

Ejercitación Clase Práctica 2

Programación en C++

Ejercicio3: ejercicios avanzados

Abrir el proyecto Ejercicio3. Recordar de abrir “New window” para crear una nueva conexión de docker con su respectivo compilador y herramientas.

Luego abrir con “Open folder”.

Ejercicio login

El sistema SUVE posee un protocolo de seguridad para constatar que una tarjeta pertenece al sistema.

Este protocolo consiste en que el sistema instalado en los transportes le envíen un código aleatorio de 5 dígitos a la tarjeta, la cual procesa cada dígito con el siguiente algoritmo:

- Convierte cada número recibido al término iésimo de la sucesión de fibonacci
- Agrega 1 número extra de seguridad al final, que debe corresponder a la cantidad de Zettabytes de información consumida globalmente en el último año.
- Modificar “login_suve.cpp” de manera que la tarjeta pida los 5 números de seguridad.
- Implementar la función login del archivo “lib/seguridad.cpp” que se comporte de acuerdo al algoritmo descripto.
 - *Aclaración:* Si se desea usar una función de otro archivo, debe incluirse en el archivo en que se usará la siguiente instrucción:

```
#include "nombre_archivo_que_define_la_funcion.h"
```
- Compilar el proyecto
- Notar que hay 2 ejecutables. Ejecutar el correspondiente al login.
 - Ejemplo: 8 2 9 3 2 debe dar 21 1 34 2 1 79

Ejercicio mayor racha

Por último, el sistema SUVE quiere saber calcular la mayor cantidad de viajes en colectivos “rápidos” de forma consecutiva por una persona.

Detalle: Los colectivos tienen distintos valores de acuerdo a sus recorridos, pero además si son “el rápido” cuestan más caros.

Vamos a tener el registro de uso de nuestra “Suve” con los valores de cada viaje comprado en orden temporal.

Por simplicidad se usarán los siguientes valores de viaje (del más barato al más caro): 18, 22, 27, 35 para colectivos normales 40, 48, 50 para colectivos rápidos (cualquier parecido con la realidad es pura coincidencia).

Ejemplos:

- `<40, 50, 18, 40, 48, 50, 48, 40>`, empieza viajando 2 veces consecutivas en rápidos, después se toma un común y luego viaja en 5 rápidos más. El resultado será entonces 5, ya que es mayor que 2.
- `<18, 40, 50, 27, 40, 50, 40, 48, 50, 22, 40, 50, 40, 48, 50, 40, 27, 40, 50, 40>`, empieza viajando 2 veces consecutivas en colectivo rápido, luego 5, luego 6 y por último 3. La mayor cantidad de viajes en colectivo rápido consecutivos es entonces 6.
- `<22,18,40>`, el resultado es 1.

Enunciado:

- Abrir el archivo “mayor_racha_test.cpp”
- Compilar y ejecutar.
- Implementar la función `mayor_racha` en “mayor_racha.cpp” de manera que todos los test ejecuten satisfactoriamente.

Debugger

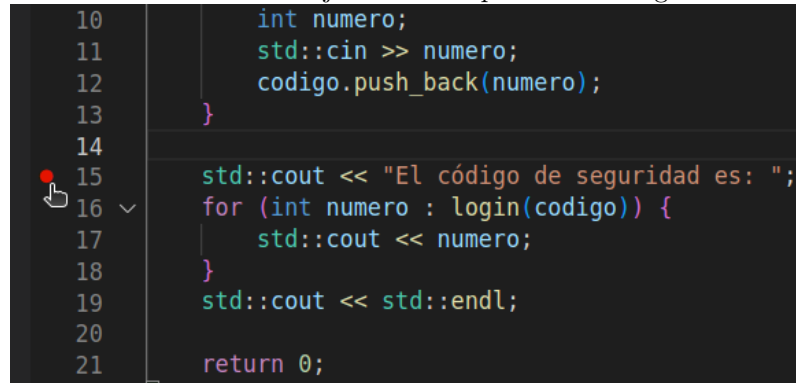
El debugger es una herramienta que permite ejecutar el código instrucción por instrucción, además de mostrar el estado de las variables en cada paso.

Se trata de una herramienta muy útil para identificar errores en el código.

