Projecte ESIN:

DATA LÍMIT 15 gener 23h

programa computació simbòlica ESINMATH

mode interactiu: usuari entre comanda i respon el sistema, ESINMATH processa la info i la dona la resposta requerida, i espera una nova comanda de l’usuari

comandes:

1. expressions que ESINMATH avalua i el resultat de les quals imprimeix.
2. assignacions de la forma x := E, on x és una variable qualsevol i E una expressió.
3. unassign x, ens permet desassignar el valor assignat a una variable x.
4. la comanda byebye finalitza la sessió amb ESINMATH.

Constants numèriques:

* Nombre enter →seq dígits del 0..9 seguit + o -
* Nombre racional→ mantissa seguit opcional Exponent

(si Exponent entre mantissa i exponent escriure E) exponent es constant entera

Mantissa part entera seguida opcionalment un punt ‘ . ’ i part fraccionària(const enters sense signe)

Resultat realitzar quocient de dues constants enteres

* Nombre real en coma flotant

Variables identifiquen mitjançant un nom, una o més lletres maj o min i caràcter ‘\_’

Nom variable no pot coincidir nom d’una funció predefinida(sqrt,exp,log,evalf) ni constant e (base log naturals), ni la variable predefinida %

Operadors reconeguts:

* suma +
* resta -
* canvi signe -
* multi \*
* divisió /
* exponent ^

assignació := // operador unassign- no expressions però utilitzar en comandes

+utilitzar parèntesis per agrupar subexpressions i canviar prioritat

* Qualsevol constant entera, racional o real en coma flotant és una expressió vàlida. També ho és la constant predefinida e.
* Qualsevol variable i la variable especial % és una expressió vàlida.
* Si E és una expressió vàlida llavors (E) i −E i +E són expressions vàlides.
* Si E1 i E2 són expressions vàlides llavors també ho són E1 + E2, E1 − E2, E1 ∗ E2, E1/E2 i E1ˆE2.
* Si f és una de les funcions predefinides sqrt, exp, log o evalf i E és una expressió vàlida llavors f(E) és una expressió vàlida

Precedència:

- (canvi de signe), + (signe positiu)

∗, /

+ (suma), - (resta)

Dos operadors mateixa prioritat s’avalua ESQUERRA-DRETA

Expressions incorrectes:

* qualsevol expressió E, E/0 impossible
* no derivar expressió E respecte qualsevol cosa q no sigui una variable
* no evaluar logaritme/arrel nombre negatiu
* variable x sense cap valor no podem tenir expressions de x=x+1 -recursivitat infinita

1. Un cop llegida l’expressió, ESINMATH descomposa l’expressió en una seqüència

de tokens; cada **token** és:

* un **valor literal** (una constant entera, racional o en coma flotant)
* un identificador (de variable o símbol de funció), un operador (+, \*, . . . )
* un parèntesi d’obertura o tancament o una coma.

1. A continuació construeix un **arbre d’expressió**, comprovant a la vegada la correctesa sintàctica de l’expressió llegida.
2. L’algorisme que s’usa per construir l’arbre d’expressió és molt similar al que s’usa per convertir una expressió en notació infixa (l’habitual) a l’anomenada **notació postfixa o polaca.** La generació de l’arbre d’expressió, incloent la detecció d’errors de sintaxi, constitueix la fase que s’anomena **anàlisi sintàctic.**
3. L’últim pas consisteix en avaluar l’expressió. L’avaluació d’una expressió és una altra expressió i això és el que ESINMATH acabarà imprimint en el canal estàndar de sortida—també emmagatzemarà l’expressió resultant a la variable especial %. Aquesta fase en la que es **simplifica** i **s’avalua** una expressió s’anomena **fase semàntica**

Valor: valors que et donen les lletres de l’enunciat

Avaluació: valor numèric donat a una variable ??

→va variant si els valors de les variable canvia

Normes determinar valor o avaluació:

1. El **valor i l’avaluació d’una expressió constant numèrica** és la **pròpia constant** (en el cas de la constant predefinida e el seu valor i la seva avaluació és el seu nom).
2. El **valor d’una variable és l’avaluació de l’última expressió que se li hagi assignat;** en el seu defecte, el seu nom. L’avaluació d’una variable és l’avaluació del seu valor en aquest instant.
3. En la comanda x := E s’imprimeix el resultat de l’avaluació d’E; addicionalment s’assigna l’avaluació de E com a valor de la variable x.

