Óscar Font 183826 P201
Lucía Gasión 183849 G-AY

INFORME PRÀCTICA 3

Comprensió de la Recursivitat

6.1 Expliqueu amb les vostres paraules l'algorisme recursiu que proporciona la solució del joc, detallant quina és la base de recursió, quina és la regla recursiva i per què funciona. Afegir dibuixos que us semblin necessaris per fer entenedora l'explicació i el concepte.

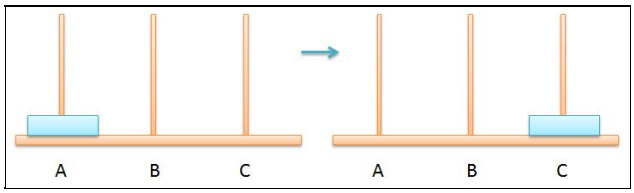
El mètode general o algorisme simplificat que que plantegem per a solucionar el joc és el següent:

```
hanoi (n, A, B, C)
    si nd==1
          moure (A, C)
    sino
          hanoi (nd-1, A, C, B)
          moure (A, C)
          hanoi (nd-1, B, A, C)
    fi-si
```

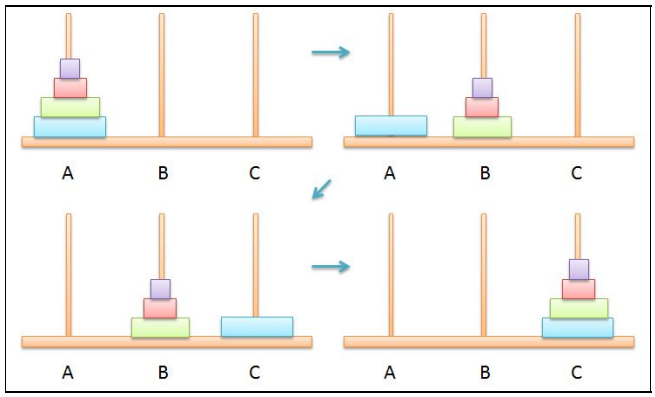
On nd fa referència al número de discos que tenim en el joc i, les tres lletres —A, B i C—, fan referència a els tres pilars que tenim i sobre els quals podem efectuar els moviments dels discos.

Aquest algorisme, farà una cosa o una altra depenent del nombre de discos (n) que té el joc:

Cas 1: Si nd té valor 1, és a dir, si el joc està compost per un sol disc, llavors el programa simplement el mourà del pilar A al pilar C i el joc finalitzarà.



Cas 2: Si nd té valor diferent de 1, l'algorisme farà que nd-1 discos es moguin del pilar A al pilar B. Després mourà el disc restant del pilar A al pilar C i, a continuació, els nd-1 discos del pilar B al pilar C.



6.2 Construir l'arbre d'execució del model de les còpies fet a ma per l'execució de les crides 1) hanoi(3,0,1,2), hanoi(4,0,1,2) i hanoi(5, 0, 1, 2). Posar el número de moviment en cada moviment que apareix a l'arbre, i posar el nivell de profunditat en cada nivell. Comprovar que els moviments i els nivells de profunditat concorden amb els donats pel programa. (Fer foto de cada un dels arbres i incloure en l'entrega els tres fitxers per separat).

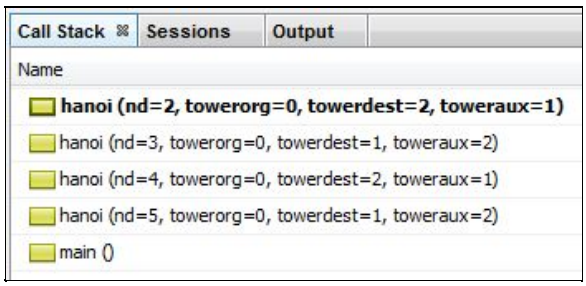
a) Quina és la torre destí, i nivell de profunditat del primer moviment en cada un dels tres casos?
En el cas hanoi(3,0,1,2) la torre destí és la torre 1 (o A) i el nivell de profunditat és 3. En el cas hanoi(4,0,1,2) la torre destí és la torre 2 (o B) i el nivell de profunditat és 4. En el cas hanoi(5, 0, 1, 2) la torre destí és la torre 1 (o A) i el nivell de profunditat és 5.

b) L'últim moviment del primer sub-arbre de profunditat 2 en el cas de 3 discs és el 3. Quina sub-torre ha mogut aquest sub-arbre al acabar aquest moviment?
Ha mogut la sub-torre formada pel disc mitjà i el disc més petit a la torre 3 (o C).

c) Quin és l'últim moviment del primer sub-arbre de profunditat 2 en el cas de 4 discs i quina sub-torre ha mogut?
L'últim moviment és el 3 i mou la sub-torre formada pels dos discs més petits a la torre 2 (o B).

d) Quin és el primer moviment de l'últim sub-arbre de profunditat 2 en el cas de 4 discs i a quin nivell de profunditat s'efectua.
El primer moviment és el 15 i s'efectua al nivell de profunditat 4.

Executa el codi bàsic amb debugger i para l'execució en aquest nivell i fes una captura del callstack i inclou-la aquí. Comprova i comenta els valors dels paràmetres de les diferents crides recursives del debugger en relació el camí del teu arbre recursiu fet a ma.



Com podem veure, al executar la funció hanoi, ens apareixen cinc components al call stack, dels quals quatre són sub-funcions hanoi i la restant és el main. Veiem que els valors que hem introduït prèviament, efectivament coincideixen amb els valors de les diferents crides de les funcions hanoi que s'han creat. A més, també apareix la funció principal main.

Els diferents call stacks ens aclaren també quins són els valors que hem cridat. Nosaltres introduïm hanoi(3, 0, 1, 2), hanoi(4, 0, 1, 2) i hanoi(5, 0, 1, 2), però no sabem què volen dir els números. Així doncs, el call stack ens diu que, per exemple, en el primer cas, el nombre de discos és 3, la torre origen és la 0 (la primera), la torre destí és la 1 (la del mig) i la torre auxiliar és la 2 (la del final). Així doncs, en aquest cas, els discos ordenats s'iniciaran en la torre 0 i acabaran en la torre 1.

6.3 Com s'ho fa el programa bàsic per recordar quin moviment ha de fer i ser coherent amb els moviments anteriors i posteriors si no guarda informació ni dels discs ni de l'estat de les torres? Raona la resposta. Podem saber quin disc es mou en cada moviment? Si és que si explica com podem fer que el programa bàsic ens ho mostri. Si és que no indica com s'ha de fer per poder imprimir amb els moviments quin disc es mou.

El programa bàsic, mitjançant la funció `hanoi`, sap el moviment que ha de fer i, mitjançant la funció `move`, se li informa a l'usuari de dit moviment. Així doncs, la part fonamental per a entendre els moviments és la que veiem a continuació; pertany a la funció `hanoi` i representa el cas 2, estudiat en la qüestió 6.1.

```
hanoi(nd - 1, towerorg, toweraux, towerdest);
move(towerorg, towerdest);
hanoi(nd - 1, toweraux, towerdest, towerorg);
```

nd: número de discos.
towerorg: pilar origen (A).
toweraux: pilar auxiliar (B).
towerdest: pilar destí (C).

