

Debugging

2022.04.25

SWPP Practice Session

Seunghyeon Nam

Typical Bugfix Process

- Notice an error
- **Narrow down** the line that causes the error
 - If the program crashes, look for the assertion or invalid pointer
 - If the program yields wrong output, look for the output variable
- **Traceback** to the line that started to go wrong

Narrowing Down the Line

- How do you locate the exact line that crashes?
 - **Guess** the location
 - Insert some `std::cout` or `std::cerr` all over the code
 - **Rebuild**
 - Look for the last printed message
 - **Repeat** until you actually pinpoint the location

Narrowing Down the Line

- This is horribly inefficient!
 - Taking a wild guess in a large codebase purely depends on luck
 - Rebuilding a large codebase may take minutes or even hours
 - And you have to repeat it until you actually find the bug
- Locating a single bug **may already take hours or days**

Traceback

- Most of the errors cannot be fixed locally
 - It is likely that the code that 'causes' the error is not a bug
 - The code that 'leads to' the error is the real verdict
 - But these two are usually far away from each other...
- You have to locate the code that **first** went wrong
 - Narrow down, take a step back, narrow down, again and again

Debugger to the Rescue

- Debugger can control the execution of your program
 - Line by line
 - In and out of function
 - Pause on assertion, throw, catch, breakpoint

Debugger to the Rescue

- Debugger can expose the execution context of your program
 - Call stack
 - Local/global variables and values

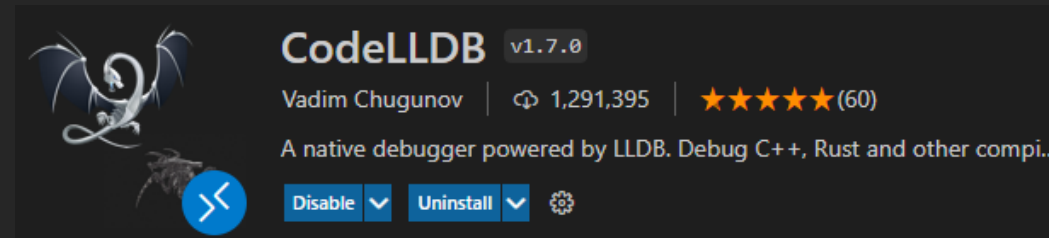
Using the Debugger

- LLDB: LLVM debugger
 - You have to enable LLDB project when building the LLVM
 - Already included if you built the LLVM using class repo script
- You must build your program **with clang**
- You must build your program **in debug mode**

Using the Debugger

- We'll use vscode extension for convenience

- CodeLLDB



- LLDB directory should be added to your PATH
 - What is PATH?

File Edit Selection View Go Run Terminal Help inst.cpp - swpp-compiler [S] /visual Studio Code

RUN AND DEBUG ...

RUN

Run and Debug

To customize Run and Debug create a launch.json file.

Show all automatic debug configurations.

To learn more about launch.json, see Configuring C/C++ debugging.

src > lib.cpp > {} 'anonymous-namespace' > Input C++ (GDB/LLDB)
C++ (Windows)
LLDB
Install an extension for C++...

Select LLDB from the options

```
19 std::string message;
20
21 public:
22 InputFileError(sc::parser::ParserError & err) {
23     using namespace std::string_literals;
24     message = "invalid input file\n"s.append(err.what());
25 }
26
27 InputFileError(fs::FilesystemError &&_err) noexcept {
28     using namespace std::string_literals;
29     message = "invalid input file\n"s.append(_err.what());
30 }
31
32 const char *what() const noexcept { return message.c_str(); }
33 };
34
35 You, last month | 1 author (You)
36 class OutputFileError : public Error<OutputFileError> {
37 private:
38     std::string message;
39 public:
40     OutputFileError(const std::string_view __message) {
41         using namespace std::string_literals;
42         message = "invalid output file\n"s;
43         message.append(__message);
44     }
45     const char *what() const noexcept { return message.c_str(); }
46 };
47
48 Result<std::string, InputFileError>
49 readFile(const std::string_view __filename) {
50     auto read_result = fs::readFile(__filename);
51     return decltype(read_result)::mapErr<InputFileError>(<
52         std::move(read_result),
53         [](auto &&err) { return InputFileError(std::move(err)); });
54 }
```

