PUNTEROS

Introducción

Todas las variables y estructuras de datos que se han utilizado hasta el momento, exceptuando archivos, han sido estructuras estáticas. La cantidad de espacio de memoria ocupada por una variable estática debe ser conocida a priori y declarada por anticipado y no puede ser incrementada/decrementada durante la ejecución de un programa. Por otro lado, usando las estructuras estáticas, la única manera de ordenar datos es haciéndolo físicamente: intercambiar dos datos de un lugar físico a otro de la estructura. La eliminación e inserción de datos también implica corrimiento de elementos. Esta falta de flexibilidad se convierte en una desventaja en algunas situaciones.

Los punteros permiten crear estructuras de datos dinámicas, que tienen la capacidad de variar su tamaño y ocupar la memoria necesaria durante la ejecución de un programa. Las variables que se crean y se destruyen durante la ejecución de un programa se llaman variables dinámicas.

Existen diferentes tipos de estructuras dinámicas, además de archivo, que se construyen a partir del concepto de puntero.

Definimos entonces:

Un puntero es una variable cuyo valor o contenido es una dirección de memoria.

Un puntero es una variable que se utiliza entonces para almacenar la dirección de otra variable, que es la que va a contener el dato a almacenar. Es decir que cuando se trabaja con un puntero, en realidad hay dos variables implicadas: la variable puntero (quien contiene la dirección de memoria de la variable de datos) y la variable apuntada (a quien apunta el puntero y quien contiene el dato en sí mismo).

Declaración de punteros.

Tipos estructurados

```
ptr = ^entero 2;      /*puntero a un entero*/
pp = ^real;           /*puntero a un real*/
```

Variables

```
p: ptr
q: pp
```

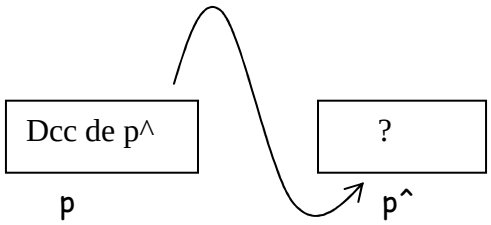
La declaración de un puntero reserva espacio de memoria (en el ej. para las variables p y q), dejando sus contenidos indefinidos hasta que se creen las correspondientes variables apuntadas.

Operaciones con punteros.

- **Creación de punteros**

Las variables de tipo puntero apuntan a variables denominadas dinámicas. Estas variables se crean en forma explícita por medio del procedimiento **crear**.

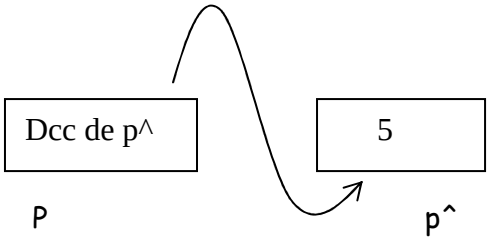
crear(p) → asigna memoria a la variable apuntada por p (p^) y carga en p la dirección de p^



p contiene ahora la dirección de p^ y p^ será quien almacene el dato entero.
La variable p^ se crea en una zona de la memoria para variables dinámicas denominada 'pila' o 'heap'. Si se excede el tamaño de dicha memoria, dará error de ejecución.

- **Asignación de valor y muestra de contenido de la variable apuntada**

p^:= 5 → asigno el valor 5 a la variable apuntada por p.

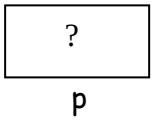


Imprimir: p^ → imprime el valor de la variable apuntada por p.

- **Liberación de un puntero**

La instrucción **Liberar** libera la posición de memoria heap ocupada por una variable apuntada (la de datos) y deja indefinido el contenido de la variable puntero.