Així doncs, el programa sap, amb aquestes tres línies de codi que, partint des de 0, és a dir, amb tots els discos ordenats en el pilar origen (A), que ha de moure tots els discos menys l'últim del pilar

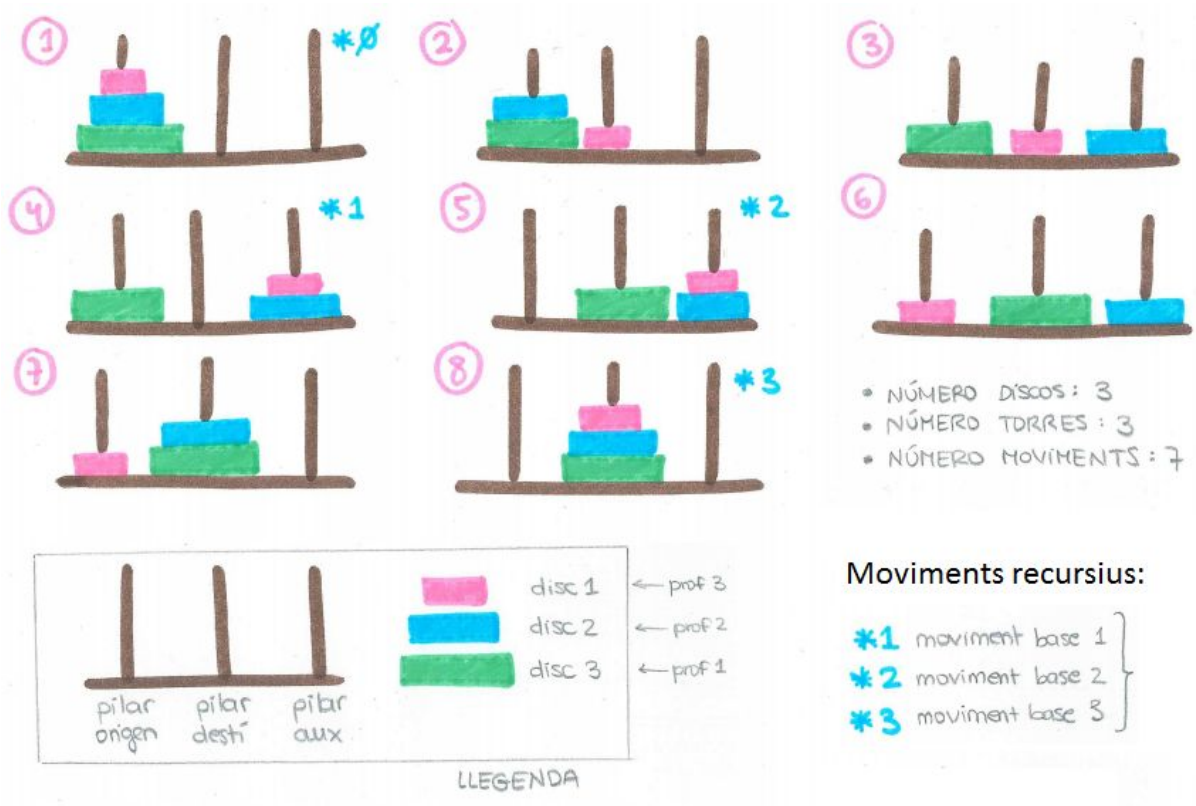
origen (A) al pilar auxiliar (B). Després canviar el disc restant del pilar origen (A) al pilar destí (C) i, a continuació, canviar tots els altres discos del pilar auxiliar (B) al pilar destí (C). Un cop acabat, doncs, retorna els moviments realitzats per pantalla. D'aquesta manera és impossible que s'equivoqui a l'hora de realitzar els passos, atès que sempre treballa amb 1 disc o amb n-1 discos.

Respecte a la pregunta si podem saber quin disc es mou en cada moviment, la resposta és sí. Sí que podem saber quin disc es mou en cada moviment perquè, per una banda, sabem que en les línies de codi 1 i 3 —de les que hem inclòs en aquesta pròpia qüestió— el disc que es mourà primer serà aquell que tingui més discos a sota, seguit del que està a sota d'aquest i així succesivament. Per altra banda, la línia de codi 2 només mourà un sol disc, que és el més gran del joc. De totes maneres, la millor forma de comprovar-ho seria imprimint els moviments del joc un per un, en una representació com la que se'ns ha proporcionat com a material utilitzable.

6.4 Construcció d'un moral per la comprensió: Executar l'executable hanoiplus que implementa la solució que se us demana en aquest enunciat. Executeu-lo pels casos que vulgueu, se us proporciona l'output del cas de 3, 4, i 5 discs per si no us executa al vostra ordinador. Construir un moral gran que contingui l'arbre de recursió anterior, però ara afegint l'estat de les torres en cada node de l'arbre i el número de disc que es mou en cada moviment. Això ocupa espai i pot ser gran. Ho podeu fer retallant els dibuixos del fitxer de sortida i posant-los espaiadament en una cartolina o similar (per poder-ho transportar). Feu el moral pel cas que us sembli millor per entendre-ho, 3, 4 o 5 discs. Fer una foto d'aquest moral amb màxima resolució per poder visualitzar-lo correctament a diferents nivells de zoom/detall electrònicament. Entregar aquesta foto com un fitxer separat en la vostra entrega final. Si heu fet més d'un cas aporteu tants fitxers com arbres hagueu muntat. Anomenar aquest fitxer com moral-hanoi-n amb l'extensió del format que sigui (jpg/gif/...) i on n indiqui el nombre de discs que correspon el moral. Per cada sub-arbre crear un paper que cobreixi tot el seu sub-arbre (fills) i que contingui l'estat final de les torres de hanoi que és el final de l'últim moviment del sub-arbre. Quan es posi sobre el paper estem eliminant l'execució i detall d'aquest sub-arbre pel seu efecte i per tant emula el retorn recursiu de tot el sub-arbre que oculta. Si disposeu de tots els papers per cada arbre i obriu l'arbre al anar fent crides recursives, i els tapeu amb el resultat per anar retornant de les crides recursives, teniu una eina visual pel seguiment de l'execució recursiva del problema i per consolidar la comprensió. Per cada un d'aquests papers que sigui blanc en una cara i l'estat de sortida en l'altre. Així si no s'ha creat encara aquest sub-arbre en l'execució el podeu fer servir per deixar en blanc l'espai en l'arbre total, anar destapant quan es fan les crides i tornar a tapar amb el resultat amb els retorns de les crides. Feu un seguit de fotos amb els papers ben col·locats per mostrar la seqüència de l'execució. Entregar aquesta seqüència de fotos en un fitxer separat ppt per exemple i incloure l'explicació de cada punt que es mostra. Se us demana que la seqüència mostri almenys quatre (4) punts diferents de l'estat de l'execució. Portar el moral en paper a la sessió de pràctiques que es revisa la pre-entrega.

Nom del fitxer:	`moral-hanoi-3.png`
Annexat a:	la pròpia carpeta que conté la pràctica.

A partir del fractal recursiu que hem creat, hem aconseguit identificar el nombre de moviments òptims a realitzar per a 3 discs i 3 torres. La adaptació que hem recreat és la que es mostra a continuació, tenint en compte que nosaltres hem pres com a referència pel programa la torre 0 com a origen, la torre 1 com a destí i la torre 2 com a auxiliar.



6.5 Després d'aquest estudi, creieu que teniu el concepte de recursivitat clar? Indiqueu la impressió de la vostra comprensió i aporteu aquí les preguntes, dubtes o comentaris que tingueu pendents de resoldre per completar la comprensió de la recursivitat. Si creieu que sí expliqueu què és el que més us ha ajudat a entendre-ho i perquè.

Sí, creiem que el concepte de recursivitat ens queda bastant clar. Pensem que el que ens ha ajudat a entendre bé el concepte és la realització dels arbres dels algorismes d'exemple, perquè es veu a cada pas com es comporta l'algorisme, juntament amb el mapa conceptual del fractal recursiu.

