Pràctica 1. Calculadora recursiva

Arey Ferrero Ramos

22 de gener del 2019

Índex

Jocs de proves 2
Primera etapa: Llegir dades i mostrar-les 2
Segona etapa: Comprovar correctesa de les dades: 4
Tercera etapa: Càlcul de l'expressió 8
Decisions de disseny 11
Qüestions a valorar 16
Problemes sorgits durant el desenvolupament de la pràctica 17
Conclusions 20
Bibliografia 22

Jocs de proves

Primera etapa: Llegir dades i mostrar-les

Primer de tot s'ha de comprovar si la lectura de dades es correcta. Per a provar-ho, s'ha dissenyat una funció que permeti escriure per pantalla totes les dades llegides del fitxer. La funció és cridarà des del main amb la següent línia de codi:

```
EscriureExpressio(Expressio);
```

La funció és la següent:

```
void EscriureExpressio(char Expressio[])
{
    int i=0;

    while (Expressio[i]!='\0') {
        printf("%c", Expressio[i]);
        i++;
    }
    printf("\n");
}
```

Prova	Descripció	Sortida esperada	Sortida real
1	El fitxer no existeix.	`El fitxer no existeix.`	`El fitxer no existeix.`

2	Prova de fum. Es llegeix el fitxer tal com s'ha descarregat del Moodle, sense	`((2+3)*4+1)))`
		`f*(2+4)`
		`5(3+2)`
		`5*(3++2)`
		`2*3`
		`(3+5)*(2-4)`
		`(3*(10+2))+((5+4)*(20-2))`
		`35-(12*8)`
		`(1000-500)*(2*1000)`
			`((2+3)*4+1)))`
			`f*(2+4)`
			`5(3+2)`
			`5*(3++2)`
			`2*3`
			`(3+5)*(2-4)`
			`(3*(10+2))+((5+4)*(20-2))`
			`35-(12*8)`
			`(1000-500)*(2*1000)`

	realitzar-li cap modificació.	(((10+2)+5)+5)*(20-2) (20-2)*(((10+2)+5)+5) 20/(35/7) 20/(12/(3*2)) 20/(4-(2*2))	(((10+2)+5)+5)*(20-2) (20-2)*(((10+2)+5)+5) 20/(35/7) 20/(12/(3*2)) 20/(4-(2*2))
3	S'afegeix una operació més i s'augmenta el número d'expressions donat al principi del fitxer.	((2+3)*4+1))) f*(2+4) 5(3+2) 5*(3++2) 2*3 (3+5)*(2-4) (3*(10+2))+((5+4)*(20-2)) 35-(12*8) (1000-500)*(2*1000) (((10+2)+5)+5)*(20-2) (20-2)*(((10+2)+5)+5) 20/(35/7) 20/(12/(3*2)) 20/(4-(2*2)) 40/((10-3)*8)	((2+3)*4+1))) f*(2+4) 5(3+2) 5*(3++2) 2*3 (3+5)*(2-4) (3*(10+2))+((5+4)*(20-2)) 35-(12*8) (1000-500)*(2*1000) (((10+2)+5)+5)*(20-2) (20-2)*(((10+2)+5)+5) 20/(35/7) 20/(12/(3*2)) 20/(4-(2*2)) 40/((10-3)*8)
4	S'afegeix una operació més sense augmentar el número d'expressions donat al principi del fitxer.	((2+3)*4+1))) f*(2+4) 5(3+2) 5*(3++2) 2*3 (3+5)*(2-4) (3*(10+2))+((5+4)*(20-2)) 35-(12*8) (1000-500)*(2*1000) (((10+2)+5)+5)*(20-2) (20-2)*(((10+2)+5)+5) 20/(35/7) 20/(12/(3*2)) 20/(4-(2*2)) El nombre d'expressions donat al inici del programa es inferior al nombre d'expressions que realment te el programa.	((2+3)*4+1))) f*(2+4) 5(3+2) 5*(3++2) 2*3 (3+5)*(2-4) (3*(10+2))+((5+4)*(20-2)) 35-(12*8) (1000-500)*(2*1000) (((10+2)+5)+5)*(20-2) (20-2)*(((10+2)+5)+5) 20/(35/7) 20/(12/(3*2)) 20/(4-(2*2)) El nombre d'expressions donat al inici del programa es inferior al nombre d'expressions que realment te el programa.
5	S'elimina una operació del fitxer i es redueix el número d'expressions donat al principi del fitxer.	((2+3)*4+1))) f*(2+4) 5(3+2) 5*(3++2) 2*3 (3+5)*(2-4) (3*(10+2))+((5+4)*(20-2)) 35-(12*8) (1000-500)*(2*1000) (((10+2)+5)+5)*(20-2)	((2+3)*4+1))) f*(2+4) 5(3+2) 5*(3++2) 2*3 (3+5)*(2-4) (3*(10+2))+((5+4)*(20-2)) 35-(12*8) (1000-500)*(2*1000) (((10+2)+5)+5)*(20-2)

		(20-2)*(((10+2)+5)+5) 20/(35/7) 20/(12/(3*2))	(20-2)*(((10+2)+5)+5) 20/(35/7) 20/(12/(3*2))
6	S'elimina una operació del fitxer sense reduir el número d'expressions donat al principi del fitxer. No s'imprimirà l'ultima línia.	((2+3)*4+1))) f*(2+4) 5(3+2) 5*(3++2) 2*3 (3+5)*(2-4) (3*(10+2))+((5+4)*(20-2)) 35-(12*8) (1000-500)*(2*1000) (((10+2)+5)+5)*(20-2) (20-2)*(((10+2)+5)+5) El nombre d'expressions donat al inici del programa es superior al nombre d'expressions que realment te el programa.	(((2+3)*4+1))) f*(2+4) 5(3+2) 5*(3++2) 2*3 (3+5)*(2-4) (3*(10+2))+((5+4)*(20-2)) 35-(12*8) (1000-500)*(2*1000) (((10+2)+5)+5)*(20-2) (20-2)*(((10+2)+5)+5) El nombre d'expressions donat al inici del programa es superior al nombre d'expressions que realment te el programa.