Regles:

// c,c’c1,c2 constants numériques

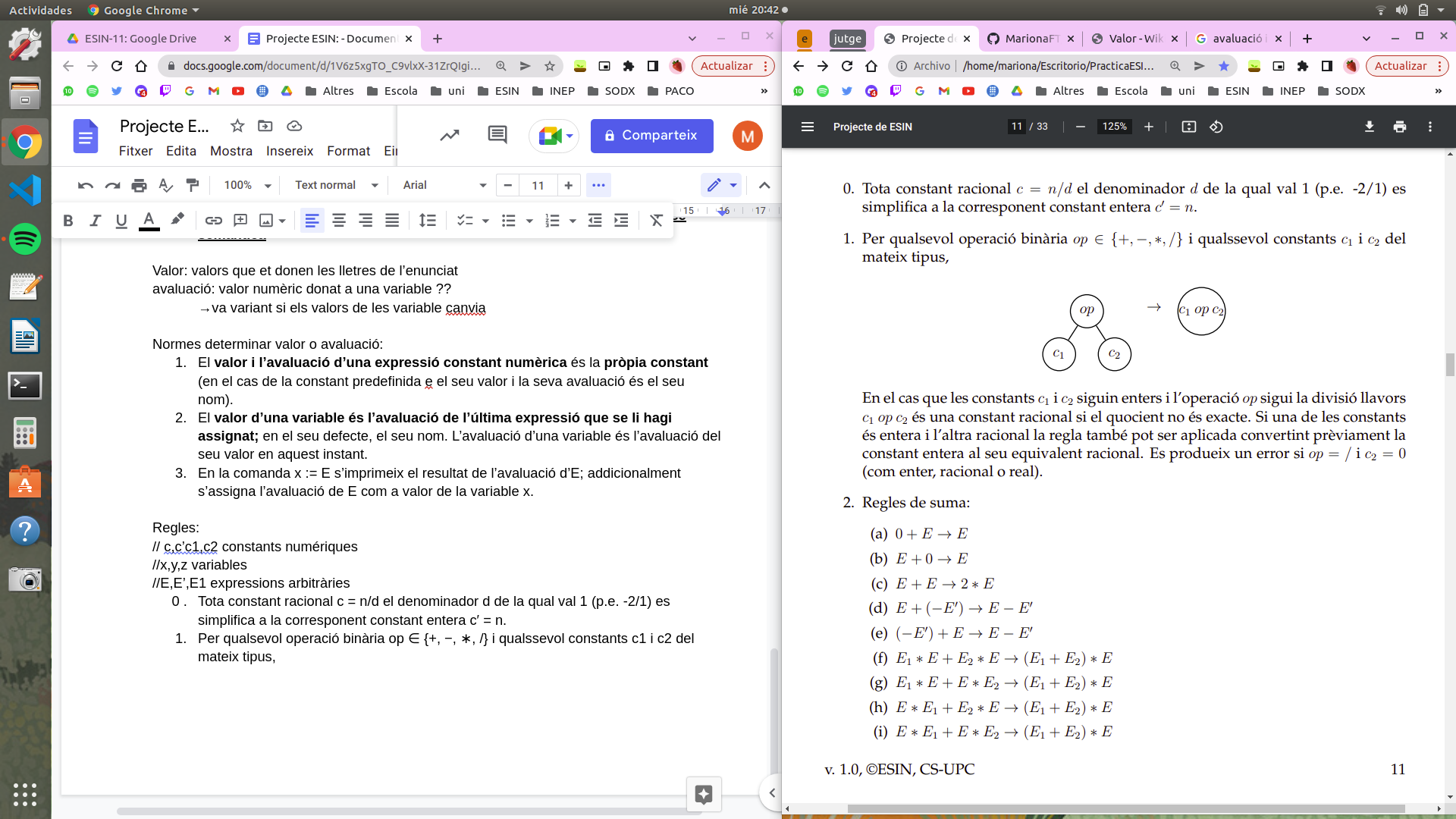
//x,y,z variables

//E,E’,E1 expressions arbitràries

0 . Tota constant racional c = n/d el denominador d de la qual val 1 (p.e. -2/1) es

simplifica a la corresponent constant entera c′ = n.

