Test de la classe Programa X55206_ca

Això és el test de la classe `Programa`, que correspon a la pràctica **Simulació d'un SO**. Aquesta classe implementa un intèrpret de programa escrit en un **llenguatge assemblador** senzill. Aquesta classe **contindrà** un programa en aquest llenguatge, i a més, l'executarà instrucció a instrucció. Com veureu, us donem els fitxers `programa.hpp` i `programa.cpp` que contenen, respectivament, l'especificació de la classe i la implementació d'algunes dels mètodes. En aquesta part haureu d'implementar els mètodes que manquen.
Per a entendre la part que us falta implementar, caldrà que entengueu el joc d'instruccions del programa i també l'estructura interna d'un programa. Molts d'aquests conceptes ja els haureu vist en assignatures d'estructura de computadors.
En primer lloc, un programa tindrà un **comptador de programa** (`PC`) que indicarà la línia de codi que cal executar del programa. En un programa, cada línia té implícit un número de línia, que comença amb el 10 i avança de 10 en 10. Per exemple:

```
10: DISPBUIT 50
20: LLEGEIX x
30: EMPILA x
40: GOTO 10
50: PILABUIDA 90
60: DESEMPILA x
70: ESCRIU x
80: GOTO 50
90: NOFARES
```

La primera línia sempre és la 10. Diem que els números de línia són **implícits** perquè quan instanciem un programa, no els posem de manera explícita. Per exemple, el programa anterior estaria escrit de la següent manera:

```
DISPBUIT 50
LLEGEIX x
EMPILA x
GOTO 10
PILABUIDA 90
DESEMPILA x
ESCRIU x
GOTO 50
NOFARES
```

però com que es tracta d'un llenguagte que no té la declaració ni de funcions ni de subrutines, és bastant fàcil deduir el número de cada línia.
Per altra banda, un programa també té un **identificador** (`_id`), una **prioritat** (`_prioritat`) i un identificador del processador a què pertany (`_proc`) (això no cal que ho tingueu en compte fins que feu la versió final de la pràctica).
Internament, té **dos registres**: un registre d'**estatus** (`STATUS`) que té l'estatus en què ha acabat la darrera instrucció que s'ha executat, i també un estatus de la darrera **comparació** (`COMPARACIO`), que indica l'estatus de la darrera instrucció de comparació.
El registre `STATUS` pot tenir els següents valors:

- -1 Pila plena.
- -2 Divisió per zero.
- -3 Desempilar pila buida.
- -5 Adreça fora de rang
- -8 Error lectura dispositiu.
- -9 Error escriptura dispositiu.
- -10 Programa acabat.

El registre `COMPARACIO` pot tenir els següents valors:

- 0: neutre.
- 1: més gran.
- 2: més petit.
- 3: igual.

A més, també té una pila interna (`pila`), que té una capacitat màxima definida a la variable `CAPACITAT = 100`, una **memòria** (un objecte de tipus `memoriaBST`) i un **dispositiu** (un objecte de tipus `Dispositiu`).
El joc d'instruccions possibles és aquest:

- Operacions aritmètiques.

```
SUMA  op1 op2
RESTA  op1 op2
MULT op1 op2
DIV op1 op2

op1 i op2 poden ser:
(1) enters
(2) noms de variables

el resultat de l'operació s'empila a la pila del programa.

errors:
-1 pila plena
-2 divisió per zero
```

- Operacions sobre pila.

```
EMPILA  op
op pot ser un enter o un nom de variable.
empila op a la pila del programa.

errors:
```

```
-1 pila plena

DESEMPILA op
op és un nom de variable.
desempila el valor de la pila del programa
i el posa a op.

errors:
-3 pila buida
```

- Operacions sobre dispositiu.

```
LLEGEIX op
op és un nom de variable.
posa el valor contingut del dispositiu a la variable de memòria op.

errors:
-8 dispositiu buit

ESCRIU op
op pot ser un enter o una variable. posa el valor d'op al dispositiu.

errors:
-9 dispositiu ple
```

- Operació d'assignació.

```
MOU op1 op2
op1 és una variable de memòria.
op2 pot ser una variable de memòria o un enter.

assigna a op1 el valor d'op2
```

- Operacions de salt.

```
GOTO op
op és un enter múltiple de 10 que indica una línia
del programa.

fa un salt incondicional a op.

errors:
-5 op és una línia que no existeix

MESGRAN op
MESPETIT op
IGUAL op
PILABUIDA op
DISPBUIT op
```

```
op és un enter múltiple de 10 que indica una línia
del programa. fa un salt condicionat a op:
si el flag de comparació és més petit, més gran, igual,
o la pila del programa és buida, o el dispositiu és buit.

errors:
-5 op és una línia que no existeix
```

- Operació de comparació.

```
CMP op1 op2

op1 i op2 poden ser enters o variables de memòria.
compara els valors continguts a op1 i op2 i posa el flag
de comparació a:

1: op1 > op2
2: op1 < op2
3: op1 = op2
```

- Operacions d'increment.

```
INCREMENTA  op
DECREMENTA  op

op és una variable de memòria.
incrementa o decrementa en una  unitat el valor
de la variable de memòria op.
```

- `NOFARES`. Aquesta operació no fa literalment res (llevat d'incrementar `PC`).

Tingueu en compte que cada vegada que executem una instrucció es pot modificar el comptador de programa (`PC`), l'estatus (`STATUS`) i l'indicador de comparació (`COMPARACIO`)
Fixeu-vos que l'enunciat d'aquest exercici ja ofereix uns fitxers que haureu d'utilitzar per a compilar: `Makefile, program.cpp`, apart de les versions `*.old` dels altres fitxers que heu d'acabar d'implementar. A més, **per a fer aquesta part haureu de penjar també les implementacions de les classes** `Dispositiu` i `memoriaBST`, a més del fitxer `BST.cpp` que heu fet en els passos anteriors.
Quan pugeu la vostra solució al jutge, només cal que pugeu un tar construït així:

```
tar cf solution.tar memoriaBST.cpp memoriaBST.hpp BST.cpp
dispositiu.cpp dispositiu.hpp programa.cpp programa.hpp
```

Informació del problema

Autoria: PRO1

Generació: 2026-01-25T21:16:31.589Z