Després de comprovar que les dades emmagatzemades en el fitxer s'han llegit de manera correcta, s'ha eliminat tant el codi corresponent a la funció `EscriureExpressio` com la instrucció de crida a la mateixa degut a que no forma part de les especificacions del programa. No obstant, degut a que el seu codi s'ha incorporat en la documentació, sempre es podrà traslladar al programa per a fer les comprovacions pertinents. Tot i així, s'ha de tenir en compte que serà recomanable comentar altres parts del programa.

Segona etapa: Comprovar correctesa de les dades:

Seguidament, s'haurà de comprovar si les funcions dissenyades per a fer el tractament d'errors son correctes abans de fer el càlcul del resultat de les expressions que si que ho son. Si l'expressió passa el control d'errors s'imprimirà per pantalla un missatge de verificació de la següent manera:

```
printf("Totes les dades de l'expressio %i son correctes.\n", cont_expressions+1);
```

Degut a que s'ha fet un tractament dels errors deguts a un veïnatge de símbols incorrecte més exhaustiu que el que s'exigia en l'enunciat de la pràctica, s'ha modificat

el fitxer de dades per a tenir un joc de proves també més exhaustiu que provi una major quantitat d'errors, amb excepció de en la prova de fum, en la que es prova el fitxer de dades original sense cap modificació. Tot i així, s'ha de tenir en compte que difícilment es podran tractar tots els cassos possibles. S'afegiran dotze expressions al fitxer, que son les següents:

```
)+(((5*3)+10)/4)(
(((5+5)*8))+20)(
*((10+2)/6)*(4+4)
((5+7)*3)-
((((7*8)/4)//2)--10)
((((7**8)/4)//2)-10)
((((7*8)/4)//2)-10)
10*((+5)-3)
4*((7-7/)+2)
50*((3-2)5+8)
((8+30)(8/2))
((19*9)-10)+()+4*5
(17+8)+(5)+(3*4)
```

Prova	Descripció	Sortida esperada	Sortida real
1	Prova de fum.	L'expressió 1 no es valida perquè la distribució dels parèntesis no es coherent. L'expressió 2 no es valida perquè els símbols de l'expressió no son vàlids. L'expressió 3 no es valida perquè el veïnatge dels símbols no es correcte. L'expressió 4 no es valida perquè el veïnatge dels símbols no es correcte. Totes les dades de l'expressió 5 son correctes.	L'expressió 1 no es valida perquè la distribució dels parèntesis no es coherent. L'expressió 2 no es valida perquè els símbols de l'expressio no son vàlids. L'expressio 3 no es valida perquè el veïnatge dels símbols no es correcte. L'expressio 4 no es valida perquè el veïnatge dels símbols no es correcte. Totes les dades de l'expressió 5 son correctes.

		Totes les dades de l'expressió 6 son correctes. Totes les dades de l'expressió 7 son correctes. Totes les dades de l'expressió 8 son correctes. Totes les dades de l'expressió 9 son correctes. Totes les dades de l'expressió 10 son correctes. Totes les dades de l'expressió 11 son correctes. Totes les dades de l'expressió 12 son correctes. Totes les dades de l'expressió 13 son correctes. Totes les dades de l'expressió 14 son correctes.	Totes les dades de l'expressió 6 son correctes. Totes les dades de l'expressió 7 son correctes. Totes les dades de l'expressió 8 son correctes. Totes les dades de l'expressió 9 son correctes. Totes les dades de l'expressió 10 son correctes. Totes les dades de l'expressió 11 son correctes. Totes les dades de l'expressió 12 son correctes. Totes les dades de l'expressió 13 son correctes. Totes les dades de l'expressió 14 son correctes.
2	S'afegeixen 12 expressions addicionals al fitxer d'expressió per a fer una comprovació més exhaustiva del correcte funcionament del tractament d'errors deguts a un veïnatge de símbols erroni.	L'expressió 1 no es vàlida perquè la distribució dels parèntesis no es coherent. L'expressió 2 no es vàlida perquè els símbols de L'expressió no son vàlids. L'expressió 3 no es vàlida perquè el veïnatge dels símbols no es correcte. L'expressió 4 no es vàlida perquè el veïnatge dels símbols no es correcte. L'expressió 5 no es vàlida perquè el veïnatge dels símbols no es correcte. L'expressió 6 no es vàlida perquè el veïnatge dels símbols no es correcte. L'expressió 7 no es vàlida perquè el	L'expressio 1 no es valida perquè la distribució dels parèntesis no es coherent. L'expressio 2 no es valida perquè els símbols de l'expressio no son vàlids. L'expressio 3 no es valida perquè el veïnatge dels símbols no es correcte. L'expressio 4 no es valida perquè el veïnatge dels símbols no es correcte. L'expressio 5 no es valida perquè el veïnatge dels símbols no es correcte. L'expressio 6 no es valida perquè el veïnatge dels símbols no es correcte. L'expressio 7 no es valida perquè el veïnatge dels símbols no es correcte.