Disseny (de l'Ampliació) del Programa

6.6 Quina diferència hi ha entre només treure una línia de text tal i com fa el codi bàsic proporcionat comparat en generar el dibuix complert que es demana que faci el programa hanoiplus? És a dir explicar i justificar els canvis que heu de fer en el programa per incrementar la funcionalitat des del punt de vista del disseny (i sense haver-ho implementat encara). Revisar totes les funcionalitats que es demana afegir (ja que hi ha més que recursivitat) en aquesta pràctica.

El problema que té el joc del hanoi és que, si els moviments que es realitzen no són els òptims, el procés per arribar a la solució final pot ser més dificultosa i més llarga. D'aquesta manera, a partir del que hem comentat a l'apartat 6.4, ens hem adonat de la importància tant de la recursivitat en el joc com dels passos intermitjos. Així doncs, si treus una sola línia recursiva del codi bàsic proporcionat, la forma d'arribar a la solució final varia molt perquè el camí a seguir presenta diferents bifurcacions, la direcció correcta de les quals no ha estat definida.

6.7 Quina estructura de dades i variables principals cal definir? Explicar la necessitat de cada estructura i variable i el perquè de com i on es defineixen i declaren. (per la pre-entrega suposar memòria estàtica i per l'entrega final aportar les declaracions necessàries amb memòria dinàmica).

Totes les variables i estructures de dades, de les quals en parlarem aquí, es troben dins el fitxer "Declaracions.h". Recordem que són les variables més importants, atès que de les variables secundàries en parlarem a l'apartat 6.10.

General: Ho vàrem pensar durant la pre-entrega i ho hem implementat en el codi entregable.

`#define NDISCS 5`	`#define NTOWERS 3`	`int nd`
És el nombre de discos a considerar en cas que l'usuari no introdueixi cap número. Pot canviar.	És el nombre de torres a considerar. No es pot canviar en aquesta pràctica.	És una variable integer que simbolitza el nombre de discos que l'usuari introdueix per a utilitzar a l'hora de realitzar el joc.
`int towerorg`	`int toweraux`	`int towerdest`
És una variable integer que fa referència al pilar origen (A).	És una variable integer que fa referència al pilar auxiliar (B).	És una variable integer que fa referència al pilar destí (C).

Pre-entrega: El que vàrem pensar que necessitaríem però finalment no ho hem implementat.

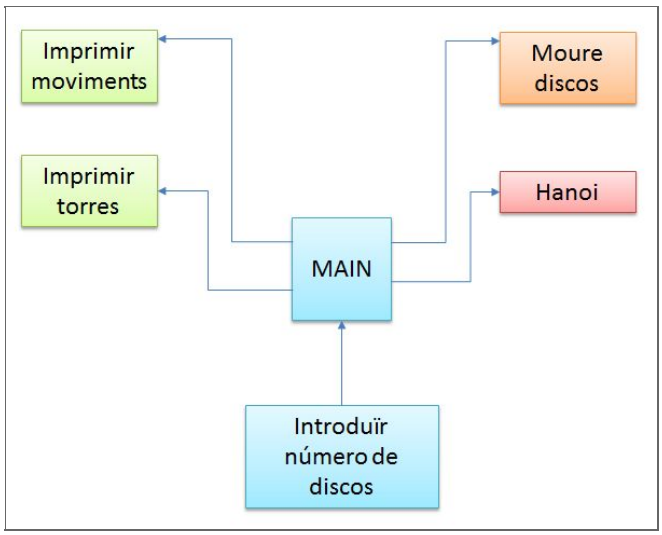
`int matriu`	`int fil`	`int col`
És un array en forma de matriu matriu de tipus entera on representarem les torres amb els seus respectius discos (*).	És una variable integer que fa referència al nombre de files que tindrà cada matriu.	És una variable integer que fa referència al nombre de columnes que tindrà cada matriu.

Entrega final: Adaptacions de les idees anteriors que donen lloc a dos structs per a la entrega final.

``typedef struct{ int move; int disc; char twrorg; char twrdest; int prof; int *v[NTOWERS]; }sState;```	```typedef struct{ int Ntwrs; int Nd; int NMov; char *Nom; char *C_op; char *Out_pth; }sCap;```
És la estructura de dades que utilitzarem per a realitzar la matriu necessària pel joc. La hem creat a partir de les variables de la taula anterior amb novetats que hem necessitat. alt [NDISCS] és la altura de la matriu (segons els discos), twrs[NTOWERS][NDISCS] és la amplada de la matriu (segons els discos i les torres) i move és el número de moviment.	És la estructura de dades que emmagatzema tres dades diferents: el nombre de torres (Ntwrs) —que sempre serà 3—, el nombre de discos (Nd) —que el decideix l'usuari i, en cas que aquest no introdueixi res, el programa prendrà com defecte el nombre 3—, el nombre de moviments (NMov) en què es realitza el hanoi escollit, un punter al nom del fitxer (*Nom), un punter al codi d'operació del fitxer (*C_op) i un punter a la ruta de sortida (*Out_pth).

6.8 Aportar el dibuix del mapa conceptual del disseny (primer esborrany abans de començar la implementació) i explicar-lo.

Aquest és l'esborrany del mapa de disseny. Conté les quatre funcions principals, la funcionalitat d'introduïr el nombre de discos i la funció main. Faltaria saber com relacionar cada funció amb el main i com implementar el disseny, però d'això ja ens encarregarem més endavant, quan tinguem clares les estructures de dades i les variables principals que volem utilitzar.

