Onboarding

Objetivos	● Tener un ambiente de desarrollo y testeo local. ● Familiarizarse con Github y el Sercom. ● Repaso de temas de compilación, memoria y C++.
Entregas	● **Entrega obligatoria:** clase 3.
Cuestionarios	● Onboarding - Recap 01 - Proceso de Building y Testing ● Onboarding - Recap 02 - Memoria en C++ ● Onboarding - Recap 03 - Librería Estándar de C++ ● Onboarding - Recap 04 - Debugging
Criterios de Evaluación	● Criterios de ejercicios anteriores. ● Resolución completa (100%) de los cuestionarios *Recap.* ● Entrega digital en el Sercom con la solución correcta al ejercicio *Contador de Palabras*.

El trabajo es personal*: debe ser de autoría completamente tuya. Cualquier forma de* ***plagio es inaceptable****: copia de otros trabajos, copias de ejemplos de internet o copias de tus trabajos anteriores en otras materias (self-plagiarism).*
Si usas material de la cátedra deberás dejar en claro la fuente y dar crédito al autor (a la materia).

Índice

Introducción

La idea de este *onboarding* es contarte cómo trabajaremos durante el cuatrimestre y guiarte en los primeros pasos.

En la primera mitad del cuatrimestre realizarás entregas individuales que resuelven pequeñas partes del juego a implementar.

Durante esta etapa aprenderás y pondrás en práctica distintos conceptos de C++, POO, sockets, threads y manejo de *deadlines*.

En la segunda mitad del cuatrimestre formarás un grupo de 3 personas para completar la implementación del juego.

En este momento practicarás no solo programación sino organización, coordinación y trabajo en equipo.

En la vida profesional siempre trabajarás con otras personas y saber hacerlo es tan importante como saber como codear en C++.

Va a ser una travesía exigente, pero te daremos todas las herramientas para que la completes.

¿Arrancamos?

Entorno de Trabajo

El juego está planeado para correr en plataformas Linux, en particular Ubuntu.

Antes de empezar a desarrollar es importante que tengas un buen ambiente de trabajo, que te sea fácil de usar y que a la vez sea lo más parecido posible al ambiente donde se probará el juego.

La guía Entorno de trabajo te ofrece 3 alternativas principales para configurar tu máquina.

¡Cuanto antes tengas tu entorno mejor!

Inscripción al Sercom

Con compilar no va a alcanzar. Los entregables deberán tener una buena performance y calidad.

En Taller tenemos un sistema de *integración continua* (o CI por sus siglas en inglés) llamado Sercom

Para usarlo deberás tener una cuenta gratuita en Github y luego realizar una inscripción al Sercom y al correspondiente curso como lo muestra la guía Inscripción al Sercom

Repositorios de Github

Al inscribirte el Sercom te creará 3 repositorios privados en Github y te agregará como colaborador.

En estos repositorios será en donde programes las distintas partes del juego en la etapa individual. Para la parte grupal será el grupo quien decida donde tendrá su repositorio.

- *Onboarding*: es parte de este onboarding y lo usarás para practicar y familiarizarte con el Sercom.
- *Sockets*: aquí te enfocarás en resolver una parte de las comunicaciones entre un solo cliente y el servidor del juego.
- *Threads*: aquí extenderás el diseño y le darás soporte al servidor para que acepte múltiples clientes en concurrencia.

Para este onboarding podes ignorar los repositorios de sockets y threads. Ya los usarás más adelante.

Pre-commit

Con *git* puedes instalar unos hooks que se ejecuten en cada commit y así automatizar la ejecución de linters y checkers.

Primero necesitas instalar *pre-commit*

```
pip install pre-commit
```

Luego, en cada uno de los 3 repositorios que te descargaste tenes que instalar los hooks. Para ello entrá a la carpeta del repositorio y corré:

```
pre-commit install
```

Te recomiendo que hagas primero el *recap 01 sobre compilación*. En él verás un poco más sobre `pre-commit` y los checkers que usamos en Taller: `cpplint`, `cppcheck` y `clang-format`.