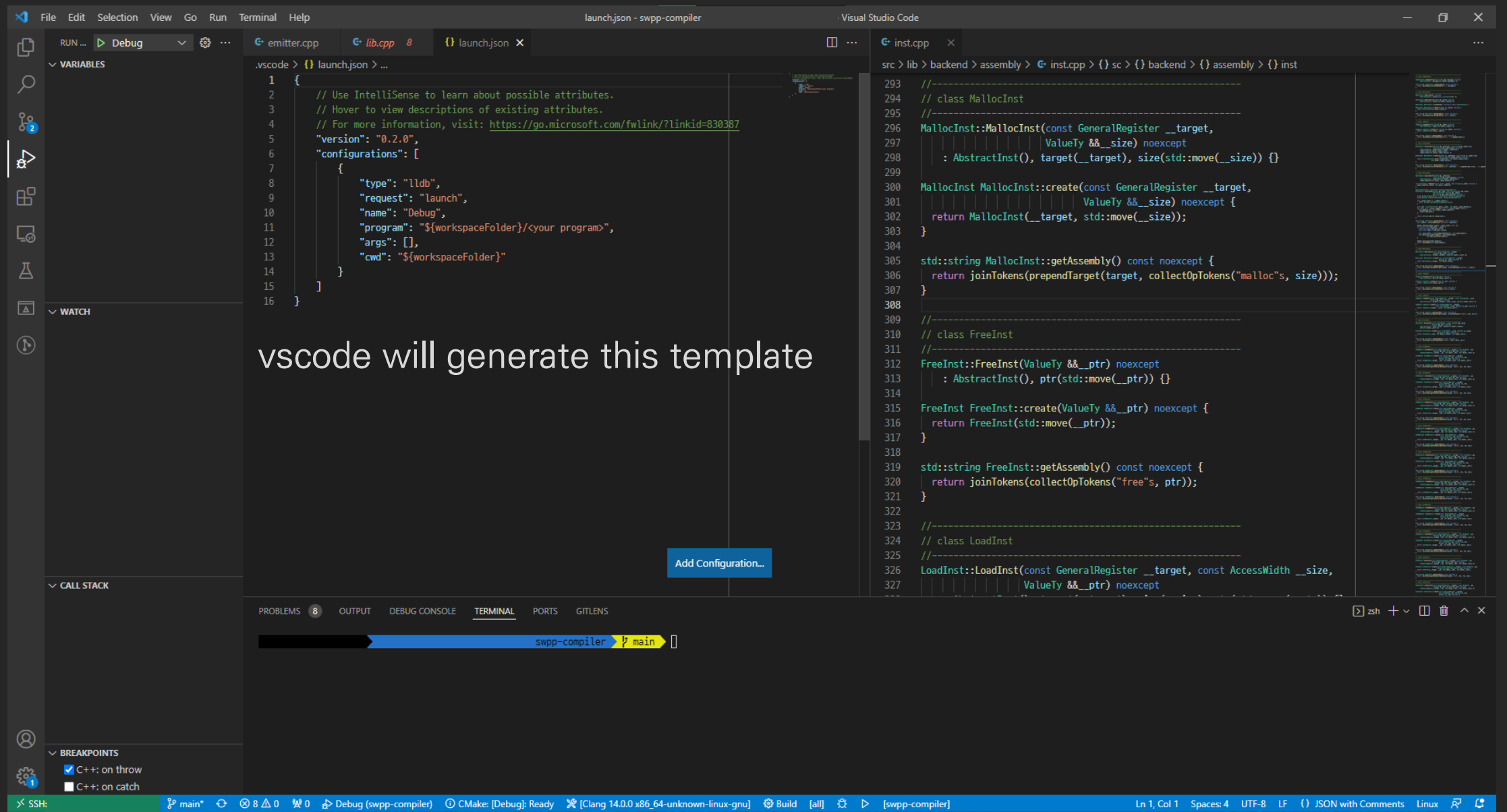
```
297
298 : AbstractInst(), target(__target), size(std::move(__size)) {}
299
300 MallocInst MallocInst::create(const GeneralRegister __target,
301                               ValueTy &&__size) noexcept {
302     return MallocInst(__target, std::move(__size));
303 }
304
305 std::string MallocInst::getAssembly() const noexcept {
306     return joinTokens(prependTarget(target, collectOpTokens("malloc"s, size)));
307 }
308
309 You, 2 months ago • Init ...
310 //-----
311 // class FreeInst
312 //-----
313 FreeInst::FreeInst(ValueTy &&__ptr) noexcept
314     : AbstractInst(), ptr(std::move(__ptr)) {}
315
316 FreeInst FreeInst::create(ValueTy &&__ptr) noexcept {
317     return FreeInst(std::move(__ptr));
318 }
319
320 std::string FreeInst::getAssembly() const noexcept {
321     return joinTokens(collectOpTokens("free"s, ptr));
322 }
323
324 //-----
325 // class LoadInst
326 //-----
327 LoadInst::LoadInst(const GeneralRegister __target, const AccessWidth __size,
328                    ValueTy &&__ptr) noexcept
329     : AbstractInst(), target(__target), size(__size), ptr(std::move(__ptr)) {}
```