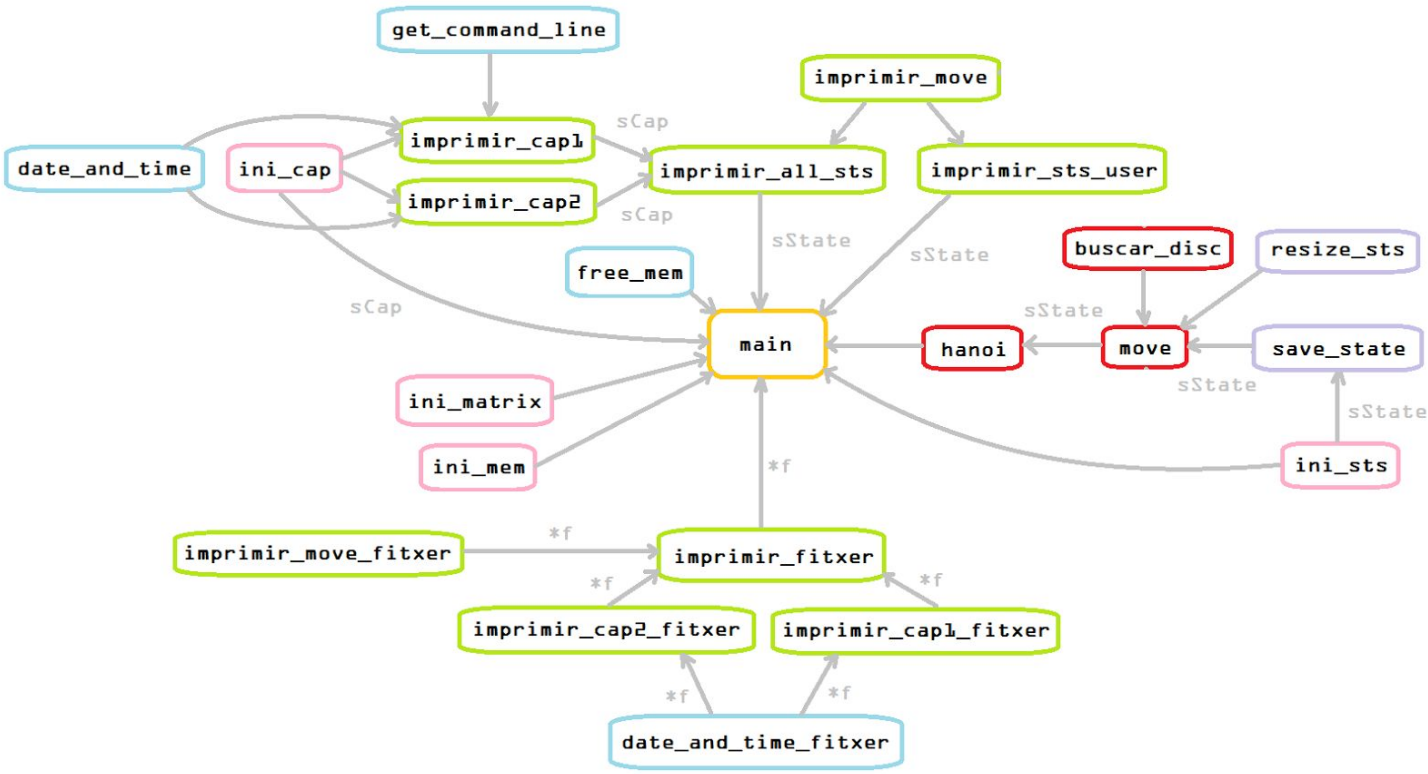


Funcions principals

—Imprimir moviments:
Serà la funció que ens mostrarà per pantalla els moviments efectuats per escrit.
—Imprimir torres:
Serà la funció que ens mostrarà per pantalla els moviments efectuats de forma gràfica.
—Moure discos:
Serà la funció que mourà un disc per a cada moviment efectuat.
—Hanoi:
Serà la funció hanoi.c que ens han proporcionat com a material utilitzable.

Implementació

6.9 Aportar el dibuix del mapa conceptual de la implementació final entregada. Explicar-lo.



6.10 Com heu modularitzat el codi en funcions? Explicar en general com s'ha fet i detallar la justificació de cada funció i els seus paràmetres. Es pot fer aquí utilitzant el mapa conceptual de la pregunta 6.9) o aportar el detall d'aquesta explicació allà amb el mapa.

NOTA: Les explicacions sobre el funcionament de cada funció utilitzada en el programa venen escrites dins del propi codi "Definicions.c". Així doncs, aquí només explicarem els paràmetres principals de cadascuna a partir del mapa conceptual de l'apartat anterior.

LLEGENDA: ■ enter ■ caràcter ■ punter ■ doble punter ▷ funció

▷ **Buscar_disc**

■ towerorg : torre origen.
■■ w[NTOWERS] : vector amb el nombre de torres.
■ disc : número del disc que hem de moure.
■ d : número del disc que hem de moure menys 1.

▷ **get_command_line**

■ argc : nombre d'arguments passats al programa.
■■ argv : array unidimensional de cadenes, cadascuna de les quals és un argument de argc.
■■ nd : nombre de discos.
■■ output : ruta de sortida.
■■ codi_op : codi d'operació del fitxer.
■■ nom_fx : nom del fitxer on s'imprimeix.

▷ **date_and_time**

▷ **save_state**

sState ■ Sts : struct sState.
■■ w[NTOWERS] : vector amb el nombre de torres.
■ disc : número del disc que hem de moure.
■ m : indica quin moviment és.
■ towerorg : torre origen.
■ towerdest : torre destí.
■ ptwr : posició de la torre.
■ depth : profunditat de la matriu.

▷ **move**

■ towerorg : torre origen.
■ towerdest : torre destí.
sState ■ Sts : struct sState.
■■ w[NTOWERS] : vector amb el nombre de torres.
■ m : indica quin moviment és.
■ Size_Sts : mida dels estats.

▷ **imprimir_move**

sState ■ Sts : struct sState.
■ move : número del moviment.
■ disc : número del disc que hem de moure.

▷ **ini_cap**

sCap ■ Cap : struct sCap.
■ Size_cap : mida de la capçalera.
■ moves : nombre de moviments.
■■ nom_fx : nom del fitxer on s'imprimeix.
■■ output : ruta de sortida.
■■ codi_op : codi d'operació del fitxer.
■ discs : nombre de discs de la comanda.

▷ **imprimir_cap1**

sCap ■ Cap : struct sCap.

▷ **imprimir_cap2**

sCap ■ Cap : struct sCap.

▷ **imprimir_all_sts**

sState ■ Sts : struct sState.
■ m : indica quin moviment és.
sCap ■ Cap : struct sCap.
■ disc : número del disc que hem de moure.

▷ **imprimir_sts_user**

sState ■ Sts : struct sState.
■ m : indica quin moviment és.
■ disc : número del disc que hem de moure.

▷ **imprimir_move_fitxer**

FILE ■ f : declaració d'arxiu.
sState ■ Sts : struct sState.

- disc : número del disc que hem de moure.
- depth : profunditat de la matriu.

▷ **ini_mem**

sState ■ Sts : struct sState.
■■ w[NTOWERS] : vector amb el nombre de torres.
■ nd : nombre de discos.
■ Size_Sts : mida dels estats.

▷ **resize_sts**

sState ■ Sts : struct sState.
■ nd : nombre de discos.
■ Size_Sts : mida dels estats.
■ m : indica quin moviment és.

▷ **ini_sts**

sState ■ Sts : struct sState.
■ nd : nombre de discos.
■■ w[NTOWERS] : vector amb el nombre de torres.
■ m : indica quin moviment és.
■ depth : profunditat de la matriu.

▷ **ini_matrix**

■ nd : nombre de discos.
■■ w[NTOWERS] : vector amb el nombre de torres.

▷ **hanoi**

■ nd : nombre de discos.
■ towerorg : torre origen.
■ towerdest : torre destí.
■ toweraux : torre auxiliar.
sState ■ Sts : struct sState.
■■ w[NTOWERS] : vector amb el nombre de torres.
■ m : indica quin moviment és.
■ size :
■ Size_Sts : mida dels estats.
■ discs : nombre de discs de la comanda.
■ depth : profunditat de la matriu.

■ move : número del moviment.
■ disc : número del disc que hem de moure.

▷ **date_and_time_fitxer**

FILE ■ f : declaració d'arxiu.

▷ **imprimir_cap1_fitxer**

FILE ■ f : declaració d'arxiu.
sCap ■ Cap : struct sCap.

▷ **imprimir_cap2_fitxer**

FILE ■ f : declaració d'arxiu.
sCap ■ Cap : struct sCap.

▷ **imprimir_fitxer**

sState ■ Sts : struct sState.
■ m : indica quin moviment és.
sCap ■ Cap : struct sCap.
■ disc : número del disc que hem de moure.

▷ **free_mem**

sState ■ Sts : struct sState.
■■ w[NTOWERS] : vector amb el nombre de torres.
sCap ■ Cap : struct sCap.
■■ nom_fx : nom del fitxer on s'imprimeix.
■■ output : ruta de sortida.
■■ codi_op : codi d'operació del fitxer.
■ moves : nombre de moviments.

▷ **main**

■ argc : nombre d'arguments passats al programa.
■■ argv : array unidimensional de cadenes, cadascuna de les quals és un argument de argc.

▷ **NOTA:**

Totes les funcions són void, és a dir, que no retornen cap valor, exceptuant la funció main, que és de caràcter enter.

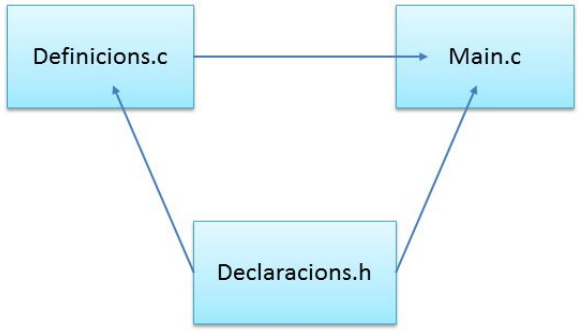
6.11 Com heu modularitzat el codi en fitxers? Aportar una breu explicació i acompanyar-la d'un dibuix conceptual dels fitxers, funcions i les seves relacions i visibilitats que ajudi explicar visualment la modularitat general del codi. Explicar per cada funció perquè està en el fitxer que està i on està visible i perquè.

