PAC2: Segona Prova d'Avaluació Continuada

Format i data de lliurament

Cal lliurar la solució en un fitxer de tipus **pdf** a l'apartat de lliuraments d'AC de l'aula de teoria.

La data límit per lliurar la solució és el **dilluns, 8 d'Abril de 2019** (a les 23:59 hores).

Presentació

El propòsit d'aquesta segona PAC és comprovar que has adquirit els conceptes explicats en els capítols '*Recursivitat'* i *'TADs*'.

Competències

Transversals

- Capacitat de comunicació en llengua estrangera.
- Coneixements de programació amb llenguatge algorísmic.

Específiques

- Capacitat de dissenyar i construir algorismes informàtics mitjançant tècniques de desenvolupament, integració i reutilització.

Objectius

Els objectius d'aquesta PAC són:

- Adquirir els conceptes teòrics explicats sobre les tècniques d'anàlisi d'algorismes i recursivitat.
- Dissenyar funcions recursives, identificant els casos base i recursius, sent capaços de simular la seqüència de crides donada una entrada.
- Implementar un algorisme iteratiu a partir d'un algorisme recursiu.
- Manipular operacions dels TADs bàsics implementats amb punters.
- Dissenyar un TAD complex fruït de la combinació de TADs bàsics.

Descripció de la PAC a realitzar

Raona i justifica totes les respostes.

Les respostes incorrectes **no** disminueixen la nota.

Tots els dissenys i implementacions han de realitzar-se en llenguatge algorísmic. Els noms dels tipus, dels atributs i de les operacions s'han d'escriure en anglès. Els comentaris i missatges d'error no és obligatori fer-los en anglès, tot i que es valorarà positivament que es faci, ja que és l'estàndard.

Recursos

Per realitzar aquesta prova disposes dels següents recursos:

Bàsics

- Materials en format **web de l'assignatura.**
- **Fòrum de l'aula de teoria.** Disposes d'un espai associat en l'assignatura on pots plantejar els teus dubtes sobre l'enunciat.

Complementaris

- **Cercador web.** La forma més ràpida d'obtenir informació ampliada i extra sobre qualsevol aspecte de l'assignatura és mitjançant un cercador web.
- Solució de la PAC d'un semestre anterior.

Criteris de valoració

Per a la valoració dels exercicis es tindrà en compte:

- L'adequació de la resposta a la pregunta formulada.
- Utilització correcta del llenguatge algorísmic.
- Claredat de la resposta.
- Completesa i nivell de detall de la resposta aportada.

Avís

- Aprofitem per recordar que **està totalment prohibit copiar en les PACs** de l'assignatura. S'entén que hi pot haver un treball o comunicació entre els alumnes durant la realització de l'activitat, però el lliurament d'aquesta ha de ser individual i diferenciat de la resta.
- Així doncs, els lliuraments que continguin alguna part idèntica respecte a lliuraments d'altres estudiants seran considerats còpies i tots els implicats (sense que sigui rellevant el vincle existent entre ells) suspendran l'activitat lliurada.

Exercici 1: Conceptes bàsics de recursivitat (20%)

Tasca: Respon les preguntes següents justificant les respostes:

i) Quina és la funció del cas base i del cas recursiu en un algorisme recursiu?

El **cas base** és el cas que finalitza la recursivitat.

El **cas recursiu** es el cas que acosta la recursivitat al cas base.

ii) Quina és la missió de la funció *duplicate* en un TAD?

La missió de la funció *duplicate* és realitzar una còpia exacta d'un objecte sobre un altre del mateix TAD. D'aquesta manera s'obté la independència de la implementació del TAD, ja que es realitza la còpia sense tenir en compte la implementació concreta del TAD, o en altres paraules, desconeixent si hi ha punters a l'interior del TAD.

iii) Quins avantatges i desavantatges tenen els algorismes recursius respecte als iteratius ?

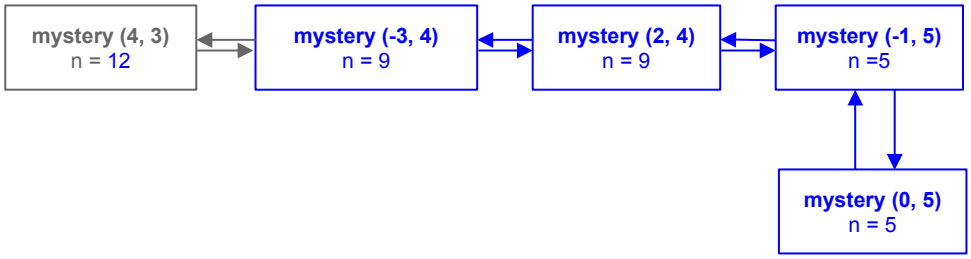
Els algorismes recursius brinden una solució intuïtiva a programes recurrents, i també són més curts que els iteratius. No obstant això poden necessitar molta memòria i ser més lents a causa de la profunditat de les crides recursives i al fet que en cada crida es crea un nou entorn, amb noves variables locals i paràmetres.