Uso del debugger

- Abrir el ejercicio 3 (resuelto)
- Para usar el debugger, el primer paso es especificar en cuáles líneas de código se quiere pausar la ejecución; de manera de darle al usuario la decisión de cómo seguir ejecutando el programa a partir de esa línea.

- Para especificar una línea en la cuál pausar la ejecución, click-ear en la barra lateral izquierda de la línea en cuestión. El resultado es un círculo rojo cómo se aprecia en la siguiente foto:



- De ahora en más, lo llamaremos *breakpoint*.


- Luego, iniciar el debugger clickeando en el botón de la barra inferior  y seleccionando el programa “login_suve”.
- Cuando el programa llegue a un *breakpoint*, se detendrá, y el usuario podrá interactuar con la ejecución del programa por medio del siguiente panel:



Figure 1: panel debugger

De izquierda a derecha, las funciones de los botones son:

1. continúa la ejecución hasta el próximo breakpoint;
2. ejecuta una instrucción;
3. ejecuta una instrucción, pero si hay un llamado a función, ingresa a la función y ejecuta línea a línea;
4. ejecuta hasta finalizar la función, luego pausa nuevamente;
5. reinicia el programa;
6. termina la ejecución.

Además, en el panel izquierdo se visualiza el estado de las variables.

Enunciado:

- Agregar un breakpoint a la línea 15 del archivo login_suve.cpp
- Agregar un breakpoint a la línea 7 en el archivo seguridad.cpp
- Ejecutar el debugger y ver el estado de las variables
- Continuar hasta el próximo breakpoint

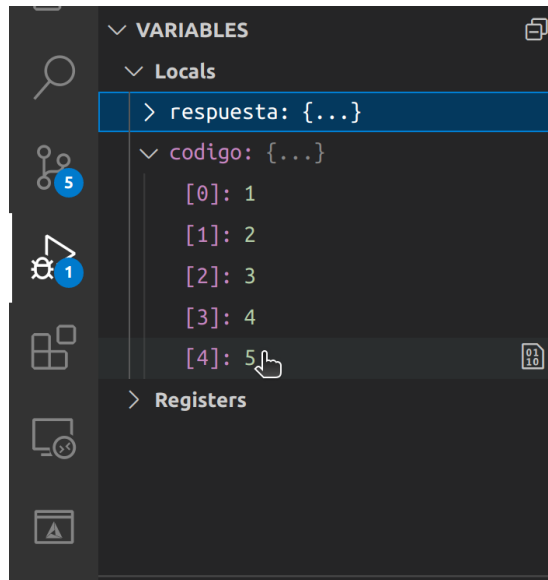


Figure 2: estado de las variables

- Ejecutar la función login línea a línea (sin entrar a los push_back)
 - analizar el scope de la variable iesimo_termino

Pasaje de parámetros

- Cambiar la signature de la función login. Debe pasar el código por referencia
- ¿en qué casos es preferible usar referencias en lugar de pasar valores por copia?
- Volver a cambiar la signature y modificar la implementación de la función acorde para que devuelva el resultado en dicha variable

Tests

Los tests sirven para poder chequear si el código hace o no lo que queremos (y esperamos), a medida que se implementa.

Un test ejecutará alguna funcionalidad de lo que programamos y comparará el resultado *obtenido* con un resultado *esperado* que ya conocemos de antemano.

Se proveen test desde el ejercicio 3 en adelante.

Enunciado:


- Agregar un test que chequee que el ejemplo 3 del enunciado se ejecuta correctamente. Es decir, que `mayor_racha` de `<22, 18, 40>` debe dar por resultado 1.
- ¿es posible asegurar solo con tests que mi programa hace lo que quiero que haga?

Google Tests

Para la materia usaremos la biblioteca específica de *Google Tests* para implementarlos.

Esta biblioteca provee la macro `EXPECT_EQ` que comprueba que los 2 argumentos que recibe sean iguales.

Enunciado:

- Abrir el proyecto “Ejercicio 3 (google tests)”
- Compilar el proyecto con `build`
- Ejecutar los tests utilizando el plugin de visual studio code de la barra lateral. Símbolo 

Estructuras de datos

- Cambiar los precios de los viajes de vector a set
- ¿por qué un set representa mejor a los precios de los viajes comparado con un vector?
- ¿por qué no puedo usar set para el historial de viajes?