La pràctica està formada per tres fitxers, explicats a continuació.

- ➔ `Definicions.c` : Inclou totes les funcions que utilitzarem en la pràctica.
- ➔ `Declaracions.h` : Inclou totes les llibreries, les definicions, les estructures de dades i les declaracions de les funcions. Està declarat en els altres dos fitxers.
- ➔ `Main.c` : Conté la funció principal main amb les crides de les funcions necessàries.

Creiem que és encertat distribuir el codi d'aquesta manera perquè així tenim: la funció principal en un arxiu on només cridem les funcions principals, les declaracions de totes les dades que necessitarem en un arxiu, i les definicions de totes les funcions en un altre fitxer. És una forma ordenada de tenir el codi i fàcil de revisar-lo en cas de voler canviar algun element o afegir-ne de nous.

Per tal d'entendre una mica millor com es relacionen entre ells, ajuntem amb aquestes línies el dibuix conceptual que vàrem realitzar per a la pràctica anterior, que presentava la mateixa composició.



6.12 Explicar el detall de la funció recursiva explicant quins paràmetres de més han fet falta i justificant per què i com ha variat el codi de la funció per estendre la seva funcionalitat.

```
void hanoi(int nd, int towerorg, int towerdest, int toweraux, sState **Sts, int *w[NTOWERS], int *m,
int size, int Size_Sts, int discs, int depth){

	depth++;
	if(nd == 1){
		move(towerorg, towerdest, Sts, w, m, size, Size_Sts, discs, depth);
	}else{
		hanoi(nd - 1, towerorg, toweraux, towerdest, Sts, w, m, size, Size_Sts, discs, depth);
		move(towerorg, towerdest, Sts, w, m, size, Size_Sts, discs, depth);
		hanoi(nd - 1, toweraux, towerdest, towerorg, Sts, w, m, size, Size_Sts, discs, depth);
	}
}
```

Per veure les diferències entre el hanoi bàsic i el hanoi adaptat a la nostra pràctica hem utilitzat només com a comparativa el que seria la funció hanoi, sense la "floritura" que hi havia al voltant. Així doncs, hem marcat en color verd claret les parts del codi que han canviat. No hem esborrat res, simplement n'hem afegit de nou.

Les parts afegides, doncs, són aquelles que provenen dels vectors dinàmics, ja que d'aquesta manera podíem adaptar el codi a la memòria de tipus dinàmic. Utilitzem l'estructura de dades sState, la matriu amb el nombre de torres, el nombre de moviment corresponent i les mesures de la matriu.

6.13 Explicar com feu la gestió de memòria que es demana i justificar perquè ho heu fet així.

La memòria dinàmica la gestionem de la següent manera:

Primer amb una funció que ens inicialitza tota la memòria, inicialitzem el vector d'estats (que es el que conté cada moviment) i la matriu ,de la qual es copien els estats i on s'aplica la funció hanoi (cal destacar que el vector d'estats es passa com a doble punter perquè si no és així, no es guarda la memòria un cop es surt de la funció).

La mida del vector d'estats és la següent:

Size_Sts = sizeof(sState) + nd*NTOWERS*sizeof(int);

Que és la mida de tots els ints que conté l'struct sState més la mida de la matriu que conté cada estat.

Un cop definides aquestes memòries a la funció hanoi per tal de poder afegir un estat hem creat una funció que es diu resize que després de fer el realloc, havíem de tornar a muntar la matriu de dins de l'estat en el que ens trobem de la següent manera:

```
/* La funció resize_states la farem servir per a donar-li, a cada moviment
que es du a terme, la mida de la matriu dins de l'struct. */
void resize_sts(sState *Sts, int size, int Size_Sts, int disc, int m){

    int i;

    for(i = 0; i < NTOWERS; i++){
        Sts[m].v[i] = (int*)calloc(disc,sizeof(int));
        if (Sts[m].v[i] == NULL){
            printf("Error: problemes de memòria");
            exit(0);
        }
    }

}
```

Si només fèiem el realloc per cada moviment cada cop que intentavem copiar l'estat de la matriu al nostre vector d'estats donava ERROR SEGMENTATION FAULT.

També es cert que utilitzem memòria dinàmica per la capçalera que se'ns imprimeix tant per pantalla com pels fitxers, la memòria de les quals gestionem igual. Tenim una funció que inicialitza la memòria necessària i després l'omplim aquesta.

Finalment al final del main tenim 4 línies de codi, una que crida una funció (free_mem) que ens allibera tota la memòria que hem reservat amb la matriu i les estructures de dades i les altres tres línies de codi són tres crides a la funció free per alliberar la memòria d'una sèrie de variables amb memòria dinàmica que tenim al main.

Amb aquesta última funció hem tingut problemes a l'hora d'implementar-la per a que funcionés correctament. El programa funcionava a la perfecció amb el debugger però, quan ens disposàvem a executar el codi sencer, petava degut a la funció que alliberava la memòria. De totes maneres, però, hem estat capaços de solucionar el problema i ara ja sabem com alliberar memòria correctament.

6.14 Quins mecanismes heu fet servir per comprovar que el codi funciona? Explicar els casos que heu fet servir i aportar per cada cas a verificar el detall concret de la comprovació. Explicar en cada cas el que es mostra i perquè. Relacionar aquesta explicació amb els fitxers de les proves aportades al directori proves. Explicar quina notació de nom de fitxers en les proves feu servir per identificar el què conté, o aporteu una taula explicant cada fitxer que conté i què representa.

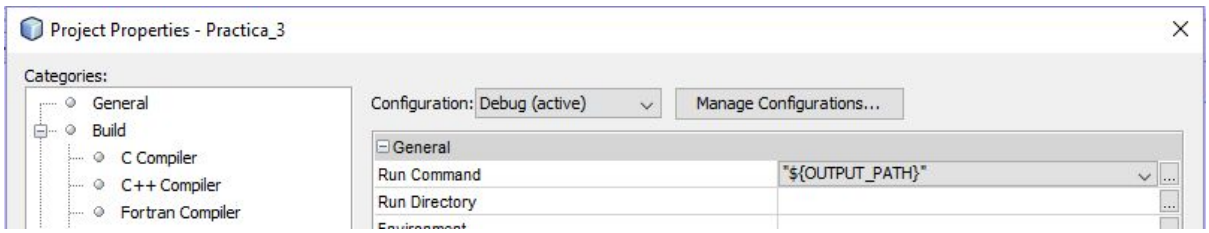
Per comprovar que el codi funciona correctament l'hem executat de cinc maneres diferents, és a dir, amb cinc crides possibles per la línia de comanda amb el NetBeans. L'estructura d'una crida havia de ser la següent:

hanoi (o el nom del programa) -d <nombre de discs> -f <nom_fitxer> -o <codi_d'operacio_fitxer>

Cada crida es difereix de la resta en els paràmetres que especifiquem segons l'estructura de dalt.