iv) Donada la funció recursiva **mystery**, calcula quin valor retorna la crida **mystery(4,3)** i completa el **model de les còpies** per veure com has arribat al resultat:

```
function mystery (n : integer, a:  integer ) : integer
var
        result : integer;
end var
        if n = 0 then
                result := a;
        else
                if n > 0 then
                        result := a + mystery (1-n, a+1);
                else
                        result :=  mystery (-1-n, a);
                end if
```

```
        end if
        return result;
    end function
```

El resultat és 12, que resulta d'executar la seqüència de crides representades en la següent figura:



Exercici 2: Disseny d'algorismes recursius (20%)

Tasca: Donada la descripció dels problemes següents, dissenya els algorismes recursius que els resolen.

Consell: Abans de començar a escriure cada algorisme, has d'identificar els casos base i recursius.

i) Dissenya la funció recursiva **dot_product** que permet calcular el producte escalar de dos vectors de **n** dimensions.

Exemple: dot_product *({1.5,2.7,3.0}, {3.0,2.5,1.0}, 3)* retorna 14.25.

```
function dot_product ( a : vector [MAX] of real,
                       b : vector [MAX] of real, n: integer ) : real
Pre: { n=N i 0 < N ≤ MAX}
var
        res : real;
end var
        if n = 1 then
                res := a[1]*b[1];
        else
                res := a[n]*b[n] + dot_product(a, b, n-1);
        end if
        return res;
end function
```

ii) Dissenya l'acció recursiva **negative_stack** que desempila tots els enters de la pila **p** i retorna el nombre d'elements negatius emmagatzemats en ella. Si la pila està buida llavors retorna el valor 0.

Exemple: negative_stack([2, -1, -6, 3>, res) retorna el valor 2 en la variable res. El cim de la pila de l'exemple és 3.

```
action negative_stack ( inout p: stack(integer), out res: integer)
var
        prev : integer;
end var
        if empty(p) then
                res := 0;
        else
                prev := top(p);
                pop(p);
                negative_stack(p, res);

                if prev<0 then
                        res := res +1;
```

```
                    end if
                end if
            end action
```

Exercici 3: Convertir algorismes recursius en iteratius (20%)

Tasca: Donada una acció recursiva transforma-la en iterativa

i) Completa el disseny de la funció recursiva que calcula la suma dels divisors positius d'un nombre **n**.

La crida inicial a *sum_divisors* es fa d'aquesta manera:

sum_divisors (n, 1);

Exemple: el resultat de la crida *sum_divisors (6,1)* és 12, ja que els divisors de 6 són 6,3,2 i 1.

```
function sum_divisors (n: integer, d: integer): integer
Pre: { n=N i N >0}
var
        result : integer;
end var

        if  n = d then
                result :=  n;
        else
                result := sum_divisors(n, d+1);
                if n mod d = 0 then
                        result := result + d;
                end if
        end if
        return result;

end function
```

ii) Transforma la funció recursiva *sum_divisors* en una funció iterativa.

```
function sum_divisors (n: integer, d: integer): integer
Pre: { n=N i N >0}
var
        result : integer;
end var

        result := n;
        while  n ≠ d  do
```

```
            if n mod d = 0 then
                result := result + d;
            end if
            d := d + 1;
        end while
        return result;
    end function
```

Exercici 4: Modificació de TAD bàsics (20%)

Tasca: Donada la implementació del TAD cua amb punters (explicada en els apunts):

```
type
        node = record
                e : elem;
                next : pointer to node;
        end record

        queue = record
                first, last : pointer to node;
        end record
end type
```

Estén el tipus afegint les operacions següents:

i) **count**: funció que retorna el nombre d'elements emmagatzemats a la cua.

```
function count (c: queue(elem)) : integer
var
        tmp: pointer to node;
        num: integer;
end var
        num := 0;
        tmp := c.first;
        while tmp ≠ NULL do
                num := num + 1;
                tmp := tmp^.next;
        end while
        return num;
end function
```

Per dissenyar aquesta funció no pots utilitzar les operacions del tipus **cua** (enqueue, dequeue, head...). Així doncs, has de treballar directament amb la implementació del tipus que us hem facilitat en l'enunciat.

ii) **to_stack**: funció que donada una cua retorna una pila amb el contingut de la cua, mantenint l'ordre d'aquesta de manera que el fons de la pila emmagatzemarà el primer element de la cua.