Liberar (p) → destruye la variable p^ y deja indefinido a p



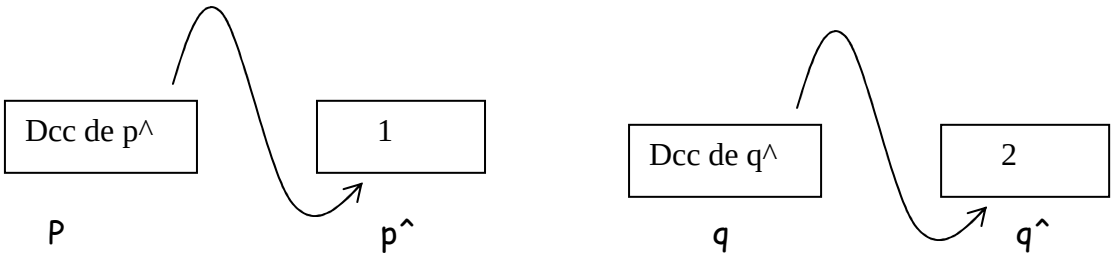
- **Operaciones entre punteros**

Las variables de tipo puntero pueden ser asignadas entre sí, siempre que sean del mismo tipo. No se puede leer ni imprimir el contenido de una variable de tipo puntero.

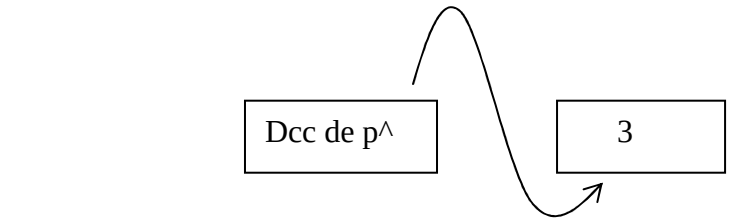
q:= p → asigna a q la dirección que contiene p, es decir las dos variables apuntan a p^.

Ejemplo (se hace seguimiento de la memoria debajo)

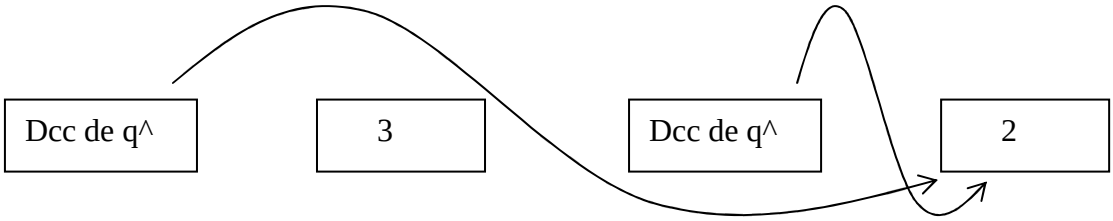
```
programa demopunteros;
tipos estructurados
        ptr=^entero 2
variables
        p,q: ptr
hacer
        crear(p)                    /* A */
        p^:= 1                      /* B */
        crear (q)                   /* C * /
        q^:=2                       /* D */
        imprimir: p^, q^
        p^:= q^+1                   /* E */
        imprimir:p^, q^
        p:=q                        /* F */
        imprimir: p^, q^
        p^:= 7                     /* G */
        imprimir: p^, q^
        crear(p)                    /* H */
        q:=p                        /* I */
finhacer
finprograma.
```



