Introducción al Desarrollo de Software I - Lanzillotta

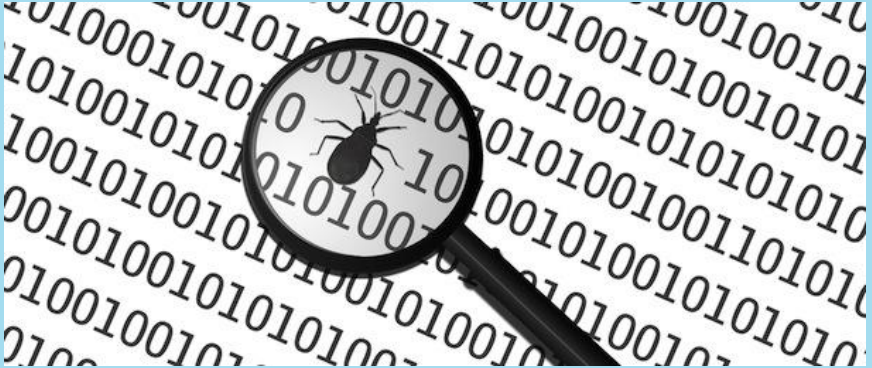
Capra - Chaves - Di Matteo - Sosa - Villegas - Palavecino

.UBAfiuba
FACULTAD DE INGENIERÍA

Agenda

- Debugging
- Ejercicios de debugging y testing

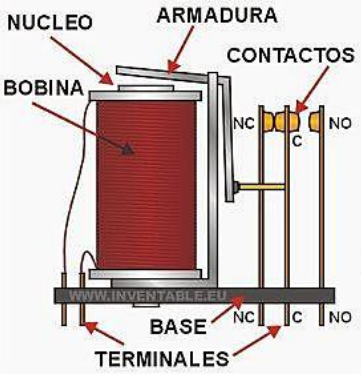
Debugging



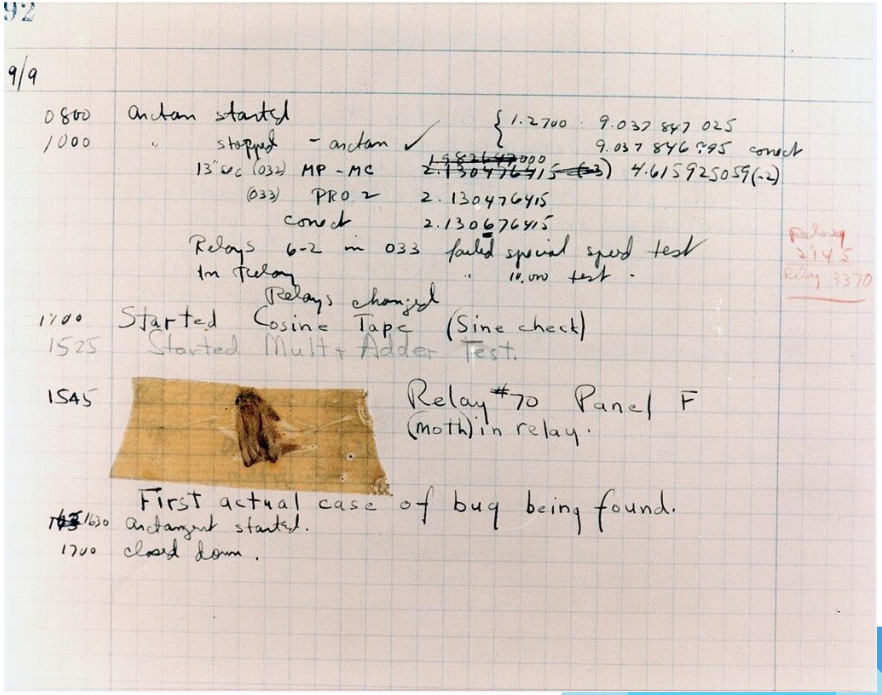
¿Qué es un Bug?

- Un bug es un error de código en un programa informático
  - Podemos decir que un bug es cualquier comportamiento error o fallo de un programa
- Podemos dividir los bugs en categorías:
  - De sintaxis
  - De ejecución
  - De diseño

Origen del término



Relé Electromagnético



¿Qué es Depurar o Debuggear?

- Es el proceso que consiste en la deteccion y correcion de bugs.
  - En inglés se dice “debugging” y en la mayoría de los ambientes van conocerlo de esta forma
- Es un proceso más que habitual en el ambiente del desarrollo tanto en la etapa de creación como en la de mantenimiento de un programa.

¿Qué es un Debugger o Depurador?

- ➔ Es una herramiento que se utiliza para el proceso de debugging.
- ➔ Existes muchos debuggers pero casi todos ofrecen las mismas posibilidades.
- ➔ Algunos debuggers son específicos de un lenguaje.
- ➔ Uno de los debuggers más famosos o que más van a escuchar es GDB: GNU Debugger.

¿Qué puede hacer un Debugger?

- ➔ Puede detener la ejecución del programa e irlo ejecutando linea a linea. Hasta puede ejecutarlo instrucción a instrucción.
- ➔ Brinda información sobre variables, scopes, threads, stack trace y de casi todo lo que compone al programa en sí.

El “debugger” más primitivo

- ➔ Una forma instintiva de hacer debugging es usar “prints” en partes específicas del código.
- ➔ Un gran poder conlleva una gran responsabilidad. Usen prints como último recurso...