1. Per qualsevol operació binària op ∈ {+, −, ∗, /} i qualssevol constants c1 i c2 del mateix tipus

SI constants c1,c2 son enteres i op divisó i resultat serà una constant racional si quocient no és exacte op divisió amb resultat racional si una de les constants és entera i l’altre racional Error si op =/ i c2 =0 (sigui enter, racional o real)-divisió de 0

| 1. Regles suma:   0 + E → E  E + 0 → E  E + E → 2 ∗ E  E + (−E′) → E − E′  (−E′) + E → E − E′  E1 ∗ E + E2 ∗ E → (E1 + E2) ∗ E  E1 ∗ E + E ∗ E2 → (E1 + E2) ∗ E  E ∗ E1 + E2 ∗ E → (E1 + E2) ∗ E  E ∗ E1 + E ∗ E2 → (E1 + E2) ∗ E  E1/E + E2/E → (E1 + E2)/E | 1. Regles resta;   0 − E → −E (canvi de signe)  E − 0 → E  E − E → 0  E − (−E′) → E + E′  E1 ∗ E − E2 ∗ E → (E1 − E2) ∗ E  E1 ∗ E − E ∗ E2 → (E1 − E2) ∗ E  E ∗ E1 − E2 ∗ E → (E1 − E2) ∗ E  E ∗ E1 − E ∗ E2 → (E1 − E2) ∗ E  E1/E − E2/E → (E1 − E2)/E |
| --- | --- |
| 1. Regles de multiplicació:   1 ∗ E → E  E ∗ 1 → E  0 ∗ E → 0  E ∗ 0 → 0  E ∗ E → Eˆ2  E ∗ (−E ′ ) → −(E ∗ E ′ )  (−E) ∗ E ′ → −(E ∗ E ′ )  E ∗ (1/E′ ) → E/E′  (1/E′ ) ∗ E → E/E′  E1ˆE ∗ E2ˆE → (E1 ∗ E2)ˆE  EˆE1 ∗ EˆE2 → Eˆ(E1 + E2)  exp(E) ∗ exp(E ′ ) → exp(E + E ′ ) | 1. Regles de divisió:   E/0 és un error.  0/E → 0  E/1 → E  E/E → 1  E/(−E ′ ) → −(E/E′ )  E/(1/E′ ) → E ∗ E ′  E1ˆE/E2ˆE → (E1/E2)ˆE  EˆE1/EˆE2 → Eˆ(E1 − E2)  exp(E)/exp(E ′ ) → exp(E − E ′ ) |
| 1. Regles canvi signe i signe positiu:   −(−E) → E  +E → E  Per qualsevol constant c (entera, racional o real) | 1. Regles exponenciació:   Si c2 és una constant de tipus enter, o si c1 i c2 son constants reals llavors  Error si c1 constant negativa(enter,racional,real) i c2 no un constant entera  Eˆ0 → 1  Eˆ1 → E  Eˆ − E ′ → 1/(EˆE ′ )  (E1ˆE2)ˆE3 → E1ˆ(E2 ∗ E3)  (exp(E))ˆE ′ → exp(E ∗ E ′ ) |
| 1. Regles radicació (funció **sqrt**)   sqrt(c) → error, per qualsevol constant entera, racional o real c < 0 sqrt(0) → 0  sqrt(E) → Eˆ 1 2 (aquí, una meitat es representa com constant racional, no com una expressió amb tres tokens)  Si la funció sqrt està aplicada sobre una constant en coma flotant es calcula d’immediat el seu resultat usant la funció corresponent de la biblioteca matemàtica de C++, i en concret es pendrà el valor positiu (p.e. tant 3.0 com -3.0 són valors correctes per sqrt(9.0), però ens quedarem amb el primer). | 1. Regles de les funcions exp i logaritme (natural):   exp(0) → 1  exp(1) → e v. 1.0  exp(log(E)) → E  log(c) → error (const enter, rac o real) c ≤ 0  log(1) → 0  log(e) → 1  log(exp(E)) → E  log(E ∗ E ′ )) → log(E) + log(E ′ )  log(E/E′ )) → log(E) − log(E ′ )  log(EˆE ′ )) → E ′ ∗ log(E) (k)  Si la funció exp o log estan aplicades sobre una constant en coma flotant es calcula d’immediat el seu resultat usant les funcions corresponents de la biblioteca matemàtica de C++. |
| 1. L’operació **evalf** retorna quelcom diferent depenent al que s’estigui aplicant:    1. Si s’aplica sobre una constant en coma flotant torna la pròpia constant.    2. Si s’aplica sobre un enter realitza la conversió, i si s’aplica sobre un racional es calcula el quocient real entre el numerador i el denominador.    3. Si s’aplica sobre la constant e retorna el valor 2.718281828.    4. Si s’aplica sobre una **variable x** que no té valor assignat **retorna la pròpia variable**.    5. Per la resta, la seva aplicació és recursiva:   evalf(E op E′ ) → evalf(E) op evalf(E ′ )  si op és una operació binària i  evalf(f(E)) → f(evalf(E)) per una funció f qualsevol except a pròpia evalf. | |