referencias A, B, C, D

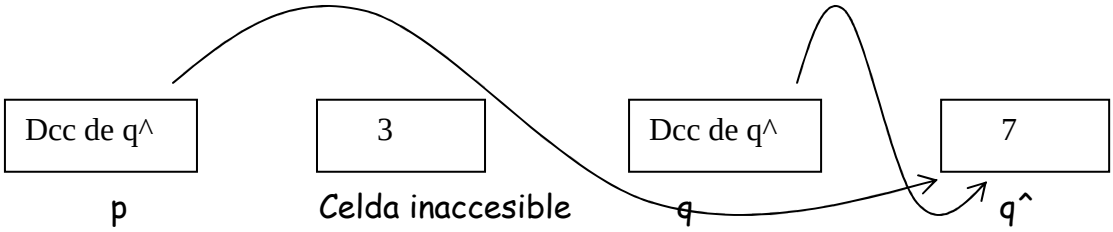


Referencia E p p^

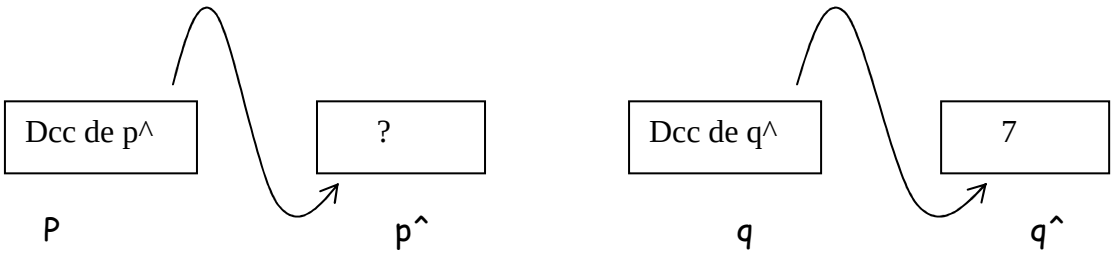


P p^ q q^

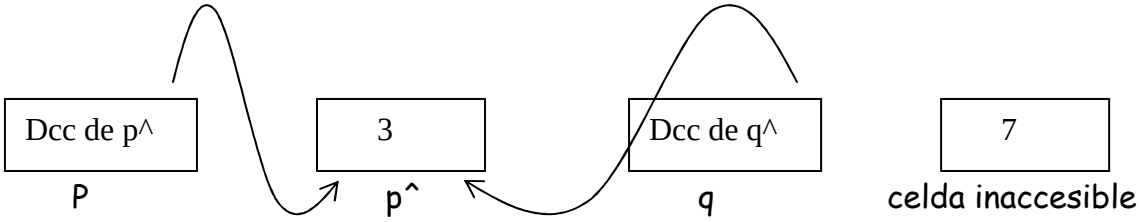
Referencia F



Referencia G



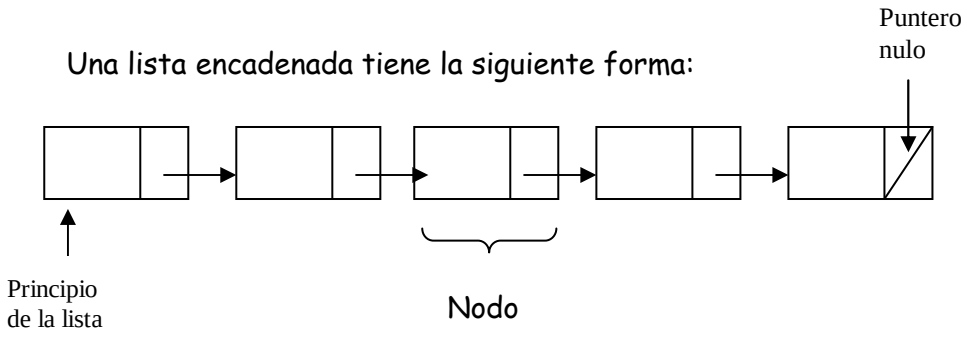
Referencia H



Referencia I

➢ Listas encadenadas

Una lista encadenada tiene la siguiente forma:



Una lista es una sucesión de nodos. Cada nodo tiene un valor y un puntero al siguiente nodo. Solo se conoce de la lista la dirección del primer nodo o el principio de la misma. Para acceder a los nodos restantes se usa el puntero o dirección guardado en cada nodo. El último nodo tiene un valor especial (nulo) denominado **nil** que indica el fin de la lista.

A diferencia de las estructuras estáticas, los elementos de una lista no están en posiciones consecutivas de la memoria, por lo cual para poder trabajar con ellos se necesita direccionarlos, es decir conocer su dirección de memoria. Por esta razón cada elemento de la lista contiene un indicador o puntero que apunta al próximo elemento de la misma.