PROBLEMS 8 OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

swpp-compiler main

SSH: main* 8 0 0 CMake: [Debug]: Ready [Clang 14.0.0 x86_64-unknown-linux-gnu] Build [all] [swpp-compiler] You, 2 months ago Ln 308, Col 1 Spaces: 2 UTF-8 LF C++ Linux





Visual Studio Code interface showing the configuration of a debugger for a C++ project named "swpp-compiler".

The **launch.json** file is open, showing the configuration for a **Debug** target. The configuration is as follows:

```
{
  // Use IntelliSense to learn about possible attributes.
  // Hover to view descriptions of existing attributes.
  // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
  "version": "0.2.0",
  "configurations": [
    {
      "type": "lldb",
      "request": "launch",
      "name": "Debug",
      "program": "${workspaceFolder}/build/swpp-compiler",
      "args": ["benchmarks/anagram/src/anagram.ll", "test.s"],
      "cwd": "${workspaceFolder}"
    }
  ]
}
```

The **inst.cpp** file is also open, showing the implementation of the **MallocInst** and **FreeInst** classes.

The **Terminal** panel at the bottom shows the output of the debugger, indicating that the program is running successfully.

Fill in the program and args.
Then run the debugger

Add Configuration...

Visual Studio Code interface showing a C++ project named "eh_throw.cc - swpp-compiler". The main editor displays the source code of "eh_throw.cc", which is paused at a breakpoint at line 77. The code defines a custom exception handler and a primary exception class. The left sidebar shows the "VARIABLES" pane with local variables like "obj", "tinfo", and "dest". The "WATCH" pane is empty. The "CALL STACK" pane shows the current function call stack. The bottom status bar indicates the current state of the debugger and the active file.