Procés avaluació:

només intervenen els valors assignats a les variables

obtenir una expressió final on:

* variables apareguin en ella NO tinguin valor assignat
* expressió estigui completament simplificada

Si compleix reduït a una **forma normal**

Passos a seguir per reduir una expressió:

1. substituir totes les variables d’E per les expressions assignades q tinguin en el moment. S’ha d’anar iterant fins q totes les variables obtinguda E’ siguin variables que no tinguin valor assignat.

error no permetre assignacions CIRCULARS on valor assignat variable x es una expressió amb la variable x

i E = a + c on a = c i c = a + 1 ⇒ a + c, c + c, (a + 1) + (a + 1), (c + 1) + (c + 1),

((a + 1) + 1) + ((a + 1) + 1)....

1. regles de simplificació E0 = E’ obtenint a cada pas una expressió Ei+1 fins q no podem aplicar més normes de simplificació. Expressió final obtinguda En resultat de l’avaluació inicial de E

Seguir recorregut LEFT-RIGHT fins trobar una subexpressió Ei on poder simplificar

Si hi han dos simplificacions a la subexpressió, aplicar l’especificada en primer lloc

| E = a + b ∗ c  variables tenen assignat valors: a = 0, b = c i c = sqrt(d) |
| --- |
| a) Aplicació de les substitucions. Per exemple (aquí l’ordre no és rellevant):  a + b ∗ c substitueix c por sqrt(d)  a + b ∗ sqrt(d) substitueix a por 0  0 + b ∗ sqrt(d) substitueix b por c  0 + c ∗ sqrt(d) substitueix c por sqrt(d)  0 + sqrt(d) ∗ sqrt(d) no és possible fer més substitucions |
| E ′ = 0 + sqrt(d) ∗ sqrt(d) |
| b) Passos de simplificació. L’ordre és necessàriament el següent (al costat de cada pas s’indica la regla aplicada per passar a la següent expressió):  1. E0 = 0 + sqrt(d) ∗ sqrt(d) regla 8d: sqrt(E) → Eˆ1/2  2. E1 = 0 + dˆ1/2 ∗ sqrt(d) regla 8d: sqrt(E) → Eˆ1/2  3. E2 = 0 + dˆ1/2∗ dˆ1/2 regla 5e: E ∗ E → Eˆ2  4. E3 = 0 + dˆ1/2^2 regla 7e: (E1ˆE2)ˆE3 → E1ˆ(E2 ∗ E3)  5. E4 = 0 + dˆ( 1/2∗ 2) regla 1: racional ∗ enter → racional  6. E5 = 0 + dˆ1/1 regla 0: racional → enter  7. E6 = 0 + dˆ1 regla 7c: Eˆ1 → E  8. E7 = 0 + d regla 2a: 0 + E → E  9. E8 = d  **forma normal** |
| E8=d expressió resultant de E=a+b\*c |

Disseny modular:

Implementar classe racional,token,expressió,variables, math\_io i math\_sessió

Totes classes implementar mètodes construcció còpia,assignació i destrucció

**CLASSE TOKEN:**

objecte classe token conté:

* un operand
  + Enters -(emmagatzema valor corresponent)-enter
  + Racionals -(emmagatzema valor corresponent)-racional
  + Reals (en coma flotant) -(emmagatzema valor corresponent)-real
  + constant predeterminada e
  + variables -(emmagatzema string del nom)
  + variable predefinida %
* un operador
  + suma +
  + resta -
  + multi \*
  + div /
  + expo ^
  + canvi signe -
  + signe positiu +
* funció predefinida
  + arrel quadrada-sqrt
  + exponencial- exp
  + logaritme-log
  + avaluació coma flotant- evalf
* símbol de puntuació () o ,
* comandes:
  + assignació :=
  + dessignació -unassign
  + final de sessió - byebye

INDICAR COST

**CLASSE RACIONAL:**

emmagatzema un nombre racional

numerador u denominador sempre primers entre si

El signe del racional sempre en el numerador(denominador sempre positiu)

Si numerador o el denominador simplificat ha de ser 1

Ex 10/-5 simplificat -2/1

INDICAR COST

**CLASSE EXPRESSIÓ:**

Objectes classe expressió permeten representar i manipular expressions inicialment creades a partir d’una seqüència de tokens.