La flexibilidad que brindan estas estructuras no solo se basa en el uso más eficiente de la memoria (ocupo solo la cantidad que necesito) sino que permite una fácil intercalación y eliminación de elementos de la misma, sin necesidad de hacer movimientos masivos de datos en la memoria (no se necesitan corrimientos).

Los nodos de una lista son registros ya que contienen información heterogénea (al menos un campo de datos y al menos un campo puntero):

- **Definición de listas**

Tipos estructurados

```
lista = ^nodo;                    /* una lista es un puntero a un registro*/
 nodo = registro                 /* el registro contiene datos y punteros*/
        dato: ...                 /* el dato puede ser de cualquier tipo */
        Psig: lista
        finregistro
```

Variables

```
 pi: lista                      /* declaración del inicio de la lista*/
```

Siempre se accede a la lista por su comienzo: el puntero inicial a la misma.

- **Operaciones para el manejo de la lista**

Crear una lista vacía

```
Procedimiento crearLista (ref P: lista)
Hacer
  P:= nil
Finhacer
Finprocedimiento
```

Imprimir los elementos de una lista.

```
Procedimiento imprimirLista (pi:lista)
Variables
 aux:lista
Hacer
aux:= pi;                          /*Se sitúa al principio de la lista*/
Repetir mientras ( aux <>nil )    /*Mientas no se llegue al final de la lista*/
        Imprimir: aux^.dato
        aux:=aux^.psig             /*Pasa al siguiente nodo*/
finrepetirmientras
finhacer
finprocedimiento
```

Insertar un elemento al principio de la lista

```
Procedimiento insertarPpio (ref L:lista; elem:integer)
Variables
 nuevo: lista;
Hacer
        crear(nuevo);              /*se crea el nodo nuevo*/
        Nuevo^.dato:=elem;         /*se carga el dato nuevo*/
        Nuevo^.psig:= L;           /*se engancha el nuevo nodo al pcpio*/
        L:=nuevo;                  /*se redefine el principio de la lista*/
FinHacer
Finprocedimiento
```

Insertar un elemento al final de la lista

```
Procedimiento insertarFinal (REF L:lista; elem:integer)
Variables
 nuevo, actual: lista;
Hacer
        crear(nuevo);              /*se crea el nuevo nodo*/
        Nuevo^.dato:=elem;         /*se carga el dato al nodo*/
        Nuevo^.psig:=nil;          /*el siguiente del último es nil*/
        actual:=l;
        Si (L = nil) entonces      /* si la lista está vacía*/
              L:= nuevo            /* agrega el único elemento como 1º*/
```

```
                sino                    /* se recorre la lista hasta el final*/
            Repetir mientras ( actual^.psig <>nil)
                actual:=actual^psig
            finrepetirmientras
            actual^. psig:=nuevo; /*se engancha el último
        finsi                                nodo con el nuevo*/
finhacer
finprocedimiento
```

Insertar ordenado por algún criterio

```
Procedimiento insertarOrdenado (REF L:lista; elem:integer)
Variables
 nuevo, actual, anterior: lista;
Hacer
crear(nuevo);                   /* se crea nuevo nodo*/
nuevo^.dato:=elem;              /* se carga el dato nuevo*/

actual:=l;                      /* se inicia búsqueda del lugar a insertar*/
anterior:=nil;
Repetir mientras ( actual <>nil) and (actual^.dato<elem)
        anterior:=actual
        actual:=actual^psig
finrepetirmientras
Si (anterior<>nil) entonces                 /* inserta en cuerpo de lista*/
                anterior^.psig:=nuevo
                 nuevo^.psig:= actual
            sino                            /* inserta al principio de lista*/
                nuevo^.psig:= L
                L:= nuevo
finsi
finhacer
finrpocedimiento
```