Debugger paused at throw

```
60 {
61     _cxa_refcounted_exception *header
62     = __get_refcounted_exception_header_from_obj (obj);
63     header->referenceCount = 0;
64     header->exc.exceptionType = tinfo;
65     header->exc.exceptionDestructor = dest;
66     header->exc.unexpectedHandler = std::get_unexpected ();
67     header->exc.terminateHandler = std::get_terminate ();
68     __GXX_INIT_PRIMARY_EXCEPTION_CLASS(header->exc.unwindHeader.exception_class);
69     header->exc.unwindHeader.exception_cleanup = __gxx_exception_cleanup;
70
71     return header;
72 }
73
74 extern "C" void
75 __cxxabiv1::__cxa_throw (void *obj, std::type_info *tinfo,
76                         void (_GLIBCXX_CDTOR_CALLABI *dest) (void *))
77 {
78     PROBE2 (throw, obj, tinfo);
79     __cxa_throw(obj, tinfo, dest);
80     globals->uncaughtExceptions += 1;
81     // Definitely a primary.
82     _cxa_refcounted_exception *header =
83     __cxa_init_primary_exception(obj, tinfo, dest);
84     header->referenceCount = 1;
85
86     #ifdef __USING_SJLJ_EXCEPTIONS__
87     _Unwind_SjLj_RaiseException (&header->exc.unwindHeader);
88     #else
89     _Unwind_RaiseException (&header->exc.unwindHeader);
90     #endif
91
92     // Some sort of unwinding error. Note that terminate is a handler.
93     __cxa_begin_catch (&header->exc.unwindHeader);
94 }
```

Console is in 'commands' mode, prefix expressions with '?'.
Launching: swpp-compiler/build/swpp-compiler benchmarks/anagram/src/anagram.ll test.s
Launched process 2701418

Visual Studio Code interface showing a C++ project named "eh_throw.cc - swpp-compiler". The editor displays the source code of "eh_throw.cc" and "inst.cpp". A yellow box highlights the "Run and Debug" icon in the top toolbar, and another yellow box highlights the "Execute until next breakpoint" icon (a blue play button with a right-pointing arrow) in the same toolbar.

The "VARIABLES" pane on the left shows the current state of variables:

- Local**
 - obj: 0x000000000242f90
 - tinfo: <invalid address>
 - dest: (libSCBackend.so`anony...
 - globals: <variable not availa...
 - header: <variable not availab...
- Static**
- Global**
- Registers**

The "WATCH" pane shows the current value of the selected variable:

- WATCH**
 - obj: 0x000000000242f90

The "CALL STACK" pane shows the current call stack:

- CALL STACK**
 - PAUSED ON BREAKPOINT
 - __cxxabiv1::__cxa_throw(void *,

The "BREAKPOINTS" pane shows the current breakpoints:

- BREAKPOINTS**
 - ☒ C++: on throw
 - ☐ C++: on catch

The "DEBUG CONSOLE" pane shows the output of the program:

```
Console is in 'commands' mode, prefix expressions with '?'.
Launching:
Launched process 2701418
```

The "TERMINAL" pane shows the output of the program:

```
src > lib > backend > assembly > inst.cpp > {} backend > {} assembly > {} inst
```


Visual Studio Code interface showing a C++ project named "eh_throw.cc - swpp-compiler". The editor displays the source code of "eh_throw.cc" and "inst.cpp". A yellow box highlights the "Execute next line" button (a circular arrow icon) in the top toolbar. Another yellow box highlights the same button in the bottom toolbar. The text "Execute next line" is overlaid on the editor area.

The left sidebar shows the "VARIABLES" panel with the following content:

```
VARIABLES
  Local
    > obj: 0x000000000242f90
    > tinfo: <invalid address>
    > dest: (libSCBackend.so` (anony-
    > globals: <variable not availa-
    > header: <variable not availab-
```

The "WATCH" panel shows the following content:

```
WATCH
  D 77 {
    PROBE2 (throw, obj, tinfo);
    _cxa_eh_globals *globals = _cxa_get_globals ();
    globals->uncaughtExceptions += 1;
    // Definitely a primary.
    _cxa_refcounted_exception *header =
    _cxa_init_primary_exception(obj, tinfo, dest);
    header->referenceCount = 1;
    #ifdef __USING_SJLJ_EXCEPTIONS__
    _Unwind_SjLj_RaiseException (&header->exc.unwindHeader);
    #else
    _Unwind_RaiseException (&header->exc.unwindHeader);
    #endif
    // Some sort of unwinding error. Note that terminate is a handler.
    _cxa_begin_catch (&header->exc.unwindHeader);
```