➔ Primera crida:

No especifiquem res, només cridem al programa pel seu nom i ja està. La línia de comanda, al NetBeans l'hem configurada així perquè simuli que no li hem introduït res:



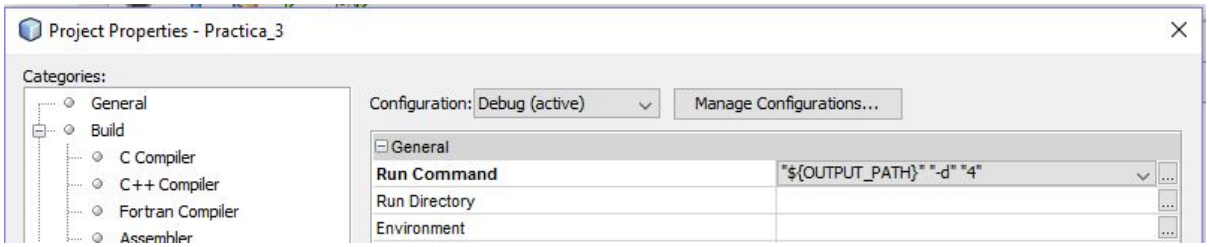
Bé, el que ens ha fet veure que el codi funciona és el fet que el programa ens imprimeix per pantalla tots els estats de les torres amb tres discs (nombre predeterminat) i després ens demana quins estats volem tornar a imprimir i res més, aquest no imprimeix res a cap fitxer ni res per l'estil. A més a més a la capçalera diu que ho imprimeix per l'stdout (pantalla).

```
==========================================
Command Line entered : /cygdrive/C/
Number of Towers     : 3
Number of Discs      : 3
Output filename      : stdout

INIT Time            : Fri Jun 17 15:49:48 2016
==========================================
```

```
Move count 7  Rec Depth 3: it moves disc 1 from TA to TB
H 2 ...| |... ..*| |*.. ...| |...
H 1 ...| |... .**| |**. ...| |...
H 0 ...| |... ***| |*** ...| |...
    ----------------------------
        A         B         C


============================================
Number of Towers   : 3
Number of Discs    : 3
Output filename    : stdout
Total Number Moves : 7

END Time           : Fri Jun 17 15:49:48 2016
============================================
Para mover 3 discos se han necesitado 7 movimientos.
A continuación, el programa le irá pidiendo el número del movimiento que desee visualizar por pantalla.
Cuando desee parar de visualizar movimientos intoduzca el 0.
¿Qué movimiento desea viusalizar?
```

➔ Segona crida:

Aquesta segona crida té la següent estructura:
`hanoi (o el nom del programa) -d 4`
Que al NetBeans cal introduir de la següent manera:



Bé, el que ens ha fet veure que el codi funciona és el fet que el programa ens imprimeix per pantalla tots els estats de les torres amb quatre discs (com li hem especificat) i després ens demana quins estats volem tornar a imprimir i res més, aquest no imprimeix res a cap fitxer ni res per l'estil. A més a més a la capçalera diu que ho imprimeix per l'stdout (pantalla).

```
============================================
Command Line entered : /cygdrive/C/
Number of Towers   : 3
Number of Discs    : 4
Output filename    : stdout

INIT Time          : Fri Jun 17 15:57:16 2016
============================================
```

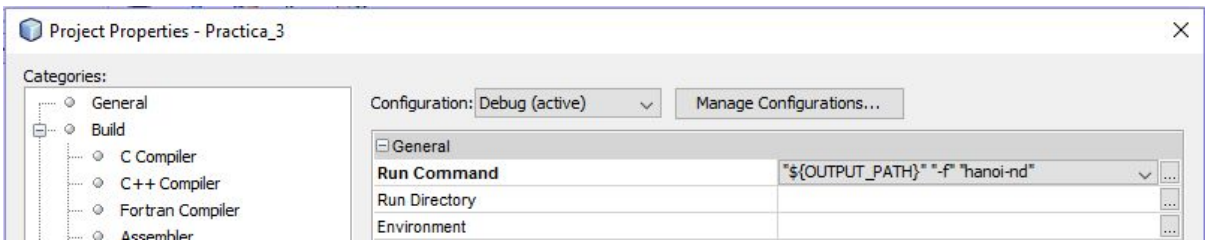
```
Move count 14  Rec Depth 3: it moves disc 2 from TA to TB
H 3 ....| |.... ....| |.... ....| |....
H 2 ....| |.... ..**| |**.. ....| |....
H 1 ....| |.... .***| |***. ....| |....
H 0 ....| |.... ****| |**** ...*| |*...
    ---------------------------------
         A          B          C

Move count 15  Rec Depth 4: it moves disc 1 from TC to TB
H 3 ....| |.... ...*| |*... ....| |....
H 2 ....| |.... ..**| |**.. ....| |....
H 1 ....| |.... .***| |***. ....| |....
H 0 ....| |.... ****| |**** ....| |....
    ---------------------------------
         A          B          C


================================================
Number of Towers   : 3
Number of Discs    : 4
Output filename    : stdout
Total Number Moves : 15

END Time           : Fri Jun 17 15:57:16 2016
================================================
Para mover 4 discos se han necesitado 15 movimientos.
A continuación, el programa le irá pidiendo el número del movimiento que desee visualizar por pantalla.
Cuando desee parar de visualizar movimientos intoduzca el 0.
¿Qué movimiento desea viusalizar?
```

➔ Tercera crida:

Aquesta tercera crida té la següent estructura:

`hanoi (o el nom del programa) -f  hanoi-nd`

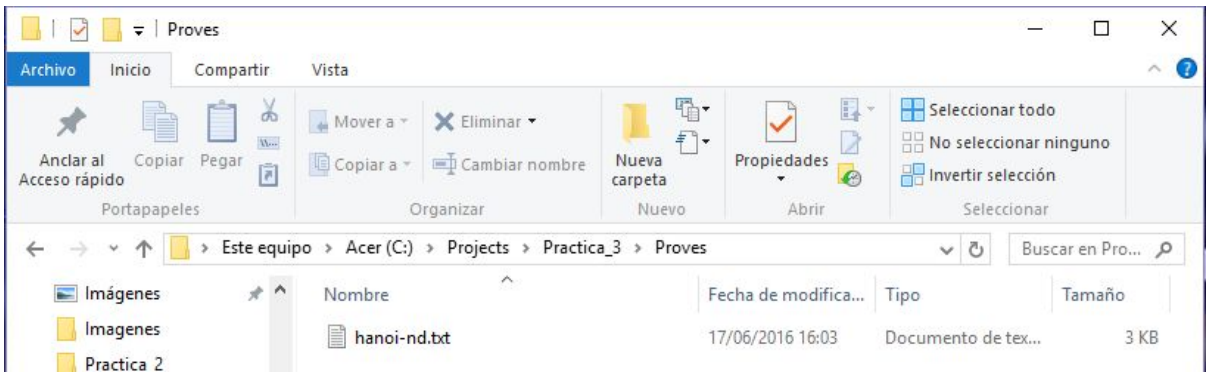
Que al NetBeans cal introduir de la següent manera:



Bé, el que ens ha fet veure que el codi funciona és el fet que el programa ens imprimeix per pantalla tots els estats de les torres amb tres discs (nombre de discs predeterminat) i després ens demana quins estats volem tornar a imprimir i a la carpeta que tenim al nostre projecte que hem anomenat proves, un cop sortim de la fase interactiva, ens tobem un fitxer amb tots els moviments impresos i que rep el nom que li hem especificat a la línia de comanda (hanoi-nd).

```
================================================
Command Line entered : /cygdrive/C/

Number of Towers   : 3
Number of Discs    : 3
Output filename    : hanoi-nd.txt

INIT Time          : Fri Jun 17 16:03:44 2016
================================================
```

```
Move count 6  Rec Depth 2: it moves disc 2 from TC to TB
H 2 ...| |... ...| |... ...| |...
H 1 ...| |... .**| |**. ...| |...
H 0 ..*| |*.. ***| |*** ...| |...
    -----------------------------
        A         B         C

Move count 7  Rec Depth 3: it moves disc 1 from TA to TB
H 2 ...| |... ..*| |*.. ...| |...
H 1 ...| |... .**| |**. ...| |...
H 0 ...| |... ***| |*** ...| |...
    -----------------------------
        A         B         C


============================================
Number of Towers    : 3
Number of Discs     : 3
Output filename     : hanoi-nd.txt
Total Number Moves : 7

END Time            : Fri Jun 17 16:03:44 2016
============================================
Para mover 3 discos se han necesitado 7 movimientos.
A continuación, el programa le irá pidiendo el número del movimiento que desee visualizar por pantalla.
Cuando desee parar de visualizar movimientos intoduzca el 0.
¿Qué movimiento desea viusalizar?
```