Eliminar un elemento de la lista

```
Procedimiento Eliminar (REF L:lista; elem:integer)
Variables
 nuevo, actual, anterior: lista;
Hacer
actual:=l;                      /*recorre toda la lista y busca el dato*/
anterior:=nil;
Repetir mientras ( actual <>nil) and (actual^.dato <> elem)
        anterior:=actual
        actual:=actual^psig
finrepetirmientras
Si (actual <> nil)  entonces            /* si el dato fue encontrado*/
```

```
            Si (anterior<>nil) entonces          /* lo borra del cuerpo de L*/
                  anterior^.psig:= actual^.psig
                        sino
                  L:= L^.psig                    /* lo borra de la cabeza de L*/
            Finsi
            Liberar (actual)                     /* libera la celda de memoria*/
      finsi
finhacer
finprocedimiento
```

Ejercicio de aplicación

Una librería ubicada en el centro de la ciudad de La Plata tiene información de los libros que en ella se venden. De los libros se conoce: su nombre, código, autor, editorial, precio y stock. Almacena dicha información en una lista. Se desea:

a) Obtener un listado en donde figure código y editorial de aquellos libros con stock nulo.

b) Simular la entrada de nuevos libros a la librería o la reposición de los ya existentes.

```
Programa ejemlistas
Tipos estructurados

Libro = registro
        Nombre: carácter 20
        Código: entero 4
        Edit: carácter 20
        Autor: carácter 20
        Precio: real 5,2
        Stock: entero 2
        Finregistro

lista = ^nodo;                    /* una lista es un puntero a un registro*/
        nodo = registro           /* el registro contiene datos y punteros*/
              dato: libro         /* el dato puede ser de cualquier tipo */
              Psig: lista
              finregistro

Procedimiento imprimirA (pi:lista)
Variables
 aux:lista

Hacer
aux:= pi;                         /*Se sitúa al principio de la lista*/
```

```
Repetir mientras ( aux <>nil )                    /*Mientas no se llegue al final de la
                                                    lista*/
        Si (aux^. dato. Stock = 0) entonces
                                  Imprimir: aux^.dato.nombre, aux^.dato.edit
        finsi
        aux:=aux^.psig                            /*Pasar al siguiente nodo*/
finrepetirmientras
finhacer
finprocedimiento

Procedimiento insertar (ref L:lista; elem: libro)
Variables
        nuevo: lista;
Hacer
crear(nuevo);              /*se crea el nodo nuevo*/
Nuevo^.dato:=elem;         /*se carga el dato nuevo*/
Nuevo^.psig:= L;           /*se engancha el nuevo nodo al pcpio*/
L:=nuevo;                  /*se redefine el principio de la lista*/
FinHacer
Finprocedimiento

Procedimiento crear (ref l: lista)
Hacer
  l:= nil
Finhacer
Finprocedimiento

Procedimiento Simular (REF L:lista)
Variables
 actual: lista
  reg: libro
  resp: carácter 1
  name: carácter 20
  stocknue: entero 2
Hacer
Imprimir: 'desea reponer libros? (s/n)'
Leer: resp
Repetir mientras (resp='s')
   Imprimir:' ingrese nombre del libro'
    Leer:  name
    actual:=l;                                    /*recorre toda la lista y busca el libro*/
    Repetir mientras ( actual <>nil) and (actual^.dato. nombre <> name)
                        actual:=actual^psig
    finrepetirmientras
   Si (actual <> nil)  entonces                   /* si el libro fue encontrado, suma stock*/
```

```
            Imprimir: ' ingrese stock comprado'
            Leer: stocknue
            Actual^. Dato.stock:=  Actual^. Dato.stock + stocknuevo
          Sino                                                    /* ingresa libro nuevo*/
            Imprimir:' ingrese nombre, código, autor, precio, editorial y stock
                       de un libro'
            Leer: reg. nombre, reg.codigo, reg.autor, reg. precio, reg.edit,
                         reg.stock
            Insertar(l, reg)
Finsi
  Imprimir: 'desea reponer más libros? (s/n)'
  Leer: resp
finrepetirmientras
finhacer
finprocedimiento

Variables
 pi: lista
resp: carácter 1
reg: libro
Hacer

CrearLista (pi);
Imprimir: 'desea ingresar libros? (s/n)'
Leer: resp
Repetir mientras (resp='s')
        Imprimir:' ingrese nombre, código, autor, precio, editorial y stock de un
libro'
        Leer: reg. nombre, reg.codigo, reg.autor, reg. precio, reg.edit, reg.stock
        Insertar(pi, reg)
        Imprimir: 'desea ingresar más libros? (s/n)'
        Leer: resp
Firnrepetirmientras
ImprimirA(pi)
Simular(pi)
FinHacer
Finprograma.
```

Aplicación de listas a Pila y Cola.

