

# Laboratorio de Programación - Labo04

## Debugging en C++

Algoritmos y Estructuras de Datos I

Departamento de Computación, FCEyN, Universidad de  
Buenos Aires.

# Motivación... ¿Cuál es el error?

## Ejercicio

Dado un string *s*, devolver el *íésimo* substring que se encuentre entre dos delimitadores.

---

```
1 indice_substring("esto#es#un#ejemplo", '#', 2)
```

---

devuelve el string "un".

---

```
1 indice_substring("#esto#es#un#ejemplo", '#', 2)
```

---

devuelve el string "es".

---

```
1 indice_substring("#esto#es#un#ejemplo", '#', 0)
```

---

devuelve el string vacío.

---

```
1 indice_substring("#esto##es#un#ejemplo", '#', 2)
```

---

devuelve el string vacío.

# Motivación... ¿Cuál es el error?

---

```
1  string indice_substring(string s, char delim, int ind) {
2      int i = 0; int j = 0; int actual = 0;
3      bool encuentre = false; string res = "";
4      while (i < s.size() && !encuentre) {
5          if (s[i] != delim) {
6              j = i;
7              while (j < s.size() && s[j] != delim) {
8                  res.push_back(s[j]);
9                  j++;
10             }
11         }
12         if (actual == ind) {
13             encuentre = true;
14         } else {
15             res.clear();
16             i = j + 1;
17             actual++;
18         }
19     }
20     if (encuentre || ind == actual)
21         return res;
22     else
23         return "Indice fuera de rango";
24 }
```

---

# Debugging

¿Qué es un bug?

# Debugging

¿Qué es un bug?

Error de software que produce un resultado o un comportamiento indeseado.

¿Cómo encontramos un bug?

# Debugging

## ¿Qué es un bug?

Error de software que produce un resultado o un comportamiento indeseado.

## ¿Cómo encontramos un bug?

Tests! (más en próximas clases).

# Debugging

¿Qué es un bug?

Error de software que produce un resultado o un comportamiento indeseado.

¿Cómo encontramos un bug?

Tests! (más en próximas clases).

¿Qué es debuggear (en español, depurar)?

# Debugging

## ¿Qué es un bug?

Error de software que produce un resultado o un comportamiento indeseado.

## ¿Cómo encontramos un bug?

Tests! (más en próximas clases).

## ¿Qué es debuggear (en español, depurar)?

Realizar un seguimiento línea por línea de nuestro programa usando un debugger.



# Debugging

## ¿Qué es un bug?

Error de software que produce un resultado o un comportamiento indeseado.

## ¿Cómo encontramos un bug?

Tests! (más en próximas clases).

## ¿Qué es debuggear (en español, depurar)?

Realizar un seguimiento línea por línea de nuestro programa usando un debugger.

## ¿Y qué es un debugger?

# Debugging

## ¿Qué es un bug?

Error de software que produce un resultado o un comportamiento indeseado.

## ¿Cómo encontramos un bug?

Tests! (más en próximas clases).

## ¿Qué es debuggear (en español, depurar)?

Realizar un seguimiento línea por línea de nuestro programa usando un debugger.

## ¿Y qué es un debugger?

Es un programa que toma como entrada otro programa (un binario) y nos permite controlar el flujo de ejecución. En CLion (y en todas las IDEs) es una herramienta que ya viene incorporada.

# ¿Hace falta?

## Alternativa 1: Imprimir por pantalla los resultados parciales.

A favor:

- ▶ Nos da una idea del recorrido del programa.

En contra:

- ▶ Proceso que consume mucho tiempo: agregar el código necesario, recompilar todo, correr el programa y analizar la salida. Todo eso cada vez que encontramos un bug.
- ▶ Ensuciamos el código.
- ▶ Al final hay que borrar todo lo que agregamos.

# ¿Hace falta?

## Alternativa 1: Imprimir por pantalla los resultados parciales.

A favor:

- ▶ Nos da una idea del recorrido del programa.

En contra:

- ▶ Proceso que consume mucho tiempo: agregar el código necesario, recompilar todo, correr el programa y analizar la salida. Todo eso cada vez que encontramos un bug.
- ▶ Ensuciamos el código.
- ▶ Al final hay que borrar todo lo que agregamos.

## Alternativa 2: Tratar de encontrar el error mirando el código.

A favor:

- ▶ Más rápido.

En contra:

- ▶ Para algoritmos no triviales, es difícil darse cuenta.

# Cómo lo usamos

1. Compilamos en modo debug. Esto introduce en nuestro programa símbolos especiales para ser usados por el debugger.
2. Corremos el programa con el debugger.

