3. Rango de direcciones de IO libres inicialmente

⁵ https://brainbell.com/tutors/A+/Hardware/I_O_Addresses.htm
https://www.dosdays.co.uk/topics/io_addresses_irq_dma.php

 **3.** Escriba el predicado **free_io_address_range/2** que describe mediante un hecho cada rango de direcciones libres que existe para elegir. Ejemplo:

```
free_io_address_range(0x100, 0x1FF).
```

Configuración de dispositivos

Los dispositivos se pueden configurar proporcionando el puerto estándar, o determinando su IRQ y dirección de IO.

Se presenta el código Prolog para algunos dispositivos de ejemplo:

```
device_config_with_io_address(floppy_disk_controller, 6, 0x3F0).
```

El predicado **device_config_with_io_address/3** tiene como argumentos: el símbolo correspondiente al nombre del dispositivo, el número decimal que corresponde a la IRQ, y un número hexadecimal para indicar la dirección de inicio de su espacio de entrada/salida de IO.

A continuación se describen las posibilidades de configuración de algunos dispositivos que usted debe convertir en cláusulas tipo I o tipo II en SWI Prolog.

Note que hay dispositivos configurados mediante el puerto estándar, como el siguiente ejemplo:

```
device_config_with_port(ecp_printer1, port(lpt1)).
```

Nombre	IRQ	IO Address	Port
floppy_disk_controller	6	0x3F0	
math_coprocessor	13	none	
sound_blaster_16	2, 5, 7, 10	0x220, 0x240, 0x260, 0x280	
scsi_iomega_zip_drive	9, 10, 11, 12	0x340	
parallel_iomega_zip_drive			lpt2
ne1000	2, 3, 4, 5	0x240, 0x300, 0x320, 0x340, 0x360	
ne2000	2, 3, 4, 5, 10, 11, 12, 15	0x240, 0x300, 0x320, 0x340,	


		0x360	
eep_printer2	5	0x278	
mitsumi_ide_16_cdrom	15	0x170	
old_ega_monitor	2	0x3C0	
new_ega_monitor	none	0x3C0	
eep_printer1			lpt1, lpt2
serial_mouse2			com2
hard_disk_c	14	0x1F0	
hard_disk_d	15	0x170	

Tabla 4. Configuraciones permitidas para dispositivos

La configuración de un dispositivo se describe mediante una de las dos alternativas anteriores: o bien se usan las dos columnas de IRQ e IO Address, o bien se usa la columna de puerto.

Si bien se anota una única dirección de IO, tenga en cuenta que el dispositivo utiliza 16 bytes, cuyo inicio es alguna de las direcciones arriba apuntadas en la Tabla 4.

 4. Escriba las cláusulas para cada uno de los dispositivos mencionados en la Tabla 4.

 5. Escriba el predicado **device_config(Name, Irq, IO_address)** que acierta cuando el dispositivo cuyo nombre es **Name**, puede configurarse con la IRQ número **Irq**, y en la dirección **IO_address**:

```
?- device_config(old_ega_monitor, Irq, IO_address).
Irq = 2.
IO_address = 960.
```

Note que hay un solo resultado en este caso, y que la **IO_address** se muestra en decimal. Si desea verlo en hexadecimal, puede consultar con:

```
?- device_config(old_ega_monitor, Irq, IO_decimal),
   format(atom(IO_hexa), '0x~16R', IO_decimal).
```

```
Irq = 2,  
IO_decimal = 960,  
IO_hexa = '0x3C0'.
```

En el caso de dispositivos con múltiples configuraciones, como `scsi_iomega_zip_drive`:

```
?- device_config(scsi_iomega_zip_drive, Irq, IO_decimal),  
   format(atom(IO_hexa), '0x~16R', IO_decimal).
```

```
Irq = 9,  
IO_decimal = 832,  
IO_hexa = '0x340' ;
```

```
Irq = 10,  
IO_decimal = 832,  
IO_hexa = '0x340' ;
```

```
Irq = 11,  
IO_decimal = 832,  
IO_hexa = '0x340' ;
```

```
Irq = 12,  
IO_decimal = 832,  
IO_hexa = '0x340' ;
```

```
false.
```

Cuando hay más de un **Irq** y más de una **IO_address**, **device_config/3** proporcionará todas las combinaciones posibles para ese dispositivo.

