Algoritmos y Estructuras de Datos.
TPL1. Trabajo Práctico de Laboratorio 1. [2018-09-06]

PASSWD PARA EL ZIP: X5QV 296R F6V8

Ejercicios

ATENCION: Deben necesariamente usar la opción `-std=gnu++11` al compilador, **si no no va a compilar**.

[Ej. 1] **[sign-split]** Implemente una función
`void sign_split(list<int> &L,vector< list<int> > &VL);` que dada una lista `L`, retornar el un vector de listas `VL` las sublistas contiguas del mismo signo (el caso 0 es considerado junto con los positivos, es decir separa negativos de no-negativos).
Ejemplos:

```
L:[4,-3,-5,-4,-5,-1,4,-1,-5,-5,1,-5,3,3,0,2,2],
=> VL:[[4],[-3,-5,-4,-5,-1],[4],[-1,-5,-5],[1],[-5],[3,3,0,2,2]],

L:[0,4,-2,4,1,-1,-4,-4,-3,-1,-4,4,1],
=> VL:[[0,4],[-2],[4,1],[-1,-4,-4,-3,-1,-4],[4,1]],

L:[2,-1,3,-3,3,-3,0,-1,-2,0],
=> VL:[[2],[-1],[3],[-3],[3],[-3],[0],[-1,-2],[0]],
```

Ayuda:

- Chequear el signo `s` del primer elemento y extraer tantos elementos de ese signo como se pueda para ponerlos en `VL[0]`.
- **Invertir** el signo (`s=-s`) y repetir, para poner la siguiente sublista en `VL[1]`.
- Seguir así, invirtiendo `s` en cada iteración hasta que se acaben los elementos de `L`.
- El algoritmo puede ser **destructivo** sobre `L` es decir puede modificar a `L` o no, da igual. Probablemente la solución más simple es la destructiva, extrayendo los elementos de `L` a medida que se insertan en los `VL[j]`.
- Tal vez convenga escribir una función `int sign(int x);` que retorna el "signo" de acuerdo a la convención establecida, es decir `sign(0)` debe retornar `1`.

[Ej. 2] **[sign-join]** Implemente una función
`void sign_join(vector< list<int> > &VL,list<int> &L);` que, dado un vector de listas `VL` generar una lista `L` donde están

- Primero concatenados todos la primera subsecuencia de no-negativos de `VL[0]`, `VL[1]`, ...
- Después los negativos.
- Después nuevamente los no-negativos.
- Asi siguiendo hasta que se acaban todos los `VL`.

Ejemplos:

```
VL: [[1,2,-1,-2,3,4,-3,-4],[5,6,-5,-6,7,8,-7,-8],[-9,-10]],
=> L: [1,2,5,6,-1,-2,-5,-6,-9,-10,3,4,7,8,-3,-4,-7,-8],
```

```
VL: [[0,4,-2,4,1],[-1,-4,-4,-3,-1],[-4,4,1,3,-2],[0,3,-2,4,-5],[-5,-5,-1,-3,-2]],
=> L:[0,4,0,3,-2,-1,-4,-4,-3,-1,-4,-2,-5,-5,-1,-3,-2,4,1,4,1,3,4,-2,-5]

VL:[[-1,-1,-1,3],[-5,3,1,2],[0,2,4,-3],[-3,0,4,-3]],
=> L:[0,2,4,-1,-1,-1,-5,-3,-3,3,3,1,2,0,4,-3],
```

Ayuda:

- Igual que en el ejercicio anterior recomendamos escribir una función `int sign(int x);` que retorna el "signo" de acuerdo a la convención establecida, es decir `sign(0)` debe retornar `1`.
- Extraer todos los que tienen signo `s=1` (o sea los no-negativos) de `VL[0]` y ponerlos en `L`, después los de `VL[1]` y así siguiendo para todos los `VL[j]`.
- Cambiar el signo (`s=-s`) y realizar la misma operación.
- Repetir hasta que todos los `VL[j]` estén vacíos.
- El algoritmo puede ser **destructivo** sobre `VL` es decir puede modificar a `VL` o no, da igual. Probablemente la solución más simple es la destructiva, extrayendo los elementos de los `VL[j]` a medida que se insertan en `L`.