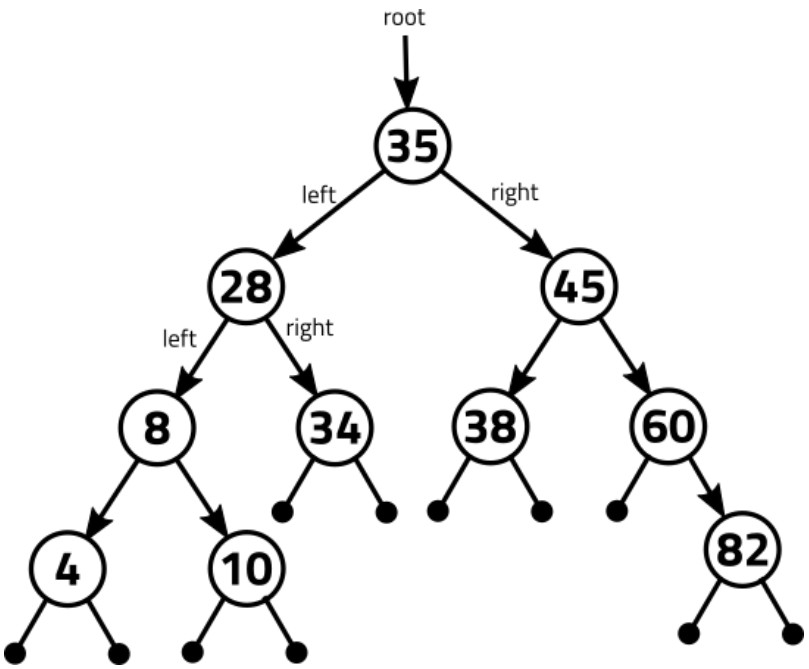
```
function to_stack (c: queue(elem)) : stack(elem)
var
        s : stack(elem);
        copy: queue(elem);
        e: elem;
fvar
        s := create();
        duplicate (copy, c);
        while not empty (copy) do
                e:= head (copy);
                dequeue (copy);
                push(s, e);
                destroy (e);
        end while
        destroy (copy);
        return s;
end function
```

Per dissenyar aquesta funció has d'utilitzar les operacions del tipus cua i pila, és a dir, aquest cop desconeixes com s'han implementat internament aquests dos tipus.

Exercici 5: Disseny d'un tipus amb punters (20%)

Tasca: Fins ara hem treballat amb els TADs pila, cua i llista, però a vegades aquests no ens permeten reflectir la realitat i necessitem crear tipus més complexos (com per exemple, una llista ordenada).

Definim el TAD *tBinTree* que conté un arbre binari ordenat d'elements de manera que cada element té dos fills: el fill esquerre és menor que l'arrel i l'arrel és menor que el fill dret. Per facilitar la seva comprensió, us proporcionem un exemple de com podria ser un cas concret d'aquest tipus implementat amb punters:



En el dibuix es pot veure que el node arrel és l'element 35 i que, per exemple, el node 28 té com a fill esquerre un subarbre on tots els elements són menors que 28 i com a fill dret té el node 34 (28 < 34). També podem veure altres casos com el node 60 que només té un fill i els nodes 4, 10, 34, 38 i 82 no en tenen cap.

i) A partir d'aquesta explicació, completa la definició dels següents TADs utilitzant punters:

```
type
        tNode = record
                elem: integer;
                left: pointer to tNode;
```

```
        right: pointer to tNode;
    end record

    tBinTree = record
        root: pointer to tNode;
    end record
end type
```

ii) Aquest nou TAD oferirà diverses operacions, entre elles et demanem que acabis la implementació **recursiva** de l'operació que retorna cert si un valor enter està emmagatzemat a l'arbre binari, o fals en cas contrari.

```
function find (t: tBinTree, e: integer) : boolean
    return find_rec ( t.root, e );
end function

function find_rec (p: pointer to tNode, e: integer): boolean
var
    found: boolean;
end var
    if p = NULL then
        found := false;
    else
        if e = p->elem then
            found := true;
        else
            if e < p->elem then
                found := find_rec (p-> left, e);
            else
                found := find_rec (p-> right, e);
            end if
        end if
    end if
    return found;
end function
```