Pila

* Una pila es una estructura de datos dinámica cuyos elementos se manipulan siguiendo una política LIFO: last-in, first-out, es decir el último dato

almacenado es el primero en ser sacado y procesado. De esta forma, los elementos de una pila son almacenados y eliminados de la misma por un extremo común o 'tope'.

Declaración de una pila

Tipos estructurados

```
Pila= ^ nodo
Nodo= registro
        Dato: …….          /* los elementos pueden ser de cualquier tipo*/
        Psig: pila
       Finregistro
```

Operaciones sobre pila

Las operaciones básicas que se asocian a pila son:
- Crear: que crea una pila vacía
- Esvacía: que permite determinar si la pila tiene elementos o no
- Apilar un elemento, en el tope de la pila
- Desapilar el elemento que está en el tope de la pila y retornarlo

```
Procedimiento crearPila (ref P:pila)
Hacer
  P:= nil
Finhacer
Finprocedimiento

Procedimiento Apilar (ref P:pila ; ele: ……)  /* Insertar al principio*/
Variables
Nuevo: pila
Hacer
  Crear(nuevo)
  Nuevo^. dato:= ele
  Nuevo^.psig:= P
  P:= nuevo
Finhacer
Finprocedimiento

Procedimiento Desapilar (ref P:pila; ref ele: …..)   /* Elimina del principio*/
variables
 aux: pila
Hacer
  Aux:= p              /*  guardo la dirección de inicio de la pila*/
  Ele:= aux^.dato      /*  retorno el elemento del tope de la pila*/
```

```
  P:= p^. psig            /*  se corre el inicio de la pila*/
  Liberar (aux)           /* se libera el nodo que estaba en el tope*/
Finhacer
Finprocedimiento

Función Esvacia ( p: pila) : booleana
Hacer
  Si p= nil entonces  esvacia:= verdadero
            sino esvacia:= falso
Finhacer
finfuncion
```

Cola

* Una cola es una estructura de datos dinámica cuyos elementos se manipulan siguiendo una política FIFO: first-in, first-out, es decir el primer dato almacenado es el primero en ser sacado y procesado. De esta forma, los elementos de una cola son almacenados y eliminados de la misma por extremos opuestos.

Declaración de una cola

Tipos estructurados

```
lista= ^ nodo                 /* lista de elementos de la cola*/
Nodo= registro
        Dato: …….          /* los elementos pueden ser de cualquier tipo*/
        Psig: lista
       Finregistro

Cola= registro                /*la cola es un registro que contiene la direcc
        Pin: lista              de inicio de la lista de elementos y la direcc
        Pfin: lista             del último elemento */
        finregistro
```