The "CALL STACK" panel shows the following content:

```
CALL STACK
  PAUSED ON BREAKPOINT
    _cxa_throw(void *,
    std::__cxx11::basic_string<char,
    sc::backend::emitter::AssemblyE
    llvm::InstVisitor<sc::backend::e
    void llvm::InstVisitor<sc::backe
    llvm::InstVisitor<sc::backend::e
    sc::backend::emitAssembly[abi:c
    (anonymous namespace)::compile(s
    result::Result<std::__cxx11::bas
```

The "BREAKPOINTS" panel shows the following content:

```
BREAKPOINTS
  [x] C++: on throw
  [ ] C++: on catch
```

The "MODULES" panel shows the following content:

```
MODULES
  >
```

The "DEBUG CONSOLE" panel shows the following content:

```
DEBUG CONSOLE
  Console is in 'commands' mode, prefix expressions with '?'.
  Launching:
  Launched process 2701418
```

The status bar at the bottom shows the following information:

```
SSH: main* 0 0 0 0 2 0 0 Debug (swpp-compiler) CMake: [Debug]: Ready [Clang 14.0.0 x86_64-unknown-linux-gnu] Build [all] [swpp-compiler] Format: auto Disasm: auto Deref: on Console: cmd Ln 77, Col 1 Spaces: 2 UTF-8 LF C++ Linux
```

Visual Studio Code interface showing a C++ project named "eh_throw.cc - swpp-compiler". The editor displays the source code of "eh_throw.cc" and "inst.cpp". A yellow box highlights the "Run and Debug" icon in the top toolbar. A large text overlay reads: "Step in (Get inside the function at current line & execute the first line of that function)". The "Run and Debug" icon is also highlighted with a yellow box in the top toolbar. The "CALL STACK" panel shows the current function being debugged: "std::__cxx11::basic_string<char, ...>". The "DEBUG CONSOLE" panel shows the output of the program, including the launch command and the process ID.

Visual Studio Code interface showing a C++ project named "eh_throw.cc - swpp-compiler". The editor displays the source code of "eh_throw.cc" and "inst.cpp". A yellow box highlights the "Run and Debug" icon in the top toolbar. A large text overlay reads: "Step in (Get inside the function at current line & execute the first line of that function)". The "Run and Debug" icon is also highlighted with a yellow box in the top toolbar. The "CALL STACK" panel shows the current function being debugged: "std::__cxx11::basic_string<char, ...>". The "DEBUG CONSOLE" panel shows the output of the program, including the launch command and the process ID.

Visual Studio Code interface showing a C++ project with a breakpoint set in `eh_throw.cc`. The editor displays the source code, and the left sidebar shows the **VARIABLES**, **WATCH**, and **CALL STACK** panels. The **DEBUG CONSOLE** at the bottom shows the output of the program.

Step out
(Execute until the end of current function & execute the next line of the caller)

The **DEBUG CONSOLE** output shows the following sequence of events:

```
Console is in 'commands' mode, prefix expressions with '?'.
Launching:
swpp-compiler/build/swpp-compiler benchmarks/anagram/src/anagram.ll test.s
Launched process 2701418
```



Narrowing Down with Debugger

- How do you locate the exact line that crashes?
 - Debugger pinpoints (automatically pauses on) the crash site
 - No need to insert new code, rebuild, etc.

Traceback with Debugger

- You have to locate the code that **first** went wrong
 - **Debugger shows you the call stack at the moment of the crash**
 - Clicking is all you need to navigate through the call stacks
 - Find the first call stack with unexpected value or control flow

Traceback with Debugger

- Sometimes you have to **monitor the change** of values
 - If your code does not throw or crash, debugger won't pause
 - You can use **breakpoint** to pause execution at a certain point

Visual Studio Code interface showing a C++ source file (`inst.cpp`) and the Breakpoints sidebar.