		veïnatge dels símbols no es correcte. L'expressió 8 no es vàlida perquè el veïnatge dels símbols no es correcte. L'expressió 9 no es vàlida perquè el veïnatge dels símbols no es correcte. L'expressió 10 no es vàlida perquè el veïnatge dels símbols no es correcte. L'expressió 11 no es vàlida perquè el veïnatge dels símbols no es correcte. L'expressió 12 no es vàlida perquè el veïnatge dels símbols no es correcte. L'expressió 13 no es vàlida perquè el veïnatge dels símbols no es correcte. L'expressió 14 no es vàlida perquè el veïnatge dels símbols no es correcte. L'expressió 15 no es vàlida perquè el veïnatge dels símbols no es correcte. L'expressió 16 no es vàlida perquè el veïnatge dels símbols no es correcte. L'expressió 17 no es vàlida perquè el veïnatge dels símbols no es correcte. Totes les dades de l'expressió 18 son correctes. Totes les dades de l'expressió 19 son correctes. Totes les dades de l'expressió 20 son correctes. Totes les dades de l'expressió 21 son correctes.	L'expressio 8 no es valida perquè el veïnatge dels símbols no es correcte. L'expressio 9 no es valida perquè el veïnatge dels símbols no es correcte. L'expressio 10 no es valida perquè el veïnatge dels símbols no es correcte. L'expressio 11 no es valida perquè el veïnatge dels símbols no es correcte. L'expressio 12 no es valida perquè el veïnatge dels símbols no es correcte. L'expressio 13 no es valida perquè el veïnatge dels símbols no es correcte. L'expressio 14 no es valida perquè el veïnatge dels símbols no es correcte. L'expressio 15 no es valida perquè el veïnatge dels símbols no es correcte. L'expressio 16 no es valida perquè el veïnatge dels símbols no es correcte. L'expressió 17 no es vàlida perquè el veïnatge dels símbols no es correcte. Totes les dades de l'expressió 18 son correctes. Totes les dades de l'expressió 19 son correctes. Totes les dades de l'expressió 20 son correctes. Totes les dades de l'expressió 21 son correctes. Totes les dades de l'expressió 22 son correctes.

		Totes les dades de l'expressió 22 son correctes. Totes les dades de l'expressió 23 son correctes. Totes les dades de l'expressió 24 son correctes. Totes les dades de l'expressió 25 son correctes. Totes les dades de l'expressió 26 son correctes. Totes les dades de l'expressió 27 son correctes.	Totes les dades de l'expressió 23 son correctes. Totes les dades de l'expressió 24 son correctes. Totes les dades de l'expressió 25 son correctes. Totes les dades de l'expressió 26 son correctes. Totes les dades de l'expressió 27 son correctes.

Tant les expressions addicionals que s'han introduït en el fitxer com la instrucció per a imprimir un missatge per pantalla s'han eliminat un cop feta la comprovació. No obstant, totes les dades necessàries per a fer les comprovacions pertinents s'ha incorporat en la documentació i, per tant, sempre es podran traslladar al programa. Novament, s'ha de tenir en compte que per a fer les comprovacions serà recomanable comentar algunes parts del programa i modificar el nombre d'instruccions indicat en la primera línia del fitxer de dades.

Tercera etapa: Càlcul de l'expressió

Finalment, un cop s'ha garantit que només arribaran a ser avaluades aquelles expressions que tenen un format correcte, s'ha de comprovar si el càlcul del resultat de l'expressió és correcte. Es a dir, s'ha de comprovar el correcte funcionament de la funció recursiva `Calcula`. Degut a que el nombre d'expressions contingudes en el fitxer per a comprovar si el funcionament de la funció `Calcula` ja és prou extens, únicament s'ha afegit una expressió addicional, excepte en la prova de fum.

```
(4*(20-(2*(3+5))))*2
```

El motiu d'afegir aquesta expressió és perquè únicament s'ha pogut comprovar el comportament de la funció en trobar-se amb un conjunt de parèntesis tancats consecutius en una de les expressions, comprovació que no es considera suficient per a garantir el correcte funcionament de l'algorisme en aquest aspecte.

Prova	Descripció	Sortida esperada	Sortida real
1	Prova de fum.	L'expressio 1 no es valida perquè la distribució dels parèntesis no es coherent. L'expressio 2 no es valida perquè els símbols de l'expressio no son vàlids. L'expressio 3 no es valida perquè el veïnatge dels símbols no es correcte. L'expressio 4 no es valida perquè el veïnatge dels símbols no es correcte. El resultat de l'expressió 5 es: 6 El resultat de l'expressió 6 es: -16 El resultat de l'expressió 7 es: 198 El resultat de l'expressió 8 es: -61 El resultat de l'expressió 9 es: 1000000 El resultat de l'expressió 10 es: 396 El resultat de l'expressió 11 es: 396 El resultat de l'expressió 12 es: 4 El resultat de l'expressió 13 es: 10 El resultat de l'expressió 14 no existeix.	L'expressio 1 no es valida perquè la distribució dels parèntesis no es coherent. L'expressio 2 no es valida perquè els símbols de l'expressio no son vàlids. L'expressio 3 no es valida perquè el veïnatge dels símbols no es correcte. L'expressio 4 no es valida perquè el veïnatge dels símbols no es correcte. El resultat de l'expressió 5 es: 6 El resultat de l'expressió 6 es: -16 El resultat de l'expressió 7 es: 198 El resultat de l'expressió 8 es: -61 El resultat de l'expressió 9 es: 1000000 El resultat de l'expressió 10 es: 396 El resultat de l'expressió 11 es: 396 El resultat de l'expressió 12 es: 4 El resultat de l'expressió 13 es: 10 El resultat de l'expressió 14 no existeix.
2	S'afegeixen 1 expressió addicional al fitxer de dades	L'expressió 1 no es valida perquè la distribució dels parèntesis no es coherent.	L'expressió 1 no es valida perquè la distribució dels parèntesis no es coherent.