[Ej. 3] **[reverse-stack]** Escribir un programa que invierta una pila `S` utilizando recursión. No está permitido el uso de constructores iterativos (`while`, `for`, etc) ni tampoco el uso estructuras de datos adicionales. Sólo pueden utilizarse métodos de la pila, a saber:

- `S.empty()`
- `S.top()`
- `S.push(s)`
- `S.pop()`

Se propone el siguiente algoritmo recursivo:

- Si la pila está vacia finalizar
- Si no:
  - Guardar una copia del elemento al tope
  - Quitarlo de la pila
  - Invertir el resto de la pila
  - Insertar el elemento guardado al final de la lista invertida

Para el último paso se requiere un método auxiliar, también recursivo, que reciba una lista e inserte un elemento en la base de la misma, siguiendo el siguiente algoritmo:

- Si la pila está vacía, insertar el elemento.
- Si no:
  - Almacenar una copia del elemento al tope
  - Quitarlo de la pila
  - Insertar el elemento recibido al final de la pila restante
  - Apilar el temporal en la pila resultado

Instrucciones generales

- El examen consiste en que escriban las funciones descriptas más arriba; impleméntandolas en C++ de tal forma que el código que escriban **compile y corra correctamente**, es decir, no se aceptará un código que de algún error de compilación o que tire alguna excepción/señal de interrupción en runtime. Básicamente se hace una evaluación de caja negra, aunque le daremos un rápido vistazo al código.

- Salvo indicación contraria pueden utilizar todas las funciones y utilidades del estándar de C++ que por supuesto contiene a la librería STL.
- Se incluye un template llamado **`program.cpp`**. En principio sólo tienen que escribir el cuerpo de las funciones pedidas.
- Para cada ejercicio hay dos funciones de evaluación, por ejemplo si **`f`** es la función a evaluar tenemos

  ```
  ev.eval<j>(f,vrbs);
  hj = ev.evalr<j>(f,seed); // para SEED=123 debe dar Hj=170
  ```

  `j` es el número de ejercicio, por ejemplo para el ejercicio 1 tenemos las funciones (**`eval<1>`** y **`evalr<1>`**). La primera **`ev.eval<j>(f,vrbs);`** toma una serie de casos de prueba de entrada, le aplica la función del usuario **`f`** y compara la salida del usuario (**`user`**) con respecto a la esperada (**`ref`**). Si la verbosidad (el argumento **`vrbs`**) se pone en uno, entonces la función evaluadora reporta por consola los datos de entrada, la salida de la función de usuario y la salida esperada

  ```
  m: 10, k: 3
  T(ref): (10 (7 (4 1) 1) (4 1) 1)
  T(user): (10 (7 (4 1) 1) (4 1) 1)
  EJ1|Caso0. Estado: OK
  ```

- **`ucase`**: Además las funciones **`eval()`** tienen dos parámetros adicionales:
  `Eval::eval(func_t func,int vrbs,int ucase);`
  El tercer argumento 'ucase' (caso pedido por el usuario), permite que el usuario seleccione uno solo de todos los ejercicios para chequear. Por defecto está en **`ucase=-1`** que quiere "hacer todos". Por ejemplo **`ev.eval4(prune_to_level,1,51);`** corre sólo el caso 51.
- **Archivo con casos tests JSON:** Los casos test que corre la función **`eval<j>`** están almacenados en un archivo **`test1.json`** o similar. Es un archivo con un formato bastante legible. Abajo hay un ejemplo. **`datain`** son los datos pasados a la función y **`output`** la salida producida por la función de usuario. **`ucase`** es el número de caso.

  ```
  { "datain": {
    "T1": "( 0 (1 2) (3 4 5 6) )",
    "T2": "( 0 (2 4) (6 8 10 12) )",
    "func": "doble" },
   "output": { "retval": true },
   "ucase": 0 },
  ```

