Funcionamiento del Laboratorio

1. Lenguaje y ambiente de trabajo

- El sistema operativo es **Linux**. La compilación y ejecución de los programas se hacen desde una terminal del sistema. El código se edita en el editor de texto preferido de cada estudiante.

- El lenguaje de programación es **C** con algunas funcionalidades de **C++**:

  - Operadores **new** y **delete**.
  - Pasaje por referencia.
  - Tipo **bool**.
  - Uso de **struct** y **enum** al estilo **C++**.

- El compilador es **g++**, que debe instalarse en caso de no estar instalado.

  Para instalarlo, por ejemplo en Ubuntu:

  $ sudo apt install g++

  En Fedora:

  $ sudo dnf install gcc-c++

- Se debe tener instalada la utilidad **valgrind** para el análisis del uso de la memoria.

  Para instalarlo, por ejemplo en Ubuntu:

  $ sudo apt install valgrind

  En Fedora:

  $ sudo dnf install valgrind

- Se deben tener instaladas las utilidades **make**, **tar** y **gzip**.

2. Materiales

Los materiales para realizar cada tarea se extraen del archivo *MaterialesTareaN.tar.gz* (donde N es el número que identifica a la tarea) que se obtiene en la carpeta *Materiales* de la sección *Laboratorio* del sitio del curso.

Para desempaquetar el material se puede usar la utilidad *tar* desde la línea de comandos:

$ tar zxvf MaterialesTareaN.tar.gz

Los archivos que están en MaterialesTareaN.tar.gz están dispuestos en una estructura de directorios, **la cual debe conservarse**.

- En la raíz deben estar el módulo principal (**principal.cpp**), el **Makefile** y el ejecutable que se generará tras la compilación.
- Los archivos de encabezamiento (**.h**, *headers*) están en el directorio **include**.
- Los archivos a implementar (**.cpp**) deben crearse en el directorio **src**.
- Los archivos que resultan de la compilación de cada módulo, (**.o**), se mantienen en el directorio **obj**.
- Los casos de prueba están en el directorio **test**. Los archivos con extensión `.in` (entradas) contienen una secuencia de comandos. Por cada uno de estos archivos hay uno con extensión `.out` (salidas) que contiene la respuesta esperada. Los archivos generados al hacer las pruebas mediante `make` van al directorio **test/salidas**. Los archivos con extensión `.sal` contienen el resultado de la ejecución. Los archivos con extensión `.diff` contienen la comparación del `.sal` con el correspondiente `.out`.

3. Desarrollo

- Para cada archivo *.h* del directorio *include* en el que se declaren tipos o funciones sin implementar se debe crear en el directorio *src* el archivo *.cpp* correspondiente e implementar en él dichos tipos y funciones.

  En particular se debe implementar la representación de un tipo, `struct _rep_`*tipo*. El texto `_rep_` solo puede ocurrir como prefijo de *tipo*. En otro caso se producirá un error de compilación.

- Se compila para obtener el ejecutable principal:

  $ make

- Se ejecuta un caso de prueba (sin valgrind) y se verifica que la salida generada por el programa es igual a la salida esperada (donde NN es el identificador del caso de prueba siendo ejecutado):

  $ ./principal < test/NN.in

  Si no se detectan errores se hace una prueba más rigurosa:

  $ ./principal < test/NN.in > test/salidas/NN.sal
  $ diff test/NN.out test/salidas/NN.out

  Si los archivos son iguales no se imprime nada. Si son diferentes se muestra las líneas en las que difieren y el contenido de ellas.

  Puede ser preferible guardar el resultado de la comparqación en un archivo:
  $ diff test/NN.out test/salidas/NN.out > test/salidas/NN.diff

  El mismo resultado, ejecución del caso y comparación con el resultado esperado se obtiene mediante
  $ make test/salidas/NN.diff
  Los archivos **sal** y **diff** quedan en el directorio test/salidas.

De manera abreviada se puede ejecutar
$ make t-NN

En este caso no se generan los archivos, sino que el resultado se muestra en la salida estándar.

Para tener en cuenta el correcto uso de la memoria y ser notificado de posibles errores de memoria, se ejecuta **además** el siguiente comando:

$ valgrind –leak-check=full ./principal < test/NN.in > test/salidas/NN.sal

La regla *testing* tiene en cuenta ambos aspectos y permite probar todos los casos de prueba:

$ make testing

En el directorio **test/salidas** quedan los archivos **sal** y **diff** correspondientes a cada uno de los casos.

Se sugiere solo usar esta regla al final, después de haber probado caso, para confirmación.

Con `make testing` los casos se prueban usando también la utilidad `timeout`:

$ timeout 4 valgrind -q –leak-check=full ./principal <test/01.in >test/01.sal

`timeout` es una utilidad que establece un plazo para la ejecución de un proceso (4 en este ejemplo). Si la ejecución demorara más de ese plazo `timeout` terminaría ese proceso.

Hay dos motivos por los que se incluye esta utilidad

1. Cortar la ejecución debido a que se entró en loop.
   En casi todos los casos el tiempo de ejecución debería ser solo una fracción de segundo por lo que si se llega al tiempo establecido como plazo, que es muy amplio, significa que es muy probable que haya un loop.
2. Algunos casos están especialmente diseñados para medir la eficiencia de la implementación.
   En estos casos el tiempo de ejecución esperado suele ser de algunos segundos y el plazo que se establece se hace más ajustado que en los casos comunes.