	per a fer una comprovació del comportament de la funció en trobar-se un conjunt de parèntesis tancats consecutius.	L'expressió 2 no es valida perquè els símbols de l'expressi no son vàlids. L'expressió 3 no es valida perquè el veïnatge dels símbols no es correcte. L'expressió 4 no es valida perquè el veïnatge dels símbols no es correcte. El resultat de l'expressió 5 es: 6 El resultat de l'expressió 6 es: -16 El resultat de l'expressió 7 es: 198 El resultat de l'expressió 8 es: -61 El resultat de l'expressió 9 es: 1000000 El resultat de l'expressió 10 es: 396 El resultat de l'expressió 11 es: 396 El resultat de l'expressió 12 es: 4 El resultat de l'expressió 13 es: 10 El resultat de l'expressió 14 no existeix. El resultat de l'expressió 15 es: 32	L'expressió 2 no es valida perquè els símbols de L'expressió no son vàlids. L'expressio 3 no es valida perquè el veïnatge dels símbols no es correcte. L'expressió 4 no es valida perquè el veïnatge dels símbols no es correcte. El resultat de l'expressió 5 es: 6 El resultat de l'expressió 6 es: -16 El resultat de l'expressió 7 es: 198 El resultat de l'expressió 8 es: -61 El resultat de l'expressió 9 es: 1000000 El resultat de l'expressió 10 es: 396 El resultat de l'expressió 11 es: 396 El resultat de l'expressió 12 es: 4 El resultat de l'expressió 13 es: 10 El resultat de l'expressió 14 no existeix. El resultat de l'expressió 15 es: 32

Un cop feta la comprovació, s'ha eliminat la instrucció addicional del fitxer que conté les dades. No obstant, es troba en la documentació per si es volen fer les comprovacions pertinents. En aquest cas, no s'haurà de fer cap modificació sobre el programa però si s'haurà d'actualitzar nombre d'instruccions indicat en la primera línia del fitxer de dades.

Decisions de disseny

El programa s'ha desenvolupat seguint la técnica de disseny descendent, és a dir, mitjançant la crida a procediments (accions y funcions) per avaluar les diferents parts especificades en l'enunciat.

En el programa principal s'ha realitzat la creació de la variable simbòlica que permetrà la lectura de les dades d'un fitxer, l'enllaç d'aquesta variable amb el fitxer en qüestió (la obertura del fitxer) i el tancament del fitxer. També es tracten els possibles errors que puguin aparèixer durant la lectura de les dades: Que el fitxer no existeixi i que el número de expressions que s'indica que tindrà el fitxer en l'inici del programa no es correspongui amb el nombre d'expressions que realment té el programa. Si el fitxer no existeix, es evident que no es podrà llegir cap dada. En conseqüència, es mostrarà un missatge d'error per pantalla. D'altra banda, si es detecta que el número de expressions que s'indica que tindrà el fitxer en l'inici del programa no es correspon amb el nombre d'expressions que realment té el programa es llegiran i es tractaran igualment totes les dades fins que es detecti l'error, moment en el que es mostrarà per pantalla el missatge d'error corresponent. Per a realitzar el tractament d'aquest últim error, s'ha implementat un comptador que es va actualitzant cada vegada que s'ha llegit i tractat una expressió fins que el seu valor es correspon amb el del número indicat en l'inici del fitxer. En aquest punt, si l'últim caràcter que s'ha llegit és el caràcter de salt de línia enlloc del caràcter de fi de fitxer, això implica que, en principi, encara quedarien expressions per tractar (o que s'ha afegit un caràcter de salt de línia al final de la última expressió que no hauria d'estar). Per altra banda, si s'ha llegit el caràcter de fi de fitxer però el valor del comptador d'expressions es inferior al número d'expressions que suposadament tindrà el fitxer (número llegit en la primera línia del fitxer), això implica que el fitxer te menys expressions que les indicades. Per últim, la crida a alguns dels procediments encarregats de llegir les dades del fitxer i de fer tot el tractament d'aquestes dades també es realitzarà des del programa principal.

L'acció `LlegirDades` serà l'encarregada de llegir les dades del fitxer. El seu funcionament és molt senzill i està estandarditzat. S'ha de destacar que no s'ha utilitzat cap funció de la capçalera String.h, la qual pertany a la biblioteca estàndard de c. En canvi, es llegeixen, un per un, tots els caràcter d'una línia fins arribar al caràcter de salt de línia i s'emmagatzemen de manera ordenada en una taula de caràcters.

L'acció `TractarDades` s'encarrega de cridar a cada una de les funcions de tractament d'errors, cada una de les quals avaluarà aspectes diferents de l'expressió per a determinar si aquesta és correcta o no. Si una d'aquestes funcions determina que l'expressió és incorrecta , s'imprimirà un missatge d'error corresponent al tipus d'error detectat. La taula en la que s'emmagatzemen tots els símbols que s'acceptaran com a vàlids en el programa es defineix en aquesta acció. Finalment, en cas que l'expressió hagi superat totes les comprovacions d'errors, l'acció `TractarDades` cridarà a la funció encarregada de calcular el resultat i imprimirà aquest resultat per pantalla, excepte en

el cas de que rebi el valor '-1000000' el qual s'ha seleccionat com a indicador de que l'expressió no es pot resoldre degut a que, en algun moment, s'havia de portar a terme una divisió en la que el denominador era 0. L'elecció d'aquest valor ha sigut completament arbitrària.