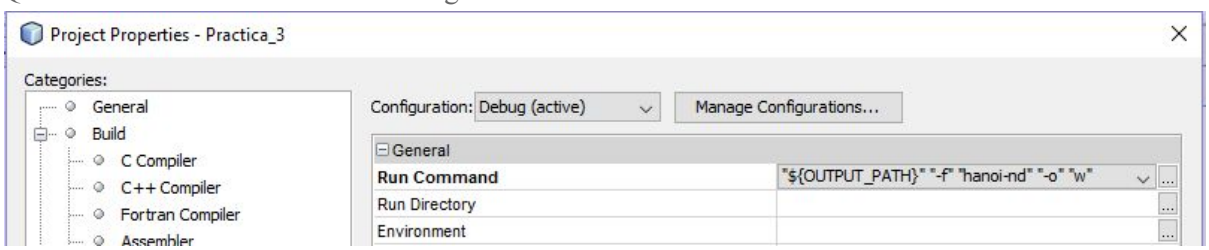


➔ Quarta crida:

Aquesta tercera crida té la següent estructura:

```
hanoi (o el nom del programa) -f  hanoi-nd -o w
```

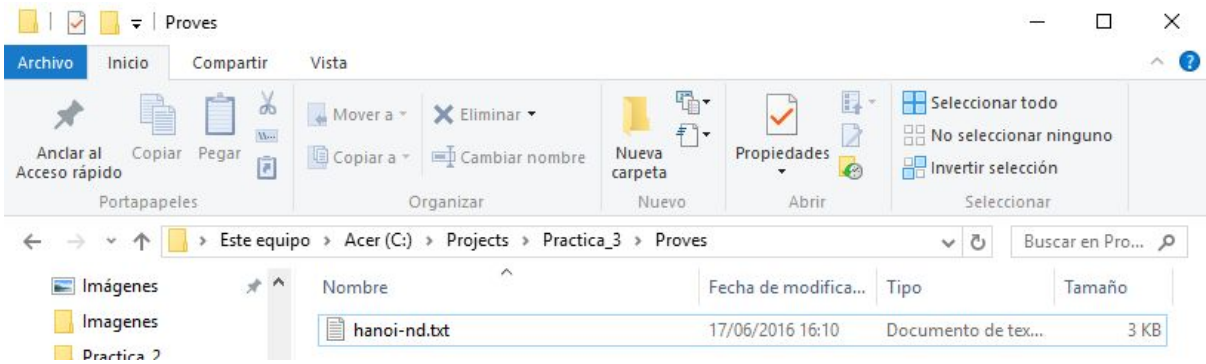
Que al NetBeans cal introduir de la següent manera:



Bé, el que ens ha fet veure que el codi funciona és el fet que el programa ens imprimeix per pantalla tots els estats de les torres amb tres discs (nombre de discs predeterminat) i després ens demana quins estats volem tornar a imprimir i a la carpeta que tenim al nostre projecte que hem anomenat proves, un cop sortim de la fase interactiva, ens tobem un fitxer amb tots els moviments impresos i que rep el nom que li hem especificat a la línia de comanda (hanoi-nd). Això es degut a que hem obert el fitxer en mode w i com que aquest fitxer ja existia, ha esborrat el contingut de l'altre i ha posat el contingut d'aquesta crida.

```
==========================================
Command Line entered : /cygdrive/C/

Number of Towers    : 3
Number of Discs     : 3
Output filename     : hanoi-nd.txt

INIT Time           : Fri Jun 17 16:10:10 2016
==========================================
```

```
Move count 6  Rec Depth 2: it moves disc 2 from TC to TB
H 2 ...| |... ...| |... ...| |...
H 1 ...| |... .**| |**. ...| |...
H 0 ..*| |*.. ***| |*** ...| |...
   ---------------------------
       A         B         C

Move count 7  Rec Depth 3: it moves disc 1 from TA to TB
H 2 ...| |... ..*| |*.. ...| |...
H 1 ...| |... .**| |**. ...| |...
H 0 ...| |... ***| |*** ...| |...
   ---------------------------
       A         B         C


==========================================
Number of Towers    : 3
Number of Discs     : 3
Output filename     : hanoi-nd.txt
Total Number Moves : 7

END Time            : Fri Jun 17 16:10:10 2016
==========================================
Para mover 3 discos se han necesitado 7 movimientos.
A continuación, el programa le irá pidiendo el número del movimiento que desee visualizar por pantalla.
Cuando desee parar de visualizar movimientos intoduzca el 0.
¿Qué movimiento desea viusalizar?
```



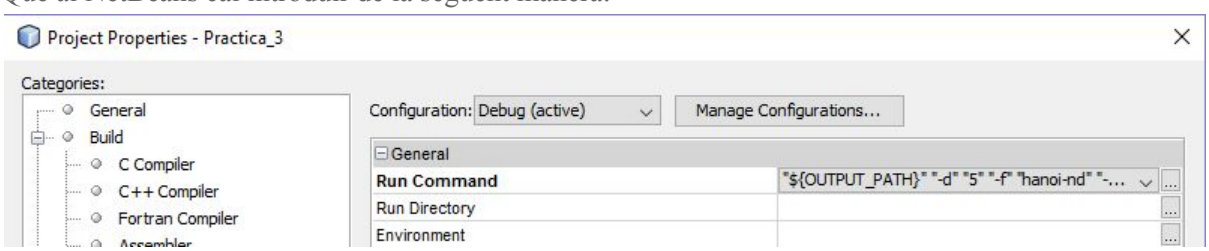
*Cal fixar-se que la hora de última modificació no és la mateix que en la captura d'abans

➔ Cinquena crida:

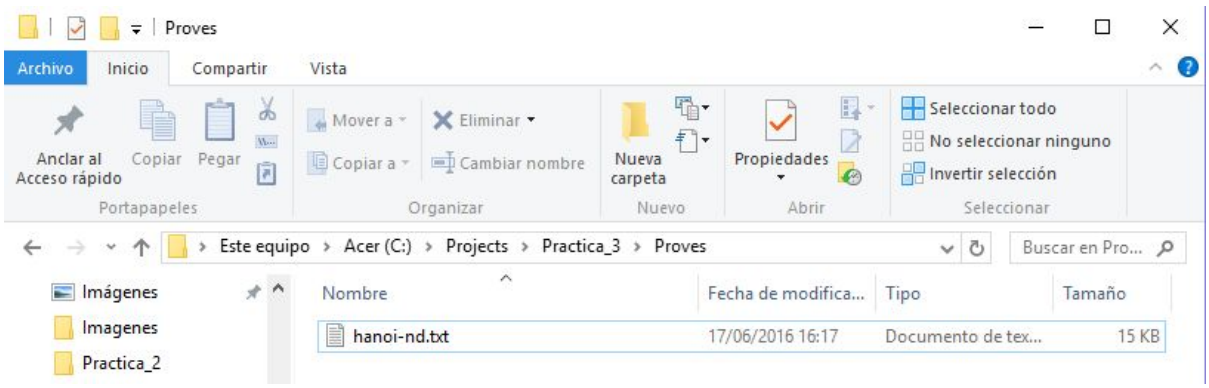
Aquesta cinquena crida té la següent estructura:

```
hanoi (o el nom del programa) -d 5 -f  hanoi-nd -o ap
```

Que al NetBeans cal introduir de la següent manera:



Bé, el que ens ha fet veure que el codi funciona és el fet que el programa ens imprimeix per pantalla tots els estats de les torres amb cinc discs (com hem especificat) i després ens demana quins estats volem tornar a imprimir i a la carpeta que tenim al nostre projecte que hem anomenat proves, un cop sortim de la fase interactiva, ens tobem un fitxer amb tots els moviments impresos i que rep el nom que li hem especificat a la línia de comanda (hanoi-nd). Això es degut a que hem obert el fitxer en mode ap i com que aquest fitxer ja existia, ha afegit el contingut d'aquesta crida al final del fitxer.