Operaciones sobre cola

Las operaciones básicas que se asocian a cola son:
- Crear: que crea una cola vacía
- Esvacía: que permite determinar si la cola tiene elementos o no
- Encolar un elemento, en el tope de la cola (al final de sus elementos)
- Desencolar el elemento que está al principio de la cola y retornarlo.

```
Procedimiento crearCola (ref C:cola)
Hacer
  c. pin:= nil
```

```
  c.pfin:= nil
Finhacer
Finprocedimiento

Procedimiento encolar (ref C:cola ; ele: ……)  /* Insertar al final*/
Variables
Nuevo: lista       /* nuevo es un puntero a un registro*/
Hacer
  Crear(nuevo)
  Nuevo^. dato:= ele
  Nuevo^.psig:= nil
  Si c.pin = nil  entonces   c.pin= nuevo        /* lista vacía, inserta 1º elem*/
                  Sino       c.pfin^. psig:= nuevo          /*inserta al final*/
 finsi
 c.pfin:= nuevo                    /*pone puntero final en el nodo agregado*/
Finhacer
Finprocedimiento

Procedimiento Desencolar (ref c:cola; ref ele: …..)  /* Elimina del prcipio*/
variables
 aux: lista
Hacer
  Aux:= c.pin                /*  guardo la dirección de inicio de la cola*/
  Ele:= aux^.dato            /*  retorno el elemento del inicio de la cola*/
  c.pin:= aux^. psig         /*  se corre el inicio de la cola*/
  si c.pin= nil entonces
                c.pfin:= nil     /*si se vació la lista se limpia el puntero final*/
  finsi
  Liberar (aux)              /* se libera el nodo que estaba al principio*/
Finhacer
Finprocedimiento

Función Esvacia ( c: cola) : booleana
Hacer
  Si c.pin= nil entonces  esvacia:= verdadero
                 sino esvacia:= falso
  finsi
Finhacer
finfuncion
```

Ejercicios completos

1- Se almacena una palabra en una pila, de a una letra, y se desea imprimir la palabra invertida.

```
Programa ejemplo1
Tipos estructurados
 Pila=^ elem
Elem= registro
         Dato: carácter 1
         Psig: pila
        Finregistro

Procedimiento crear (ref P:pila)
Hacer
  P:= nil
Finhacer
Finprocedimiento

Procedimiento Apilar (ref P:pila ; ele: carácter 1)
Variables
Nuevo: pila
Hacer
  Crear(nuevo)
  Nuevo^. dato:= ele
  Nuevo^.psig:= P
  P:= nuevo
Finhacer
Finprocedimiento

Procedimiento Desapilar (ref P:pila; ref ele: carácter 1)
variables
     aux: pila
    Hacer
      Aux:= p            /*  guardo la dirección de inicio de la pila*/
      Ele:= aux^.dato    /*  retorno el elemento del tope de la pila*/
      P:= p^. psig       /*  se corre el inicio de la pila*/
      Liberar (aux)      /* se libera el nodo que estaba en el tope*/
  Finhacer
Finprocedimiento

Función Esvacia ( p: pila) : booleana
Hacer
     Si p= nil entonces  esvacia:= verdadero
                sino esvacia:= falso
     finsi
```

```
 Finhacer
finfuncion

Variables
P: pila
Let: carácter 1

Hacer
CrearPila(p)
Imprimir: 'ingrese una letra de la palabra'
Leer: let
Repetir mientras (let <> ' ' )
            Apilar (p, let)
            Imprimir: 'ingrese una letra de la palabra'
            Leer: let
Finrepetirmientras
Imprimir: 'palabra invertida'
Repetir mientras ( not (espacia(p)))
      Desapilar(p, let)
      Imprimir: let
Finrepetirmientras
Finhacer
Finprograma.
```