- La segunda función **`evalr<j>`** es el chequeo que llamamos **SEED/HASH**. La clase evaluadora genera una serie de contenedores a partir de la semilla **`seed`**, se los pasa a la función del usuario **`f()`**. Las respuestas de la **`f()`** van siendo procesadas por la función interna de hash que genera un **checksum** **`H`** de las respuestas. Por ejemplo para el primer ejercicio si **`seed=123`** entonces el checksum es **`H=523`**. Una vez que el alumno termina su tarea se le pedirá que corra la función **`evalr<j>()`** de la clase evaluadora con un valor determinado de la semilla **`seed`** y se comprobará que genere el valor correcto del checksum **`H`**.
  Desde el punto de vista del alumno esto no trae ninguna complicación adicional, simplemente debe llenar el parámetro **`seed`** con el valor indicado por la cátedra, recompilar el programa y correrlo. La cátedra verificará el valor de salida de **`H`**.
- En la clase evaluadora cuentan con funciones utilitarias como por ejemplo:
  `void Eval::dump(list <int> &L,string s="")`: Imprime una lista de enteros por **`stdout`**. Nota: Es un método de la clase **`Eval`** es decir que hay que hacer **`Eval::dump(VX);`**. El string **`s`** es un label opcional.

  - **`void Eval::dump(list <int> &L,string s="")`**

- Después del parcial deben entregar el programa fuente (sólo el **`program.cpp`**) renombrado con su apellido y nombre (por ejemplo **`messilionel.cpp`**). Primero el apellido.

TPL1. Trabajo Práctico de Laboratorio 1. [2018-09-06]. TABLA SEED/HASH

```
S=123 -> H1=694 H2=052 H3=382
S=577 -> H1=535 H2=818 H3=993
S=393 -> H1=170 H2=362 H3=339
S=686 -> H1=091 H2=073 H3=821
S=349 -> H1=990 H2=141 H3=103
S=762 -> H1=606 H2=793 H3=379
S=690 -> H1=799 H2=917 H3=108
S=663 -> H1=111 H2=637 H3=098
S=340 -> H1=749 H2=277 H3=666
S=872 -> H1=177 H2=838 H3=753
S=711 -> H1=718 H2=734 H3=561
S=367 -> H1=314 H2=180 H3=026
S=882 -> H1=191 H2=162 H3=362
S=162 -> H1=773 H2=662 H3=833
S=767 -> H1=037 H2=640 H3=463
S=429 -> H1=428 H2=753 H3=567
S=622 -> H1=470 H2=889 H3=948
S=969 -> H1=945 H2=516 H3=027
S=893 -> H1=512 H2=397 H3=903
S=311 -> H1=311 H2=856 H3=032
S=529 -> H1=410 H2=349 H3=191
S=721 -> H1=523 H2=349 H3=906
S=384 -> H1=924 H2=074 H3=755
S=798 -> H1=703 H2=013 H3=175
S=615 -> H1=724 H2=832 H3=202
S=386 -> H1=103 H2=017 H3=751
S=215 -> H1=616 H2=358 H3=165
S=935 -> H1=205 H2=038 H3=267
S=292 -> H1=725 H2=070 H3=450
S=821 -> H1=220 H2=561 H3=598
S=527 -> H1=974 H2=931 H3=921
S=359 -> H1=367 H2=889 H3=479
S=626 -> H1=970 H2=531 H3=405
S=226 -> H1=506 H2=860 H3=796
S=236 -> H1=911 H2=256 H3=523
S=468 -> H1=416 H2=960 H3=436
S=529 -> H1=410 H2=349 H3=191
S=630 -> H1=768 H2=518 H3=773
S=923 -> H1=465 H2=190 H3=256
S=335 -> H1=869 H2=132 H3=653
S=802 -> H1=394 H2=118 H3=715
S=958 -> H1=055 H2=533 H3=510
S=967 -> H1=654 H2=474 H3=448
S=656 -> H1=205 H2=899 H3=791
S=242 -> H1=343 H2=243 H3=953
S=973 -> H1=498 H2=825 H3=398
S=219 -> H1=629 H2=173 H3=949
S=437 -> H1=311 H2=323 H3=598
S=624 -> H1=492 H2=622 H3=763
S=670 -> H1=779 H2=401 H3=206
```