En el caso de los dispositivos configurados mediante un puerto estándar, como `ecp_printer1`, el resultado se ve de la siguiente manera:

```
?- device_config(ecp_printer1, Irq, IO_decimal),  
   format(atom(IO_hexa), '0x~16R', IO_decimal).
```

```
Irq = 7,  
IO_decimal = 888,  
IO_hexa = '0x378' ;
```

```
Irq = 5,  
IO_decimal = 632,  
IO_hexa = '0x278'.
```


 **6.** Escriba el predicado **free_initial_irq_list/1** que liga su argumento a la lista de IRQs libres o disponibles inicialmente.


```
?- free_initial_irq_list(L).  
L = [2, 3, 4, 5, 6, 7, 10, 11, 12|...].
```

Note que hay puntos suspensivos en lugar del final de la lista. Esto es así para abreviar las respuestas muy largas en las sesiones interactivas. Si desea ver la respuesta completa, debe modificar un parámetro de la ejecución de SWI Prolog mediante la siguiente consulta:

```
?- set_prolog_flag(answer_write_options,  
    [quoted(true), portray(true), spacing(next_argument)]).
```


Luego puede repetir la consulta:

```
?- free_initial_irq_list(L).  
L = [2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15].
```

 **7.** De manera análoga, escriba el predicado **free_initial_io_address_list/1**, que devuelve una lista de rangos libres inicialmente. Un rango es una lista de dos elementos: el inicio y el fin del rango.

```
?- free_initial_io_address_list(L).  
L = [[256, 511], [544, 623], [624, 927], [944, 991], [1000, 1023]].
```


Note que los resultados se muestran en notación decimal.

 **8.** Escriba un predicado **interval_is_included/2** para verificar si dados dos rangos de direcciones, el primero está incluido en el segundo. Un intervalo se expresa como una lista de dos números:

```
?- interval_is_included([2,3], [1,5]).  
true.
```

```
?- interval_is_included([1,3], [1,5]).  
true.
```

```
?- interval_is_included([1,3], [10,50]).  
false.
```


 **9.** Escriba un predicado **remove_interval/3** que toma un intervalo, y trata de eliminarlo del segundo argumento. El resultado es una lista de intervalos resultante.

```
?- remove_interval([1,5], [1,10], Resultado).
Resultado = [[6, 10]]
```

```
?- remove_interval([8,10], [1,10], Resultado).
Resultado = [[1, 7]]
```

```
?- remove_interval([3,5], [1,10], Resultado).
Resultado = [[1, 2], [6, 10]].
```

```
?- remove_interval([1,5], [10,300], Resultado).
Resultado = [[10, 300]].
```

 **10.** Escriba un predicado **configure_one_device/6** que toma como entradas un nombre de dispositivo, una lista de IRQ libres y una lista de direcciones de IO libres. Como salida genera la configuración de ese dispositivo, y las listas de IRQ e IO con los valores que quedaron libres luego de configurar el dispositivo mencionado.

```
configure_one_device(Device_name,
    Free_irq_list, Free_IO_address_list,
    New_irq_list, New_IO_address_list,
    config(Device_name, Irq, Io)).
```

La configuración de un dispositivo se describe mediante un símbolo funcional **config/3** cuyos argumentos son el nombre del dispositivo, la IRQ y la dirección de IO.

Recuerde que cuando va a reservar una dirección de IO, en el predicado **device_config/3** se le devuelve la dirección de comienzo de un segmento de 16 bytes que debe eliminar de las direcciones disponibles.

Este predicado debe acertar para todas las configuraciones posibles, dadas como restricciones las IRQ y las IO proporcionadas a la entrada. Estas listas pueden ser más cortas que las libres inicialmente.

Pista: Puede usar el predicado **remove_interval/3** para eliminar el intervalo con comienzo **Io** que acaba de configurar para el dispositivo, de la lista **Free_IO_address_list** para obtener **New_IO_address_list**.