2- Determinar si una expresión aritmética parentizada está balanceada.

```
Programa ejemplo2
Tipos estructurados
 Pila=^ elem
Elem= registro
         Dato: carácter 1
         Psig: pila
        Finregistro

Procedimiento crearPila (ref P:pila)
Hacer
  P:= nil
Finhacer
Finprocedimiento

Procedimiento Apilar (ref P:pila ; ele: carácter 1)
Variables
Nuevo: pila
Hacer
  Crear(nuevo)
```

```
  Nuevo^. dato:= ele
  Nuevo^.psig:= P
  P:= nuevo
Finhacer
Finprocedimiento

Procedimiento Desapilar (ref P:pila; ref ele: carácter 1)
variables
     aux: pila
Hacer
      Aux:= p             /*  guardo la dirección de inicio de la pila*/
      Ele:= aux^.dato     /*  retorno el elemento del tope de la pila*/
      P:= p^. psig        /*  se corre el inicio de la pila*/
      Liberar (aux)       /* se libera el nodo que estaba en el tope*/
Finhacer
Finprocedimiento

Función Esvacia ( p: pila) : booleana
Hacer
 Si p= nil entonces  esvacia:= verdadero
          sino esvacia:= falso
finsi
 Finhacer
finfuncion

Variables
P: pila
dat: carácter 1
cont: entero 2

Hacer
CrearPila(p)
Imprimir:' ingrese un dato de la expresión'
Leer: dat
Repetir mientras ( dat <> '')
    Apilar(p, dat)
    Imprimir:' ingrese un dato de la expresión'
    Leer: dat
Finrepetirmientras
Cont:=0
Repetir mientras ( not (esvacia(p))
     Desapilar (p, dat)
     Si ( dat = '('  ) entonces    cont:= cont +1
                       sino cont := cont – 1
    Finsi
```

```
Finrepetirmientras
Si cont = 0 entonces   imprimir :' está balanceada'
            sino     imprimir : 'está desbalanceada'
Finsi
Finhacer
finprograma
```

2- En una sala de espera de un consultorio se encuentran los pacientes que van a visitar a un médico. Simular la atención de los mismos respetando el orden de llegada y determinar cuántos de ellos tiene obra social.

```
Programa ejemplo 3
Tipos estructurados
lista= ^ nodo
Nodo= registro
            dato: carácter 20
             obra: carácter 2
            Psig: lista
           Finregistro

    Cola= registro
            Pin: lista
            Pfin: lista
            Finregistro

Procedimiento crearCola (ref C:cola)
Hacer
  c. pin:= nil
  c.pfin:= nil
Finhacer
Finprocedimiento

Procedimiento encolar (ref C:cola ; ele: carácter 20; os: carácter 2)
Variables
Nuevo: lista
Hacer
  Crear(nuevo)
  Nuevo^. dato:= ele
  Nuevo^. Obra:= os
  Nuevo^.psig:= nil
  Si c.pin = nil  entonces    c.pin= nuevo
                  Sino        c.pfin^. psig:= nuevo
  finsi
  c.pfin:= nuevo
```

```
Finhacer
Finprocedimiento

Procedimiento Desencolar (ref c:cola; ref ele: carácter 20; ref os: carácter 2)
variables
     aux: lista
Hacer
      Aux:= c.pin
      Ele:= aux^.dato
      Os:= aux^.obra
      c.pin:= aux^. psig
      si c.pin= nil entonces
            c.pfin:= nil
      finsi
      Liberar (aux)
Finhacer
Finprocedimiento

Función Esvacia ( c: cola) : booleana
Hacer
      Si c.pin= nil entonces  esvacia:= verdadero
                        sino esvacia:= falso
      Finsi
finhacer
finfuncion

variables
Co: cola
Nom: carácter 20
Obsoc: carácter 2
C: entero 2

Hacer
CrearCola(co)
Imprimir: ' ingrese nombre de paciente 0  zzz para terminar'
Leer: nom
Repetir mientras ( nom <> 'zzz')
    Imprimir: ' ingrese si el paciente tiene obra social (si/no)'
    Leer: obsoc
     Encolar ( co, nom, obsoc)
    Imprimir: ' ingrese nombre de paciente 0  zzz para terminar'
    Leer: nom
 Finrepetirmientras
C:=0
Repetir mientras (not (esvacia(co))
```

```
        Desencolar (co, nom, obsoc)
        Si obsoc = 'si' entonces  c:= c+ 1
        Finsi
     Finrepetirmientras
Imprimir: 'hay', c, 'pacientes con obra social'
finhacer
Finhacerfinprograma.
```

Bibliografía de referencia:

- Fundamentos de Programación. Algoritmos y estructuras de datos. Luis Joyanes Aguilar.
- Programación en Turbo/Borland Pascal 7.0, Luis Joyanes Aguilar
- Algoritmos, datos y programas. Conceptos básicos, De Giusti, Madoz y otros.