Conceptos fundamentales:

- ▶ *Breakpoint*: Suspender la ejecución del programa en una línea en particular.
- ▶ *Step Over*: Ir hacia la siguiente línea.
- ▶ *Step into*: Meterse dentro de una función.
- ▶ *Step out*: Salir de una función.
- ▶ *Watchpoint*: Hacer el seguimiento de una variable durante el transcurso de una función/programa.
- ▶ *Stacktrace/frames*: Ver en orden todas las funciones que fueron invocadas hasta el momento.

# IDE online

The image shows a screenshot of an online IDE with GDB debugging tools. The main editor displays a C program named `main.c` with the following code:

```
1 // Code, Compile, Run and Debug online from browsers to yours.
2
3 *****
4 #include <stdio.h>
5
6
7
8
9
10 int multiply(int n1, int n2) {
11     int result;
12     result = n1 * n2;
13     return result;
14 }
15
16 int factorial(int n) {
17     int fact = 1;
18     int i;
19
20     for(i=1; i<=n; i++)
21         fact = multiply(fact, i);
22
23     return fact;
24 }
25
26 int main() {
27     printf("Factorial of 5 is %d\n", factorial(5));
28
29     return 0;
30 }
31
32
```

Annotations point to various features:

- Call stack display function call chain:** Points to the Call Stack panel on the right, which shows the following data:

	Function	File:Line
0	factorial	main.c:21
1	main	main.c:28

- Click to set breakpoint:** Points to the red dot on line 12 of the `multiply` function.
- Displays values of local variables of function:** Points to the Local Variables panel on the right, which shows the following data:

Variable	Value
fact	1
i	2

- Displays any expression entered:** Points to the Display Expressions panel on the right, which shows the following data:

Expression	Value
Enter expression to watch	

- List of all breakpoints:** Points to the Breakpoints and Watchpoints panel on the right, which shows the following data:

	#	Description	
<input checked="" type="checkbox"/>	1	in multiply at main.c:12	X
<input checked="" type="checkbox"/>	2	in factorial at main.c:21	X

- controls execution of program:** Points to the Debug Console at the bottom, which shows the following data:

Continuing.

Breakpoint 2, factorial (n=5) at main.c:21  
fact = multiply(fact, i);  
(gdb) █

- Enter any command of gdb via this console:** Points to the input field in the Debug Console.

Esta imagen proviene de <https://www.onlinegdb.com/blog/brief-guide-on-how-to-use-onlinegdb-debugger/>. Podemos ver algunas de las herramientas de debugging que contiene esta IDE.

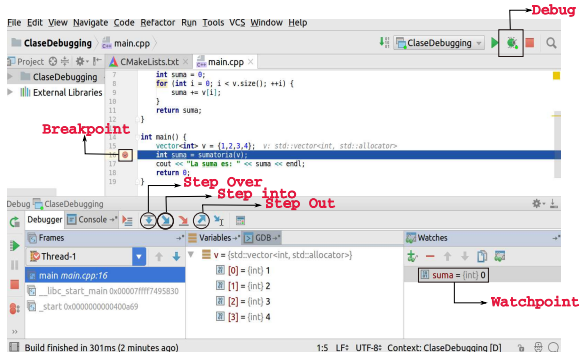
## Comandos GDB

Comando	Description
r	Ejecutar el programa hasta un breakpoint o el final
b fun	Fijar un breakpoint al inicio de la función "fun"
b N	Fijar un breakpoint en la línea N
b file.c:N	Fijar un breakpoint en la línea N del archivo "file.c"
d N	Borrar breakpoint numero N
info break	Listar breakpoints
c	Continuar la ejecución hasta un breakpoint o el final
f	Ejecutar hasta finalizar la función actual
s	Ejecutar la próxima línea
s N	Ejecutar próximas N líneas
n	Como s pero no entra en las funciones
p var	Imprime valor actual de variable "var"
set var=val	Asigna valor "val" a la variable "var"
bt	Imprime stack trace
q	Exit gdb

Esta tabla proviene de

<http://www.gdbtutorial.com/tutorial/commands>, donde además se pueden encontrar ejemplos simples de uso de GDB.

# En CLion



La figura muestra algunas herramientas de CLion para el debugging. Existen muchas otras para evaluación de expresiones durante el funcionamiento, breakpoints condicionales (se puede poner una guarda para detener el programa, que es muy util cuando se trabaja con iteraciones largas), y otras herramientas que pueden consultar en la guía online de CLion <https://blog.jetbrains.com/clion/2015/05/debug-clion/>.