GDB: GNU Debugger

ajustense los cinturones y a recenle al dios en el que crean

GDB: instalación

- ★ Ejecutar: *sudo apt install gdb*
- ★ Comprobar la instalación: *gdb –version*

```
fran@hp840g7 ~ gdb --version
GNU gdb (Ubuntu 14.0.50.20230907-0ubuntu1) 14.0.50.20230907-git
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

GDB: iniciando

- Para inicial en C debemos primero compilar el programa con el flag *–debug*.
  - *gcc <file.c> –debug -o <nombre programa>*
- Luego podemos iniciar la versión con interfaz de gdb con
  - *gdb -tui <nombre programa> <argumentos>*
- Luego tocamos la tecla '*c*' para continuar hacia el inicio del proceso
- Los argumentos también pueden setearse luego

Breakpoint

- Un breakpoint o punto de parada sirve para que al debuggear el debugger detenga la ejecución en la línea donde colocamos el breakpoint.
- Agregar un breakpoint en la linea de codigo indicada
  - Comando: b <número de la línea de código>
- Agregar un breakpoint en la primera linea de codigo de la función
  - Comando: b <nombre de la función>

Breakpoint condicional

- ➔ Es un breakpoint que tiene asociada una condición. El programa se detendrá en esa línea si y sólo si se cumple la condición asociada.
- ➔ Cada breakpoint tiene asociado un “id” y ese “id” es el que debe usarse.
  - Comando: *b main*
    - output: Breakpoint 1 at 0x8048508: file ejemplo.c, line 48.
  - Comando: *condition <id del breakpoint> <condición>*
    - Ejemplo: *condition 1 fecha == 1012025*

Comandos: Continue

- Este comando permite continuar con la ejecución hasta el siguiente breakpoint o hasta el fin del programa.
  - Comando: *continue* | *c*

Comandos: Next

- Permite avanzar a la línea inmediatamente siguiente o si existe algún breakpoint intermedio, se detiene en el.
  - Comando: *next* | *n*

Comandos: Step into

- Permite ingresar a la función de la siguiente función a ser llamada.
- Nos da la posibilidad de examinar el contenido de una función sin necesidad de agregar un breakpoint interno y poder ver los valores en el scope de esta función.
  - Comando: *step* | *s*
- En lo más bajo: *stepi* | *si* permite avanzar una única instrucción a nivel máquina.

Comandos: Step Out

- ➔ Sale de la función actual hacia la función que la llamó.
  - Comando: *finish*
- ➔ Si en el flujo de ejecución existe un endpoint se detendrá en el.

Información de variables

- Permite observar el contenido en ese instante de ejecución de las variables.
  - Comando: *print | p <nombre de la variable>*
- También se le puede dar “seguimiento” a una variable y cada vez que es usada podemos hacer que el programa se detenga.
  - Comando: *watch <nombre de la variable>*

Seteo de variables

- Permite cambiar el valor de las variables en tiempo de ejecución pudiendo así llegar a casos bordes o permitiendo hacer análisis del comportamiento del programa en ese contexto.
  - Comando: *set <nombre de la variable> = <valor>*

Stack trace

- ➔ Muestra la pila de ejecución del programa en ese momento. Nos permite observar en qué función estamos, quién y dónde se llamo a esa función.
- ➔ Un stack trace es un informe de los elementos activos en la pila de ejecución en un momento determinado durante la ejecución de un programa.
  - Comando: *backtrace* | *bt* | *backtrace full*

5) Refactorizar el código si es necesario

```
(gdb) bt
#0  mostrar (fecha=1012025) at debugger.c:4
#1  0x000055555555551a1 in main () at debugger.c:13
```

```
(gdb) bt full
#0  mostrar (fecha=1012025) at debugger.c:4
No locals.
#1  0x000055555555551a1 in main () at debugger.c:13
        fecha = 1012025
```

Evaluar expresiones

- Podemos evaluar expresiones con el contenido de las variables en ese instante de ejecución o cualquier otra expresión que nos resulte útil.
- Solo tenemos que escribir la expresión que deseamos. Podemos utilizar las variables presentes en ese scope.

Otros debuggers

- Python:
  - Como hay GDB, Python tiene PDB.
  - Documentación
- VSCode, JetBrains, etc:
  - No hay magia. Todos los debugger son parecidos y similares a GDB
- JS con chrome
  - Podemos buscar un ejemplo por internet.
  - Ejemplo

Flask

- Repo con ejemplos: link. Info de flask: link
- Ejecutar en modo debug: *flask --app hello run --debug*
- Ejecutar desde python en modo debug: *app.run(debug=True)*
- Tambien se puede desabilitar el debugger de flask:

  app.run(debug=True, use_debugger=False, use_reloader=False)

Flask

- ¿Como vamos a debuggear? Como siempre
- NUNCA JAMÁS ACTIVAR EL MODO DEBUG EN PRODUCCIÓN

¿Dudas?

¿break y ejercitamos?

Ejercitación

Tomando mate

Vamos a escribir tests de una clase y a debuggear para encontrar los errores

Loteria

Vamos a debuggear para encontrar los errores.

¡ahora solos!

Números primos

Para el siguiente código escribir tests de los casos y debuggear para corregir los errores

```
def es_primo(n):
    for i in range(2, n):
        if n // i == 0:
            return False
    return True

def primos_entre(a, b):
    primos = []
    for numero in range(a, b):
        if not es_primo(numero):
            primos.append(numero)
    return primos
```

Fibonacci

Para el siguiente código escribir tests de los casos y debuggear para corregir los errores

```
def fibonacci(n):
    if n < 2:
        return 1
    else:
        return fibonacci(n - 2) + fibonacci(n - 1)
```

Obs: el número de fibonacci de N es la suma del número de fibonacci de N-1 y N-2.

Para la clase que viene...

- Ejercitación del parcial
- Consultas


.UBAfiuba
FACULTAD DE INGENIERÍA