Onboarding - Contador de Palabras

El Contador de Palabras es un ejercicio super simple. Es más, es tan simple que junto con el enunciado también encontrarás la solución. El *catch* es que la misma se encuentra plagada de errores de distinta índole, como *overruns* y *leaks*, entre otros.

La tarea es sencilla. Deberás identificar y arreglar **todos** los errores, utilizando el output del Sercom como ayuda y otros recursos, ya sean las páginas de manual, los recursos de la cátedra, las referencias (*cppreference, cplusplus*), los recaps o el *Google-Fu*.

Existen varias maneras de resolver este ejercicio. A través del Sercom, podrás entregar el código “bugeado” para que este te de una salida concreta sobre los errores de la solución. De ahí, modificarás localmente la

solución para corregir estos errores, y una vez hecho esto, volverás a entregar la solución al Sercom.
Otra manera será testeando la solución directamente en **tu entorno local**. En ese caso, deberás compilar, ejecutar y encontrar los errores con el output del compilador y de los tests que le corras.

La opción es tuya, pero recordá que el Sercom impone un límite en la cantidad de entregas que podes hacer y además es lento. **Probar en tu máquina local es siempre más preferible.**

En este *onboarding* la idea es que no es que programes la solución, sino que pruebes tu ambiente local, como correr los hooks y tests *localmente*, que sepas como Sercom te puede ayudar a identificar los errores, que practiques como commitear en Git, como hacer un release en GitHub y cómo hacer una entrega en el Sercom.

Compilar localmente

Cada repositorio tiene un `Makefile` inicial. Basta con que ejecutes `make` para que se compile tu código.

Probar los tests localmente

Descargate del Sercom el zip `testcases.zip`, anda a la carpeta raíz del repositorio y descomprimi el zip ahí.

Deberías tener algo así:

```
onboarding/
    casos/
    salidas/
    run_tests.sh
    compare_outputs.sh
```

Tanto en la carpeta `casos/` como en la carpeta `salidas/` verás varias subcarpetas. Cada una es un caso de prueba. Dentro de `casos/` están los archivos iniciales que se requieren para ejecutar el test y en `salidas/` están las salidas esperadas.

El script `run_tests.sh` es quien ejecuta todos los tests. El script **no** compila tu TP así que tenes que haber hecho un `make` **antes**. Segun el TP en el que estes lo tenes que ejecutar de una u otro manera:

Onboarding:

```
./run_tests.sh . casos/ no-server no-valgrind 60 10
```

TP de Sockets:

```
./run_tests.sh . casos/ single-client no-valgrind 60 10 no
```

TP de threads:

```
./run_tests.sh . casos/ multi-client no-valgrind 60 10 yes
```

El parámetro `no-valgrind` lo podes cambiar por `valgrind` para que los tests se ejecuten con Valgrind.

Los números 60 y 10 son los timeouts: el primero es el primer timeout de un test y si el programa está colgado y no responde, el segundo timeout entra en acción y hace un kill al programa. Es posible que quieras incrementar esos números si corres los tests con Valgrind.

Luego de la ejecución, las salidas generadas (las obtenidas) quedarán guardadas en las subcarpetas dentro de `casos/`

Para comparar usaras el segundo script `compare_outputs.sh`:

```
./compare_outputs.sh casos/ salidas/
```

El script te imprimirá las diferencias encontradas usando `diff`. Ten en cuenta que si algún test te timeouteo, `tests_runner.sh` **aborta** la ejecución y hará que haya muchas más diferencias al correr `compare_outputs.sh`.

Si la salida de `diff` te confunde podes comparar vos mismo los resultados con otro programa como `meld`.

Probar un test local y manualmente

Es necesario que sepas cómo correr un caso manualmente por que te va a permitir experimentar con él y correrlo con un debugger. *¿Ya hiciste el recap de GDB?*

Cada subcarpeta dentro de `casos/` es un caso de prueba. En ellas habrá un archivo `*args__` (en el *onboarding* habrá uno solo, en el resto de los TPs habrá un `*args__` para el cliente y otro para el servidor). Este archivo contiene los **parámetros de la línea de comando** para ejecutar tu programa. Así sabrás como el test ejecuta tu entrega.