The source file (`inst.cpp`) contains the following code:

```
src > lib > backend > assembly > inst.cpp > {} 'anonymous-namespace' > joinTokens(std::vector<std::string> &&tokens) noexcept {
    Seunghyeon Nam, 2 days ago | 2 authors (You and others)
    1 #include "inst.h"
    2
    3 #include "../result.h"
    4
    5 #include <algorithm>
    6 #include <numeric>
    7
    8 using namespace std::string_literals;
    9
    10 namespace {
    11     std::string joinTokens(std::vector<std::string> &&tokens) noexcept {
    12         const auto collection_len = std::accumulate(
    13             __tokens.cbegin(), __tokens.cend(), static_cast<size_t>(0),
    14             [](const size_t len, const auto &line) -> size_t {
    15                 return len + line.size() + 1; // trailing whitespace or colon
    16             });
    17
    18         std::string joined_string;
    19         joined_string.reserve(collection_len);
    20         for (const auto &token : __tokens) {
    21             joined_string.append(token);
    22             joined_string.append(" ");
    23         }
    24         // remove trailing whitespace
    25         joined_string.pop_back();
    26         return joined_string;
    27     }
    28 }
    29
    30 using namespace sc::backend::assembly::inst;
    31
    32 std::string getToken(const IcmpCondition __cond) noexcept {
    33     switch (__cond) {
    34         case IcmpCondition::E0:
```

The Breakpoints sidebar (bottom left) shows the following configuration:

- ☒ C++: on throw
- ☐ C++: on catch
- ☒ `inst.cpp` `src/lib/backend/asse... (21)`

The status bar at the bottom indicates the current file is `inst.cpp` at line 21, column 33.

Visual Studio Code interface showing a C++ file named `inst.cpp` in the `src/lib/backend/assembly` directory. The code is paused on a breakpoint at line 21, which is highlighted in yellow. The breakpoint is located at the line `joined_string.append(D token);`.

The code in `inst.cpp` includes headers `inst.h` and `result.h`, and uses `std::string` and `std::vector`. It defines a function `joinTokens` that takes a vector of strings and returns a single string. The function uses `std::accumulate` to calculate the total length of the concatenated strings, including trailing whitespace or colons. It then reserves space in the `joined_string` and appends each token from the input vector.

The `CALL STACK` pane shows the current function call stack, with the top frame being `(anonymous namespace)::joinToker`. The `BREAKPOINTS` pane shows the active breakpoint at line 21 of `inst.cpp`.

The `WATCH` pane shows the current state of the `__tokens` variable, which is a vector of strings. The `DEBUG CONSOLE` pane shows the output of the program, which is currently empty.

The `TERMINAL` pane shows the command prompt, which is currently empty.

The `PROBLEMS` pane shows the current state of the program, which is currently empty.

The `OUTPUT` pane shows the output of the program, which is currently empty.

The `DEBUG CONSOLE` pane shows the output of the program, which is currently empty.

The `PORTS` pane shows the current state of the ports, which is currently empty.

The `GIT LENS` pane shows the current state of the git repository, which is currently empty.

The status bar at the bottom shows the current file, line, column, and other information.

