Algoritmos y Estructura de Datos I
Taller de Debugging

Departamento de Computación, FCEyN, Universidad de Buenos Aires.

Debugging

¿Qué es un bug?

Debugging

¿Qué es un bug?

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

Debugging

¿Qué es un bug?

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

¿Qué es debuggear (en español, depurar)?

Debugging

¿Qué es un bug?

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

¿Qué es debuggear (en español, depurar)?

En sentido estricto, un proceso que involucra múltiples etapas. En el sentido usual, 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.

¿Qué es debuggear (en español, depurar)?

En sentido estricto, un proceso que involucra múltiples etapas. En el sentido usual, 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.

¿Qué es debuggear (en español, depurar)?

En sentido estricto, un proceso que involucra múltiples etapas. En el sentido usual, 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: Tratar de encontrar el error mirando el código.

A favor:

- Más rápido.

En contra:

- Casi nunca funciona :(

¿Hace falta?

Alternativa 1: Tratar de encontrar el error mirando el código.

A favor:

- Más rápido.

En contra:

- Casi nunca funciona :(

Alternativa 2: 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.

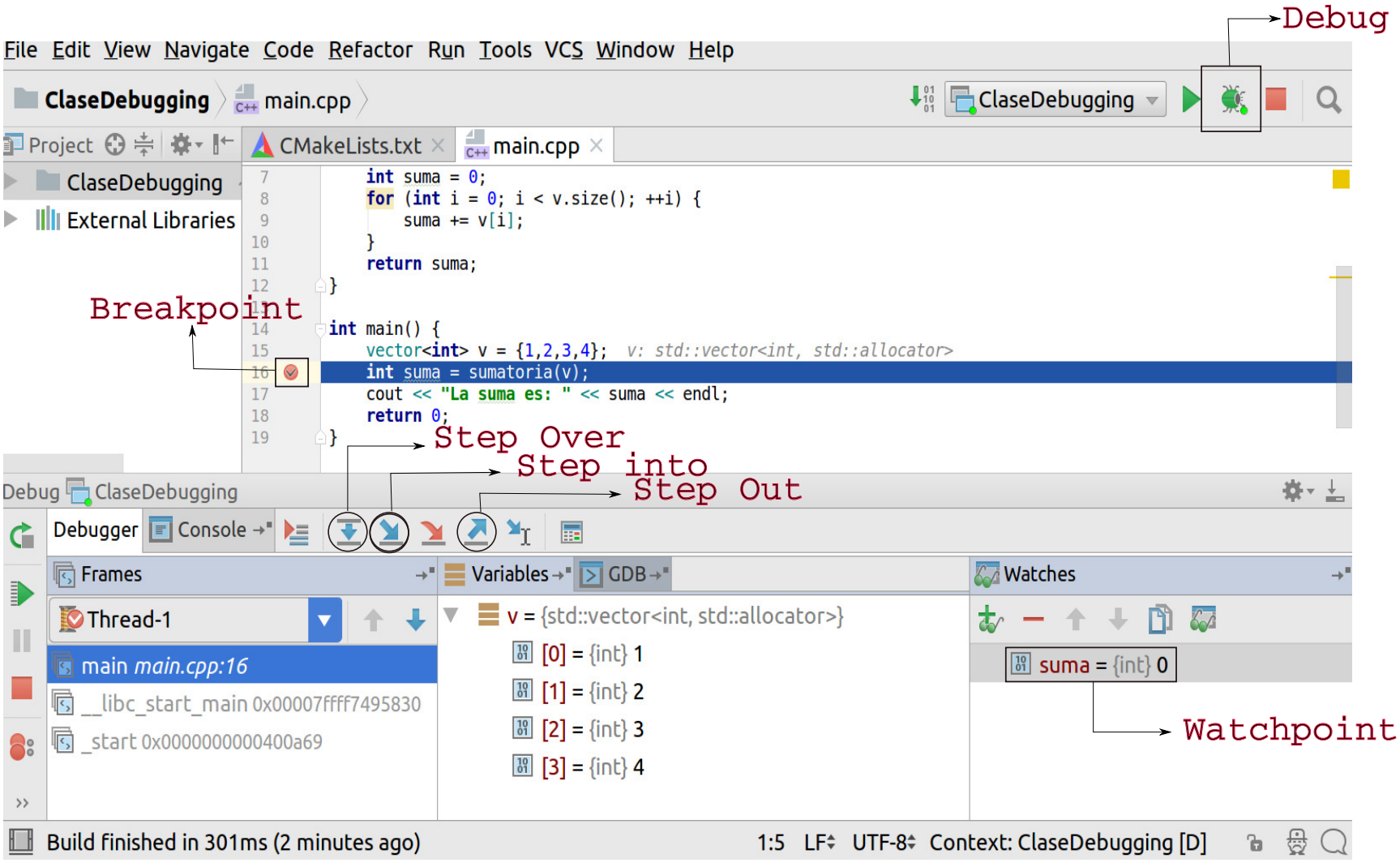
Cómo lo usamos

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.

En CLion


Debug
File Edit View Navigate Code Refactor Run Tools VCS Window Help
ClaseDebugging
main.cpp
ClaseDebugging
Project
CMakeLists.txt
main.cpp
ClaseDebugging
External Libraries
int suma = 0;
for (int i = 0; i < v.size(); ++i) {
suma += v[i];
}
return suma;
}
int main() {
vector<int> v = {1,2,3,4}; v: std::vector<int, std::allocator>
int suma = sumatoria(v);
cout << "La suma es: " << suma << endl;
return 0;
}
Breakpoint
Step Over
Step into
Step Out
Debug ClaseDebugging
Debugger
Console
Frames
Variables
GDB
Watches
Thread-1
main main.cpp:16
__libc_start_main 0x00007ffff7495830
_start 0x0000000000400a69
v = {std::vector<int, std::allocator>}
[0] = {int} 1
[1] = {int} 2
[2] = {int} 3
[3] = {int} 4
suma = {int} 0
Watchpoint
Build finished in 301ms (2 minutes ago)
1:5 LF UTF-8 Context: ClaseDebugging [D]

¿Cuál es el error?

Ejercicio

Dado un string s y un delimitador, devolver el iésimo substring.

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

devuelve el string "un".

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

devuelve el string "es".

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

devuelve el string vacío.

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

devuelve el string vacío.

Asserts

- Es una función para indicarle al programa que aborte la ejecución cuando no se cumple una condición.

```
1 #include <assert.h>
2 int primero(vector<int> v){
3   assert(v.size() > 0);
4   return v[0];
5 }
```