La funció `SimbolsValids` és la primera funció de comprovació d'errors i s'encarrega de determinar si els símbols de la expressió formen part del conjunt símbols que s'han determinat com a vàlids en l'enunciat de la pràctica. La seva implementació s'ha portat a terme seguint al detall les especificacions de l'enunciat de la pràctica. Es recorre la taula que conté l'expressió i, per cada element de l'expressió, es recorre la taula que conté tots els símbols preseleccionats com a vàlids per a saber si el símbol que s'està avaluant es troba entre ells. Es va plantejar la possibilitat de utilitzar una crida a la funció `Comparar` (que s'explicarà més endavant) per a aconseguir que aquesta funció fos molt més òptima. El codi hagués estat el següent:

```
bool SimbolsValids (char Expressio[], char Simbols[])
{
    int i=0;
    bool valida=true;

    while ((valida)&&(Expressio[i]!='\0')) {
          valida=!(Comparar(Expressio, Simbols, i, 0,
          MAX_SIMBOLS));
          i++;
    }
    return valida;
}
```

El programa funcionava sense cap error. No obstant, degut a que la funció `Comparar` s'ha dissenyat com una extensió de la funció `VeinatgeValid` (aplicant la tècnica de disseny descendent), la seva utilització des de la funció `SimbolsValids` pot generar confusió degut a que aquesta utilitza variables amb el mateix nom que algunes de les que utilitza la funció `Comparar` però amb significats diferents. En conseqüència, el codi que es mostra en la versió final del programa és més clar i entenedor tot i tenir un major nombre d'instruccions.

La funció `ParentesisValids` és la segona funció de comprovació d'errors i s'encarrega de determinar si hi ha coherència entre els parèntesis de l'expressió, és a dir, si el nombre de parèntesis oberts es correspon amb el nombre de parèntesis tancats. La seva implementació també s'ha portat a terme seguint al detall les especificacions de l'enunciat de la pràctica. Es defineix una variable entera que farà la funció de comptador i es recorre la taula que conté l'expressió. Quan es detecta un parèntesi obert, s'incrementa el comptador, mentre que quan es detecta un parèntesi tancat, es

decrementa el comptador. Si, al final, el comptador és 0; els parèntesis de l'expressió son coherents.

La funció `VeinatgeValid` és la tercera funció de comprovació d'errors i s'encarrega de determinar si el veïnatge de símbols és correcte o no. Aquesta funció ha estat la que ha tingut l'especificació més ambiciosa de totes les funcions desenvolupades per a la pràctica i, per tant, la més complicada d'implementar degut a la gran quantitat d'errors que s'avaluen. En conseqüència, és en la que es trobaran les decisions de disseny més interessants de la pràctica. S'han intentat tractar tots els possibles cassos de veïnatge de símbols erroni que poden succeir i no només aquells errors de veïnatge de símbols concrets que presenten dues expressions del fitxer de dades (De fet, si hi ha algun error que no s'ha tractat és perquè no he sigut capaç de detectar-lo en el moment de fer l'especificació). Degut a que la quantitat d'errors tractats era molt gran, s'hagut d'aplicar novament la tècnica de disseny descendent sobre la funció per a reduir de manera considerable el nombre de instruccions i/o condicions i el tamany d'aquestes. Així, s'han definit dues funcions que no estaven planejades originalment com una extensió d'aquest procediment: La funció `Comparar` i la funció `Precomparar`.

La funció `Comparar` s'encarrega de d'avaluar si un dels símbols de l'expressió coincideix amb algun dels símbols de un subconjunt del conjunt de símbols vàlids. Aquest subconjunt haurà estat seleccionat a la funció `VeinatgeValid` a través de les variables que es passen com a paràmetre. Funciona exactament de la mateixa manera que el cos del bucle principal de la funció `SimbolsValids` però amb la diferència de que, en aquest cas, s'ofereix la possibilitat d'actuar sobre un subconjunt del conjunt de símbols i, quan hi hagi coincidència de símbols, el resultat es declararà com a invàlid en lloc de com a vàlid (la variable `vàlid` es posarà a fals en lloc de a cert). Aquesta funció es crida un important nombre de vegades des de la funció `VeinatgeVàlid` i s'utilitza per a trobar gairebé tots els errors deguts a un veïnatge de símbols incorrecte.

La funció `PreComparar` s'encarrega d'avaluar si dos símbols consecutius de l'expressió coincideixen amb algun dels símbols de un subconjunt del conjunt de símbols vàlids. Per a dur-ho a terme, es crida a la funció `Comparar`. El funcionament d'aquesta funció es gairebé idèntic al de la funció `Comparar` però, amb la diferència de que, en el moment en que es detecti una coincidència de símbols, no s'invalidarà el resultat directament, sinó que es cridarà a la funció `Comparar` la qual si que s'encarregarà de declarar el resultat com a invàlid en cas de que hi hagi una segona coincidència entre el següent element de l'expressió i algun dels símbols del mateix subconjunt. Aquesta funció s'utilitza exclusivament per a comprovar si hi ha dos operadors consecutius en l'expressió.

Mitjançant la utilització d'aquestes dues funcions es poden detectar un important nombre d'errors deguts a un veïnatge de símbols incorrecte. La única cosa que s'haurà de tenir en compte de cara a una crida de la funció o a una altra son els valors límits del subconjunt, és a dir, els valor que es passen per paràmetre a cada funció. Les funcions actuaran sempre de la mateixa manera.