*Cal fixar-se que la hora de ultima modificació no és la mateix que en la captura d'abans

Finalment l'únic arxiu .txt que adjuntem a la carpeta Proves és el hanoi-nd.txt que conté 2 execucions impreses al fitxer:

1. Una amb tres discs (i no són dos perquè recordem que la tercera crida que hem fet l'hem esborrat perquè a la quarta crida ho hem fet amb el mateix nom de fitxer i amb l'opció d'obrir fitxer w).
2. Una amb 5 discs (per la cinquena crida).

Lògicament, aquestes proves no ens han sortit perfectes al primer intent així que aquí citarem una sèrie d'errors que ens ha anat sortint i que creiem que ens han estat útils per millorar:

1. **SIGNAL ABORTED :** A la funció free memory. Ens ha fet veure que hi havia alguns punters no ben definits pel que fa a la memòria dinàmica i dels quals no podia alliberar la seva memòria.

2. SEGMENTATION FAULT : Aquest famós error entre nosaltres ens ha aparegut en varies ocasions. Una de les que més recordem va ser quan intentàvem redimensionar l'arrel d'estats amb el nombre de discs de la funció hanoi que anava canviant. Per això quan intentàvem copiar l'estat de la matriu a una posició del nostre vector ens apareixia aquest error, perqué l'espai que hi havia reservat era més petit del que intentàvem copiar. També ens ha aparegut en una part del codi on introduïem un index i a Cap per cridar el fitxer. Aquest vector només tenia una posició, però com que se'ns va oblidar indicar que i valgués 0, aquell Cap[i] no existia perquè el valor de i ves a saber quin era.

6.15 Per tots els punts anteriors comentar coses que heu provat i no us hagi funcionat i explicar què heu après en aquest procés de programació no satisfactòria. Podeu fer una recopilació (ordenada i entenedora) de totes les coses aquí o incloure aquest aspecte en la resposta de tots els apartats anterior.

Durant el procés de fer funcionar el programa, ens hem trobat amb diferents aspectes que ens han ajudat a aprendre a millorar a partir de resultats no satisfactoris. El recull de cadascú d'aquests ha estat incorporat en cada apartat segons crèiem convenient.

6.16 Estat de l'entrega: Per cada criteri d'avaluació indicar l'estat de l'entrega de cada criteri abordant els tres sub-criteris transversals en cada cas. Podeu copiar la taula de criteris d'avaluació i omplir la columna d'estat de l'entrega que tal i com indica, s'ha d'omplir pels alumnes en l'entrega. S'espera que indiqueu si funciona i sinó expliqueu quins problemes heu tingut, o quines proves heu fet i no heu pogut solucionar, o qualsevol aspecte rellevant referent als 4 sub-criteris que pugui ajudar entendre i interpretar millor l'estat del programa i esforç realitzat per la correcció.

	Concepte	Func.?	Codi enten.?	Informe i disseny
1	Comprensió de la recursivitat (preguntes 1-5)			Hem entès el concepte de recursivitat d'aquest problema i hem respost les preguntes requerides amb les imatges corresponents, les quals ens han ajudat molt.
2	Estructura de dades i variables	✓	✓	Les dues estructures de dades i les variables declarades creiem que són correctes, simples i entenedores.
3	Funció recursiva necessària en el programa	✓	✓	La funció recursiva que venia predeterminada (hanoi) ha estat modificada per tal que s'adapti a qualsevol nombre de discos que introdueix l'usuari. Per això hem utilitzat paràmetres com size (mida de la matriu) i depth (profunditat de la matriu) amb el fi de generar una matriu característica pel joc segons el número de discos que introdueix l'usuari.
4	Gestió de memòria	✓	✓	La funcionalitat de guardar les dades en memòria dinàmica la utilitzem en moltes de les funcions, com exigia la pràctica. Així doncs, la funció encarregada d'alliberar-la, un cop acabat el joc, és free_mem.
5	Entrada de dades via paràmetres del programa	✓	✓	El pas per paràmetres funciona i està justificat en l'apartat 6.10.
6	Verificació de les dades entrades	✓	✓	El programa té dos punts on comprova si l'usuari ha introduït les dades correctament. En primer lloc, quan introdueix per la command line la comanda que executa el programa i, en segon lloc, durant la fase interactiva, on s'imprimeix un missatge d'error si l'usuari introdueix un valor fora del rang possible de moviments.
7	Sortida de dades a fitxer	✓	✓	El fitxer de sortida que es crea, amb les capçaleres i els moviments impresos un a un, és clar i ordenat. Tot correcte.
8	Entrada de dades en fase interactiva	✓	✓	Els moviments estan correctament numerats atès que, d'aquesta manera, mostrar a l'usuari l'X moviment que vol visualitzar és molt més senzill al estar guardar en memòria.
9	Visualització dels moviments en la fase interactiva	✓	✓	Els moviments es mostren correctament en la fase interactiva, de la mateixa manera que s'indicava en l'enunciat de la pràctica.

10	Modularitat en funcions	✓	✓	Les funcions estan separades entre elles de la següent manera: capçalera-explicació-funció-...-explicació-funció-.
11	Modularitat del codi (llibreries)	✓	✓	El programa està separat en tres arxius, tal i com veiem a l'apartat 6.11.
12	Verificació del codi (traces i pla de proves)	✓	✓	La verificació del codi la hem realitzat nosaltres mateixos repetint el procés d'execució moltes vegades i canviant sempre el nombre de discos fins que no donés cap error.
13	Explicació dels casos no satisfactoris i aprenentatge que se n'ha derivat	✓	✓	A l'hora de recrear amb caràcters les torres i els discos, vam tenir grans idees, però durant la implementació ens vam adonar que el NetBeans no suportava els caràcters utilitzats, així que vam haver de canviar-los per uns de més "bàsics". També hem tingut algun problema amb la duplicació de codi: no hem sabut imprimir per pantalla i alhora en fitxer el mateix codi en una sola funció sense repetir el text.

Satisfacció de la pràctica

6.17 (Opcional) Què us ha semblat la pràctica des del punt de vista de l'aprenentatge, utilitat, temàtica, durada, dificultat, interès, motivació...? Comentaris i recomanacions.

Un cop acabada la pràctica anterior, la qual vam criticar força, trobar-nos amb aquesta nova temàtica relacionada amb un joc d'enginy ens va fer saltar d'alegria, metafòricament, és clar.

Ens ha semblat una pràctica molt entretinguda a nivell de pensar com aconseguir la recursivitat que demanava l'enunciat, a l'hora de crear els morals dels arbres recursius i, fins i tot, a l'hora de programar. També ens hem ajudat de pàgines que permeten jugar al hanoi on-line, on nosaltres escollim el nombre de discos i el nombre de pilars és sempre 3, complint les mateixes característiques que la nostra pràctica, per a entendre perfectament el funcionament del joc.

Potser ha estat una mica complicada la part de la memòria dinàmica, ja que no la dominàvem gaire però, en general, tot ha anat prou bé. Realment aconsellem que aquesta pràctica es mantingui per l'any vinent, donat que creiem que als nostres futurs companys de carrera els hi agradarà tant com a nosaltres i és una bona forma de compensar la possible mala nota de la pràctica anterior.