Decisió: nombre de tokens a partir del qual es construeix l’expressió no està acotat

Es poden usar classes **list i stack**

Desenvolupar gradualment l’implementació:

impl regles suma,producte…

processar adequadament les expressions que involucrin exclusivament constants enteres i racionals, variables i els operadors ’+’, ’-’ (canvi de signe), ’-’ (resta), ’\*’ i ’/’. ⇒ incorporeu el codi necessari per processar la resta d’operadors i funcions, les constants reals

NO INDICAR COST

**CLASSE VARIABLE:**

Diccionari de parells <clau,valor> clau=string representa variable

valor=expressió q té assignada

Decisió:un objecte conté un nombre no acotat de variables, on totes son diferents i sempre un valor associat

Es pot usar **list** per retornar dades del mètode **dump** NO més objectiu

MÈTODES EFICIENTS EN CAS PITJOR O MIG

constructor copia,assignació,destructor cost=O(N) N=num variables

resta (except dump) cost=O(log(N))o inferior

INDICAR COST

**CLASSE MATH\_IO:**

Info necessaria desenvolupar programa ESINMATH, guarda tot moment estat sessió i executa comandes consultes/modificar l’estat

Usar **list** NOMÉS entrada i sortida de dades NO més objectiu

NO INDICAR COST-FETA

**CLASSE MATH\_SESSIÓ:**

conté mètodes scan realitza anàlisi i descomposició d’un string de tokens u un altre tostring que realitza transformació de tokens a strings

PRE/POST

* *static node\* copia\_nodes(node \*m);*
* *static void esborra\_nodes(node\* m);*
* *static void imp\_arbre(node \*x);*
* *static node\* node\_creat(node\*x,token t,node \*y);*
* *expressio crea\_expressio(expressio e1,token t, expressio e2);*
* *static bool prioritat(token t1,token t2);*
* *static bool arbre\_igual(node\*x, node \*y);*
* *static bool llista\_sintactica(const list<token> &l);*
* *static void node\_sintactic(node \*x);*
* *static bool expressio\_en\_variable(const string & v, node \*x);*
* *static node\* substituir(node\* &x,const string & v, expressio e);*
* *static list<string> llista\_variable(node \*x, list<string> &l);*
* *static void llista\_arbre(node \*x,list<token> &l);*
* *static node\* simplificacio\_numeral\_binaria(node \*x);*
* *static node\* simplificacio\_numeral\_unaria(node \*x);*
* *static node\* suma\_simplificada(node \*x);*
* *static node\* resta\_simplificada(node \*x);*
* *static node\* canvi\_signe\_simplificat(node \*x);*
* *static node\* multiplicacio\_simplificada(node \*x);*
* *static node\* divisio\_simplificada(node \*x);*
* *static node\* exponenciacio\_simplificada(node \*x);*
* *static node\* sqrt\_simplificada(node \*x);*
* *static node\* logexp\_simplificada(node \*x);*
* *static node\* evalf\_simplificada(node \*x);*
* *static node\* arbre\_simplificat(node\* &x, bool &simp);*
* *static node\* simplificacio\_total(node \*x);*
* static node\* copia\_nodes(node \*m);
* static void esborra\_nodes(node\* m);
* static void imp\_arbre(node \*x);
* static node\* node\_creat(node\*x,token t,node \*y);
* expressio crea\_expressio(expressio e1,token t, expressio e2);
* static bool prioritat(token t1,token t2);
* static bool arbre\_igual(node\*x, node \*y);
* static bool llista\_sintactica(const list<token> &l);
* static void node\_sintactic(node \*x);
* static bool expressio\_en\_variable(const string & v, node \*x);
* static node\* substituir(node\* &x,const string & v, expressio e);
* static list<string> llista\_variable(node \*x, list<string> &l);
* static void llista\_arbre(node \*x,list<token> &l);
* static node\* simplificacio\_numeral\_binaria(node \*x);
* static node\* simplificacio\_numeral\_unaria(node \*x);
* static node\* suma\_simplificada(node \*x);
* static node\* resta\_simplificada(node \*x);
* static node\* canvi\_signe\_simplificat(node \*x);
* static node\* multiplicacio\_simplificada(node \*x);
* static node\* divisio\_simplificada(node \*x);
* static node\* exponenciacio\_simplificada(node \*x);
* static node\* sqrt\_simplificada(node \*x);
* static node\* logexp\_simplificada(node \*x);
* static node\* evalf\_simplificada(node \*x);
* static node\* arbre\_simplificat(node\* &x, bool &simp);
* static node\* simplificacio\_total(node \*x);