La funció `Calcula` és l'encarregada de calcular el resultat de cada expressió en cas de que les funcions de comprovació d'errors no n'hagin detectat cap. Es tracta d'una funció recursiva i, per tant, l'altra funció de gran complexitat en la seva especificació de les que s'han dissenyat en la pràctica. No obstant, tampoc té un disseny particular respecte al que s'esperaria d'una funció recursiva. Aquesta funció rep per paràmetre la taula de caràcters que conté l'expressió i el punter o índex que indica quin element de l'expressió s'ha de llegir en cada moment (en la primera crida a la funció valdrà 0), ambdós passats per referència. La taula de caràcters s'ha de passar per referència degut a que els criteris de funcionament de c determinen que totes les taules es passen sempre per referència, tot i que, en aquest cas, no seria necessari. En canvi, en el cas de l'índex és imprescindible que aquest es passi per referència per a que així el seu valor estarà actualitzat després de retornar de cada crida recursiva. La funció llegeix i avalua, un per un, tots els elements de l'expressió els quals es gestionaran mitjançant tres variables locals definides a l'inici de la funció. Una d'aquestes variables emmagatzemarà l'operador i les altres dues emmagatzemaran els dos números a operar. La funció ha estat dissenyada per a que, en cas de que s'hagin superat totes les comprovacions de correctesa de les dades, la variable a la que li correspon emmagatzemar l'operador mai pugui llegir un parèntesi obert. Per tant, només les dues variables que emmagatzemen un número podran llegir un parèntesi obert. En el cas de que l'element llegit sigui un parèntesi obert en lloc d'un número, la funció es crida recursivament a si mateixa per avaluar el contingut del parèntesi. De fet, es podria afirmar que, en cada crida recursiva, s'avalua el contingut d'un dels parèntesi; de manera que la funció realitzarà tantes crides recursives com parelles de parèntesis presenti l'expressió. Es important recalcar que l'índex s'actualitzarà just després d'haver llegit i emmagatzemat en una de les tres variables un element de l'expressió (fins i tot abans de tractar aquest mateix element). Una vegada s'ha comprovat que l'element llegit correspon a un número es crida a una nova funció.

La funció `ConvertirCaractersANumeros` s'encarrega de convertir un conjunt de caràcters consecutius que corresponen a números en un únic número enter amb tantes xifres com caràcters hi havia en el conjunt. Primer es compten el número de caràcters que representen números que estan disposats de forma consecutiva. Per fer-ho, primer s'efectua un bucle en el qual en cada iteració s'incrementa un comptador de caràcters consecutius. A continuació s'efectua un segon bucle en el qual, en cada iteració, es resta 48 a un dels caràcters numèrics per a obtenir el número enter corresponent al valor numèric codificat en codi ASCII i després es multiplica per la potencia de 10 elevat al número de caràcters (comptador del bucle anterior) menys el comptador d'aquest segon bucle i menys 1. L'índex de l'expressió també es passa per referència a aquesta funció degut a que tots els caràcters que s'hagin llegit ja no s'hauran de llegir al retornar a la funció `Calcula`. En conseqüència, és imprescindible que l'índex estigui sempre actualitzat.

Una vegada es tenen els dos valors numèrics definitius i l'operador, s'ha de calcular el resultat i, per tant, es crida a una nova funció.

La funció `Operacio` s'encarrega de seleccionar, a partir del tipus d'operador, quina es la operació que s'ha d'efectuar entre els dos valors numèrics i obtenir el resultat d'aquesta operació. També, en el cas de que la operació a resoldre sigui una divisió, s'encarrega d'avaluar si el segon operand es 0. En cas afirmatiu, no es podrà resoldre la divisió i, per tant, en la variable resultat s'emmagatzemarà el valor '-1000000'. L'elecció d'aquest valor com a indicador de que no es podrà resoldre la operació es completament arbitrària. Es tracta d'una funció molt senzilla la comprensió de la qual no suposarà cap problema

Finalment, els resultats parcials que es van obtenint en cada crida recursiva de la funció `Calcula` es van emmagatzemant en una de les variables encarregades de emmagatzemar un dels dos valors numèrics de la operació (en funció de quina de les dos és la que ha llegit el parèntesi obert prèviament en aquesta crida recursiva) i es calcula el següent resultat parcial. Aquest pas s'anirà repetint fins a obtenir el resultat final de l'expressió.

Qüestions a valorar

A nivell general, totes les qüestions a valorar que poden ser interessants en aquesta pràctica ja s'han explicat en els apartats anteriors. No obstant, les més importants a destacar son les següents:

S'ha fet un **tractament d'errors molt més exhaustiu** que el que s'exigia en l'enunciat de la pràctica. Per motius de disseny, tots aquests errors addicionals que s'han considerat s'han tractat en la funció `VeinatgeValid`, degut a que s'han interpretat com errors deguts a un veïnatge de símbols incorrecte. Es veritat que alguns d'ells es podrien haver considerat com a errors degut a una falta de coherència en els parèntesis de l'expressió, però en tots aquests cassos aquesta falta de coherència era deguda a un veïnatge dels parèntesis incorrecte o a que aquests es localitzaven en un extrem en el que no podien estar (L'expressió no pot començar amb un parèntesi tancat ni pot acabar amb un parèntesi obert). S'han tractat tots els tipus d'errors per veïnatge de símbols possibles, o, si s'escau, tots els que se m'han acudit. En conseqüència, per a comprovar que tot aquest tractament d'errors funcionés de manera correcta, ha estat necessari el desenvolupament d'un **joc de provés més extens** que el que s'ha subministrat. Evidenment, no es poden tractar tots els cassos possibles però, al menys, s'ha dissenyat una expressió per a tractar cada tipus d'error que s'ha implementat. Totes les expressions que s'han dissenyat es troben recollides en l'apartat dels jocs de proves.