Veamos un caso donde se usan las listas de direcciones disponibles inicialmente:

```
?- free_initial_irq_list(Old_irq),
    free_initial_io_address_list(Old_io),
    configure_one_device(scsi_iomega_zip_drive, Old_irq, Old_io,
                        New_irq, New_io,C).

Old_irq = [2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15],
Old_io = [[256, 511], [544, 623], [624, 927], [944, 991], [1000, 1023]],
New_irq = [2, 3, 4, 5, 6, 7, 11, 12, 13, 14, 15],
New_io = [[256, 511], [544, 623], [944, 991], [1000, 1023], [624, 831],
[848, 927]],
C = config(scsci_iomega_zip_drive, 10, 832) ;

Old_irq = [2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15],
Old_io = [[256, 511], [544, 623], [624, 927], [944, 991], [1000, 1023]],
New_irq = [2, 3, 4, 5, 6, 7, 10, 12, 13, 14, 15],
New_io = [[256, 511], [544, 623], [944, 991], [1000, 1023], [624, 831],
[848, 927]],
C = config(scsci_iomega_zip_drive, 11, 832) ;

Old_irq = [2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15],
Old_io = [[256, 511], [544, 623], [624, 927], [944, 991], [1000, 1023]],
New_irq = [2, 3, 4, 5, 6, 7, 10, 11, 13, 14, 15],
New_io = [[256, 511], [544, 623], [944, 991], [1000, 1023], [624, 831],
[848, 927]],
C = config(scsci_iomega_zip_drive, 12, 832) ;
false.
```

Puede verificar por ejemplo el primer resultado. La configuración obtenida es **config(scsci_iomega_zip_drive, 10, 832)**. En la lista de IRQ de salida, el **10** ya no está, pues lo ha asignado al dispositivo. La dirección elegida fue **832** decimal. Si mira **Old_io** verá que está disponible el rango **[624, 927]**, y en **New_io** se ha desdoblado en dos rangos: **[624, 831]**, **[848, 927]**. Se han reservado 16 bytes, del 832 hasta el 847, ambos incluidos.

Y ahora veremos un caso donde las listas no son las iniciales:

```
?- Old_irq = [10], free_initial_io_address_list(Old_io),
    configure_one_device(scsci_iomega_zip_drive, Old_irq, Old_io,
                        New_irq, New_io,C).


Old_irq = [10],
Old_io = [[256, 511], [544, 623], [624, 927], [944, 991], [1000, 1023]],
New_irq = [],
```

```
New_io = [[256, 511], [544, 623], [944, 991], [1000, 1023], [624, 831],
[848, 927]],
C = config(scsi_iomega_zip_drive, 10, 832) ;
false.
```

Y ahora probemos de dejar fuera de las IO disponibles, la que el dispositivo necesita:

```
?- free_initial_irq_list(Old_irq),
   Old_io = [[256, 511], [544, 623], [944, 991], [1000, 1023]],
   configure_one_device(scsi_iomega_zip_drive, Old_irq, Old_io,
                        New_irq, New_io, C).

false.
```

 **11.** Finalmente, defina un predicado **configure_devices(Devices_list, Config_list)** que, dada una lista de dispositivos **Devices_list**, acierte con cada lista de configuraciones posible para los mismos, **Config_list**.

Pista: use **configure_one_device/5** para cada elemento de la lista de dispositivos.

Ejemplos:

```
?- configure_devices([math_coprocessor, hard_disk_c, hard_disk_d,
serial_mouse1, ecp_printer1, old_ega_monitor], CL).

CL = [config(math_coprocessor, 13, none), config(hard_disk_c, 14, 496),
config(hard_disk_d, 15, 368), config(serial_mouse1, 4, 1016),
config(ecp_printer1, 7, 888), config(old_ega_monitor, 2, 960)] ;

CL = [config(math_coprocessor, 13, none), config(hard_disk_c, 14, 496),
config(hard_disk_d, 15, 368), config(serial_mouse1, 4, 1016),
config(ecp_printer1, 5, 632), config(old_ega_monitor, 2, 960)] ;

false.
```

```
?- configure_devices([math_coprocessor, hard_disk_c], CL).

CL = [config(math_coprocessor, 13, none), config(hard_disk_c, 14,
496)] ;

false.
```