IR Visualization

- Visualizing the control flow of your IR program can be helpful

```
define dso_local i32 @main() #0 {
entry:
  store i64* null, i64** @root, align 8
  %call = call i64 (...) @read()
  br label %for.cond

for.cond:                                ; preds = %for.inc, %entry
  %i.0 = phi i64 [ 0, %entry ], [ %inc, %for.inc ]
  %cmp = icmp ult i64 %i.0, %call
  br i1 %cmp, label %for.body, label %for.cond.cleanup

for.cond.cleanup:                        ; preds = %for.cond
  br label %for.end

for.body:                                ; preds = %for.cond
  %call1 = call i64 (...) @read()
  %call2 = call i64 (...) @read()
  %cmp3 = icmp eq i64 %call1, 0
  br i1 %cmp3, label %if.then, label %if.else

if.then:                                 ; preds = %for.body
  %call4 = call i64 @insert(i64 %call2)
  br label %if.end

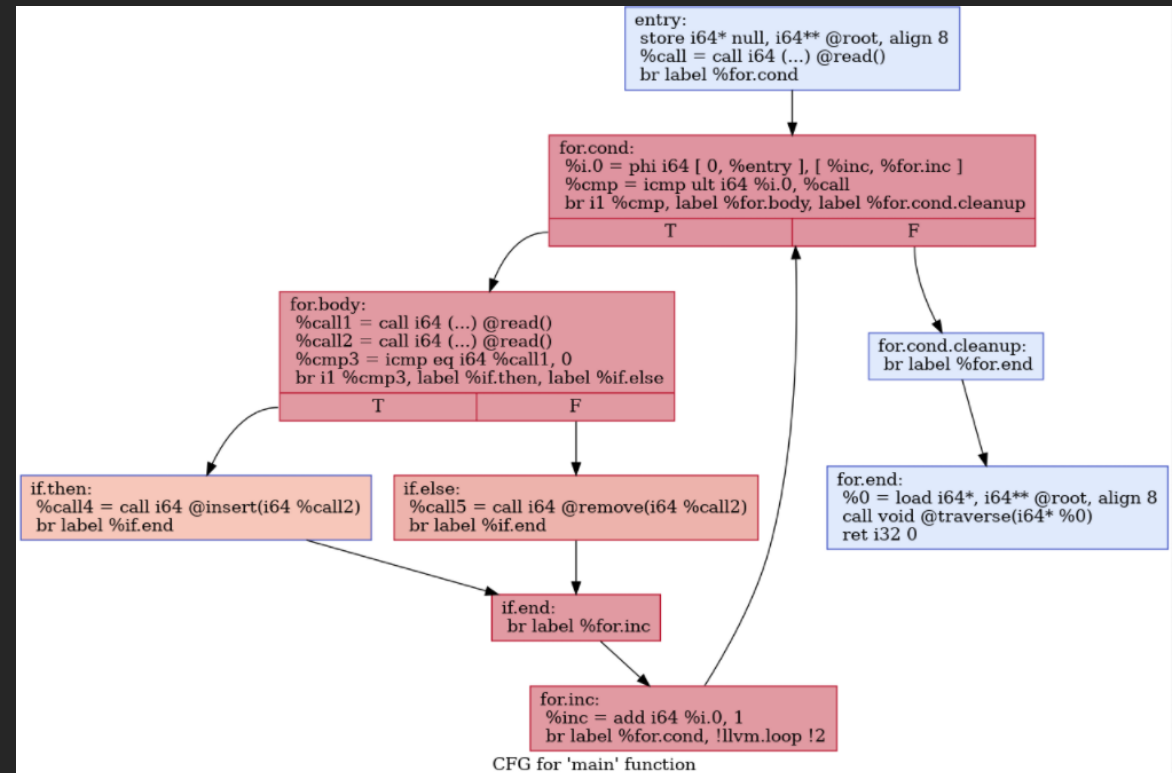
if.else:                                  ; preds = %for.body
  %call5 = call i64 @remove(i64 %call2)
  br label %if.end

if.end:                                   ; preds = %if.else, %if.then
  br label %for.inc

for.inc:                                  ; preds = %for.end, %if.end
  %inc = add i64 %i.0, 1
  br label %for.cond, !llvm.loop !2

for.cond.cleanup:
  br label %for.end

for.end:
  %0 = load i64*, i64** @root, align 8
  call void @traverse(i64* %0)
  ret i32 0
```



IR Visualization

- Install GraphViz
 - Use the package manager to handle dependencies for you
- Run `<llvm-dir>/opt --dot-cfg <IR-program.ll>`
 - You'll get a .dot file for each function in the program
- Run `dot <dot-file.dot> -Tpng -o <image-name.png>`