També s'ha de destacar que l'ús de la **tècnica del disseny descendent** s'ha tingut molt en compte a l'hora de dissenyar el programa amb l'objectiu d'obtenir un codi més estructurat, més òptim o ambdues coses.

Problemes sorgits durant el desenvolupament de la pràctica

Un cop acabada la implementació de la pràctica aquesta no funcionava correctament degut a tres motius principals. Es per aquests tres motius que no vaig aconseguir entregar un codi completament funcional en la primera convocatòria.

El primer error que es va cometre, va ser el de no considerar la possibilitat de que en una expressió poguessin aparèixer números de dos o més xifres. Quan un número de dos o més xifres s'ha de tractar com un caràcter, cada una de les xifres es tracta com un caràcter diferent. Per tant, un nombre de vàries xifres estaria format per diversos elements consecutius de la taula de caràcters que emmagatzema l'expressió complerta. En aquesta situació, la funció `Calcula` estava dissenyada per a que, un cop s'hagués llegit un caràcter i s'hagués determinat que corresponia a un valor numèric, el següent caràcter que es llegís correspongués o a l'operador o al parèntesis tancat que indicava el final de la operació. En conseqüència, per a totes aquelles expressions que continguessin números de dos o més xifres es calculava un resultat completament erroni. La solució va ser dissenyar la funció `ConvertirCaractersANumeros`. Cada vegada que es detecta que un caràcter de l'expressió equival a un valor numèric, es crida a aquesta funció, la qual obté aquest caràcter i tots els caràcters consecutius que tinguin un valor numèric i els transforma en un número enter el qual retornarà fins a la funció `Calcula` (amb l'actualització de l'índex inclosa). El funcionament d'aquesta funció està explicat més en detall en l'apartat de disseny de les especificacions i en la descripció de la funció en el programa.

En segon lloc, la funció `VeinatgeValid`, degut a la seva complexitat, presentava certs erros d'execució. Un d'ells i el principal era que no es comprovava si l'expressió ja s'havia declarat com a no vàlida abans de comprovar si hi havia més errors. Això implicava que, encara que s'hagués detectat que la funció no era vàlida perquè s'havia detectat un error, el següent error s'avaluava igualment; i, si aquest següent error no es detectava en l'expressió, la funció es tornaria a declarar com a vàlida ignorant la existència de l'error anterior. Aquest error es va solucionar fàcilment afegint la següent condició abans de fer qualsevol de les comprovacions d'errors (amb excepció de la primera comprovació, en la que no serà necessari):

```
if (vàlida) {
```

Durant l'avaluació d'aquesta funció també es van detectar altres errors que si ve no es mostraven en l'execució del programa ja que era molt difícil que es donés aquell cas concret, eren errors al cap i a la fi. Un d'aquests errors, per exemple, va ser que quan s'estava avaluant un element de l'expressió i s'havia de comparar amb el següent element de l'expressió, no es comprovava que aquest següent element no fos el caràcter de fi de cadena. Aquest error es va solucionar fàcilment mitjançant l'addició de la següent condició:

```
if (Expressio[i]==')') {
```

En general la detecció dels errors que s'han produït en aquesta funció ha estat relativament senzilla. Una lectura i comprensió més profunda del codi ha estat suficient per a resoldre'ls tots.

En tercer lloc, la funció `Calcula` també presentava alguns errors. Els errors que ha tingut aquesta funció han estat molt més difícils de detectar que els errors anteriors. Realment, el fet de garantir que aquests errors estaven en aquesta funció ja de per si ha estat bastant complicat, tot i que semblava probable degut al tipus d'errors. Un cop es va tenir la certesa de que l'error es trobava en aquesta funció va ser necessària la utilització del Depurador (*Debugger*) que permet executar el programa pas a pas i permet comprovar el contingut de les variables després de l'execució de cada instrucció. Amb l'ajut d'aquesta eina, es van aconseguir trobar dos errors de gran importància. El primer d'ells, el qual afectava a gairebé tots els resultats de les expressions (excepte aquelles en les que l'expressió era molt simple i, o no s'utilitzaven parèntesis o s'utilitzaven pocs i al final) afectava al índex utilitzat per recórrer l'expressió. Com que aquest índex es passava per valor en cada crida recursiva, després de calcular el primer resultat parcial i del retorn, amb aquest resultat parcial, a la funció superior; el índex no estava actualitzat, ja que es recuperava el valor anterior a haver executat aquesta última crida recursiva. Un cop detectat l'error, la solució ha estat molt simple: Passar la variable índex per referència en lloc de per valor.

```
*i
```

El segon error afectava únicament a l'expressió 7, que es la única expressió que presenta dos parèntesis tancats consecutius en una localització diferent al final del programa. Aquest error no es produïa en la resta d'instruccions degut a que després de que s'executés una crida recursiva i s'emmagatzemés el resultat parcial en una de les variables, sempre s'incrementava una posició l'índex abans de llegir un nou element de l'expressió. Això és suficient per evitar llegir un parèntesi tancat, però no més d'un. Aquest error s'ha solucionat mitjançant l'addició d'un bucle al final de la funció (just abans de retornar el resultat), el qual incrementa l'índex de l'expressió mentre es llegeixi un parèntesi tancat.

```
while (Expressio[*i]==')') {

        (*i)++;

}
```

Per descomptat, les instruccions d'incrementar una unitat l'index just després d'una crida recursiva s'han eliminat. Després d'això la funció `Calcula` funcionava correctament. Per a comprovar que aquest segon error s'hagués solucionat correctament, s'ha afegit alguna instrucció addicional al joc de proves que presentava diversos parèntesis tancats consecutius, tal com s'explica en l'apartat dels jocs de proves.

