Universidad Autónoma de Baja California

Facultad de Ingeniería, Arquitectura y Diseño

Ingeniería en Software y Tecnologías Emergentes

Materia: Organización de Computadoras

Jonathan Crespo Ragland

Excepciones

Alumnos:
Carlos David Cruz Santiesteban 379909
Erick Lizarraga Beltran 377995
Pablo Ivan Berumen Verdugo 380341

27 de Noviembre de 2025

Trabajo en clase: Excepciones. Equipos de 2 a 3 personas

Investigar lo siguiente:

1. Que son Breakpoints dentro del contexto de programación.
2. De acuerdo al IDE que más utilizan, investigar cómo se pueden utilizar Breakpoints.
3. Generar un snippet de código en ensamblador que simule el uso de un bloque de Try/Catch en ensamblador, utilizando saltos condicionales/incondicionales (No tiene que compilar).
4. Realizar un programa en C que utilice las funciones setjmp y longjmp para simular el manejo de excepciones.
5. Realizar un snippet de código que utilice los breakpoints en su IDE (En el lenguaje de su elección).

El entregable será un documento con evidencia de cada uno de los puntos, y los snippets de código que generaron. Los códigos se agregan al repositorio de código más adelante.

La siguiente clase se seleccionará equipos al azar para que presenten ante la clase sus códigos, explicando su funcionamiento.

El código en C debe tener un objetivo o una regla de negocio en particular, no puede ser por ejemplo un código que solo sea una división y evalúe que el divisor sea 0.

1.- ¿Qué son los Breakpoints en Programación?

Los breakpoints (puntos de interrupción) son herramientas esenciales en la depuración (debugging) de software.

Definición: Un breakpoint es una parada intencional que un programador inserta en una línea específica de código ejecutable.

Función: Cuando el programa se ejecuta en un depurador (debugger), la ejecución se detiene automáticamente en la línea marcada.

Utilidad: Esto permite al programador inspeccionar el estado del programa en ese momento:

- Ver los valores de las variables y la memoria.
- Examinar la pila de llamadas (call stack).
- Ejecutar el código paso a paso (step over, step into, step out) a partir de ese punto para observar el flujo de control y detectar dónde se producen los errores (bugs).

En esencia, permiten una "radiografía" del código en tiempo de ejecución.

2.- Uso de Breakpoints en Visual Studio Code (VS Code)

El uso de Breakpoints en Visual Studio Code se resume en tres etapas principales:

1. **Colocar el Breakpoint:**
    - Acción: Haz clic en el margen izquierdo junto al número de línea deseado.
    - Atajo: Presiona F9 con el cursor en la línea.
    - Evidencia: Aparece un círculo rojo en la línea.

2. **Iniciar la Depuración:**
    - Acción: Presiona la tecla F5 o haz clic en el icono de "Run and Debug" (el insecto) en la barra lateral y luego en "Start Debugging".
    - Resultado: La ejecución del programa se detendrá automáticamente en la línea marcada.

3. **Inspeccionar y Controlar:**
    - Inspección: Usa el panel lateral "Variables" para ver los valores actuales de las variables, y "Call Stack" para ver la secuencia de llamadas.
    - Control (Stepping): Utiliza las teclas o la barra de control flotante para moverte por el código:

        - F10 (Step Over): Ejecutar la línea actual y pasar a la siguiente (sin entrar en funciones).
        - F11 (Step Into): Entrar a la función que se llama en la línea actual.
        - F5 (Continue): Seguir la ejecución hasta el siguiente breakpoint.

3.- Generar un snippet de código en ensamblador que simula el uso de un bloque de Try/Catch en ensamblador, utilizando saltos condicionales/incondicionales (No tiene que compilar).

```
section .data
   err_code    dd 0       ; 0 = ok, 1 = división por cero

section .text
global _start

_start:
   mov    eax, 10
   mov    ebx, 0         ; divisor

   cmp    ebx, 0
   je     .throw_div_zero  ; si divisor es 0
   jmp    .try_end ; si divisor no es 0

.throw_div_zero:
   mov    dword [err_code], 1 ; guardamos 1 porque salio mal
   jmp    .catch

.try_end:
   mov    dword [err_code], 0 ; guardamos 0 porque salio bien
   jmp    .after_catch

.catch:
   mov    eax, [err_code] ; se carga el codigo del error 0 o 1
   cmp    eax, 1
   je     .handle_div_zero ; si es 1
   jmp    .after_catch

.handle_div_zero:
   jmp    .after_catch

; salida del programa
.after_catch:
   mov    rax, 60        ; exit
   xor    rdi, rdi
   syscall
```

4.- Realizar un programa en C que utilice las funciones setjmp y longjmp para simular el manejo de excepciones.

```
#include <stdio.h>
#include <setjmp.h>

// Buffer que guarda el punto seguro:
jmp_buf contexto;

// Códigos de excepción:
#define EX_NO_MEMBRESIA 1

// Regla de negocio: validar acceso:
void accederPremium(int tieneMembresia) {
  if (!tieneMembresia) {
    longjmp(contexto, EX_NO_MEMBRESIA);  // Lanzar excepción:
  }
  printf("Acceso concedido al contenido premium.\n");
}

int main() {
  // Guardamos el estado inicial:
  int ex = setjmp(contexto);

  if (ex == 0) {
    printf("Intentando acceder sin membresía...\n");
    accederPremium(0);  // Esto desencadena la excepción:
  } else {
    if (ex == EX_NO_MEMBRESIA) {
      printf("EXCEPCIÓN: No tienes membresía premium.\n");
      printf("Mostrando pantalla de pago...\n");
    }
  }

  return 0;
}
```

5.- Realizar un snippet de código que utilice los breakpoints en su IDE (En el lenguaje de su elección).

```
#include <stdio.h>

int sumar(int a, int b) {
   int resultado = a + b;  // <-- Coloca un breakpoint aquí
   return resultado;
}

int main() {
   int x = 5;              // <-- Breakpoint opcional
   int y = 7;              // <-- Breakpoint opcional

   int total = sumar(x, y); // <-- Breakpoint aquí para entrar a la función

   printf("El resultado es: %d\n", total);
   return 0;               // <-- Breakpoint final
}
```