Además de `*args__` habrán archivos `*stdin__`. Estos son la entrada estándar que deberás redireccionar a cada programa. *¿Ya hiciste el recap de Proceso de Building y Testing?*

En el TP *onboarding* tendrás solo un único programa (por lo tanto un único `*args__` y `*stdin__`); en el TP de *Sockets* tendrás 2 programas, `client` y `server`, por lo que tendrás 2 `*args__` y 2 `*stdin__`.

En el TP de *Threads* tendrás también 2 programas, `client` y `server`, pero cada test lanzará **múltiples clientes** así que tendrás **múltiples** `*args__` y `*stdin__` (verás que cada uno está numerado y corresponde al número de cliente). Es muy posible que quieras solo usar algunos y debuggear un solo cliente en estos casos.

Correr los pre-commit hooks manualmente

Para correr los linters tendrás que ejecutar **manualmente** los hooks de `pre-commit`. Reemplaza `` `xxx.cpp` `` por el o los archivos `.h` y `.cpp` que tengas:

```
pre-commit run --hook-stage manual --files xxx.cpp
```

En este *onboarding* los hooks solo se ejecutan **manualmente**. En el resto de los TPs será de forma **automática en cada commit.**

Entrega digital en el Sercom

Luego de que hagas las modificaciones que solucionen los problemas en el código y hayas verificado localmente tu código estarás listo para hacer una entrega en el Sercom.

En Entrega digital en GitHub y Sercom verás cómo hacerlo paso a paso

Será **condición necesaria** que el código del Contador de Palabras, luego de tus correcciones, pase todos los tests que se encuentren en Sercom..

Aun si tu entrega corre y pasa todas las pruebas en el Sercom eso no significa que la entrega está aprobada. En Taller no solo buscamos que programes sino también que programes bien.

Proceso de code review

Cuando el período de entrega termine el Sercom dejará de aceptar entregas y comenzará el periodo en donde los docentes tendrán una semana para hacer el code review. Leeremos tu código y marcaremos todos los errores que veamos.

Tendrás luego una semana más para hacer la correcciones, hacer un nuevo release y subirlas al Sercom.

Un comentario: para el Contador de Palabras **no tendrás ningún code review**. No tiene sentido hacerlo si fuimos nosotros quienes te dimos el código con la solución.

Para compensar esto y solo para este *onboarding*, en el Sercom podras bajarte otro zip que tiene la solución propuesta por la cátedra para que la veas y la **compares** con tu solución. No hay una solución perfecta pero hay soluciones más robustas que otras, más simples que otras y menos propensas a errores que otras. Usa `diff`, `meld` u otro para comparar los archivos fácilmente.

Cuestionarios Recaps

En el Sercom también encontrarás unos ejercicios adicionales, los *Recaps*.

Son cuestionarios *multiple choice* con el objetivo de que repases ciertos temas vistos en clase o que complementan a estas.

Tendrás tantos intentos como desees y en todas las preguntas de esos cuestionarios hay uno o varios links a las respuestas (o a la documentación oficial con las respuestas).

¿Por qué? Porque codear a ciegas, "solo probando a ver si anda", es una manera muy ineficiente de trabajar. Los recaps están para guiarte y darte una idea de que herramientas hay y donde hay buena documentación.

Porque saber C++ está bueno, pero saber dónde buscar las cosas que aun no sabes es mucho mejor. Será un skill que usarás en Taller y en el resto de tu vida profesional, sea para C++, Rust, Python o Brainfuck.

Para este proceso de onboarding habrá 4 cuestionarios sobre:

- el proceso de compilación
- el manejo de memoria en C++
- la librería estándar de C++
- el uso de un debugger

Recordá que tenes el Discord para las consultas!

Cronograma

Este diagrama es el cronograma tentativo del cuatrimestre. Cada *slot* representa 1 semana, arrancando por el primer día de clases.
Tenes en que semanas se habilitarán en el Sercom las entregas, cuales son los deadlines (fechas de entrega obligatorias) y las semanas de code review.