L'últim error que es va detectar va ser en la funció `ConvertirCaractersANumeros` i afectava l'expressió 9, per a la que donava sempre el mateix resultat incorrecte, el qual

era més gran que el resultat esperat. Aquest error era degut a la utilització de la funció `pow` de la biblioteca estàndard de c, funció que permet calcular la potencia d'un nombre qualsevol elevat a un altre nombre qualsevol. Degut a que aquesta funció no ha estat dissenyada per mí, he hagut de fer una recerca d'informació a internet sobre aquesta funció per a veure si trobava l'error. Finalment he trobat que la funció `pow` retorna un valor real amb moltes xifres significatives que es una aproximació molt propera al resultat verdader del càlcul de la potencia. En conseqüència, en fer el càsting d'aquest valor real per emmagatzemar-lo en una variable entera, es pot arribar a perdre fins a una unitat sencera del valor final després d'efectuar la potència. Per a solucionar-ho he buscat també a internet una funció de la biblioteca estàndard de c que em permeti arrodonir un valor. La funció trobada ha estat la funció `round`.

```
int round (float número_per_arrodonir);
```

La funció s'ha utilitzat de la següent manera.

```
round(pow(10, max-j-1));
```

Després de solucionar aquest últim error, ja no s'ha detectat cap més problema en l'execució del programa ni amb el joc de proves original ni amb els jocs de proves extensos que s'han provat.

Conclusions

El desenvolupament d'aquesta pràctica permet adquirir uns coneixements en el disseny de funcions recursives molt sòlids. El disseny de la funció recursiva `Calcula` es tant complexe que, fins i tot sense aconseguir una implementació completament funcional, s'obtenen uns coneixements prou amplis en recursivitat com per passar la prova parcial (o al menys la part de la prova parcial corresponent a aquest tema) sense cap tipus de problema. Això es indicatiu de que, al menys en aquest aspecte, la pràctica era molt coherent, ja que estava molt ben relacionada amb els conceptes explicats en la classe de teoria. No obstant, si se l'hi ha de trobar algun defecte (el qual ja s'ha mencionat en les sessions de teoria), aquest és que, a aquest nivell, la recursivitat únicament sembla una manera més enrevessada de desenvolupar les mateixes funcionalitats que porta a terme un bucle. En el meu cas, com ja he cursat l'assignatura d'estructures de dades soc conscient de la seva utilitat per al desenvolupament de funcions d'exploració d'arbres, però per a la major part d'estudiants encara no s'aprecia la utilitat d'aquesta tècnica de programació. Tot i així, aquest és un defecte sense major importància.

Per altra banda, el desenvolupament de unes funcions de comprovació d'errors que avaluïn tots els errors possibles suposa un nivell d'exigència considerablement elevat. Evidenment el desenvolupament d'aquestes funcions m'ha permès consolidar una mica més els meus coneixement de llenguatge c, però no ha implicat l'obtenció d'uns coneixements tan disruptius com ha estat el desenvolupament de la funció `Calcula` i, en canvi, si que ha suposat la mateixa càrrega de treball. També es veritat, que en l'enunciat s'especifica que no es necessari desenvolupar una funció de comprovació d'errors gaire complexa (que amb que detecti els errors en el fitxer de proves és suficient) i jo he comprovat molts més errors. Per tant, d'aquí la seva major dificultat i, desenvolupar una funció tan complexa, ha estat la meva decisió. De cara a un futur, podria ser interessant que això es recalqués com una activitat optativa de la primera pràctica de cara a, per exemple, obtenir la màxima nota.

També he consolidat els meus coneixements sobre el disseny descendent. Sobre tot respecte a la dualitat de possibilitats que ofereix aquesta tècnica, ja que l'he pogut utilitzar per desenvolupar un codi molt més estructurat, com en el cas del main, la funció `TractarDades` i la funció `Calcula` o per obtenir un codi molt més òptim, com en el cas de la funció `VeinatgeValid`. Queda clar que, quan s'estructura un programa, aquest sol reduir el seu nivell d'optimització mentre que quan s'optimitza el mateix codi, aquest normalment perd en estructuració. D'aquí la complexitat de desenvolupar una bona tècnica de disseny descendent.

Tot i que no estava entre els objectius de la pràctica, també s'ha de destacar que he aprés a utilitzar la eina coneguda com Depurador (*Debugger*), la qual mai havia necessitat utilitzar fins ara i que, en aquest cas, ha estat molt útil per a trobar errors que no era capaç de trobar de cap altra manera.

I, per acabar, també podria dir que és la primera documentació d'una pràctica que he pogut desenvolupar en detall i amb tota la informació que volia plasmar sobre la pràctica, cosa que també ha estat bastant útil, ja que ara també sé com fer una bona documentació.

A nivell general, el balanç de la realització de la pràctica en quan a aprenentatge ha estat molt positiu.

Bibliografia

S'han consultat les següents pàgines web durant el desenvolupament de la pràctica:

- http://www.carlospes.com/curso_de_lenguaje_c/01_08_01_03_la_funcion_pow.php
- https://www.google.com/search?q=funcion+round+en+c&rlz=1C1CHBF_esES807ES808&oq=funcion+round+en+c&aqs=chrome..69i57j69i60l3.4303j0j7&sourceid=chrome&ie=UTF-8
- https://www.youtube.com/watch?v=MFMzDZEdL4k
- https://www.youtube.com/watch?v=Jab1qj_QR8s