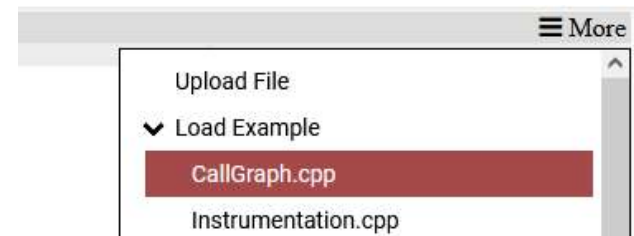


TUTORIAL



Tutorial - Preparation

- Open Clava temporary demo website
 - <http://specs.fe.up.pt/tools/clava/>
- Delete contents of top-right area
- To reload C/C++ example:
 - More->Load Example->CallGraph.cpp



Tutorial - Preparation

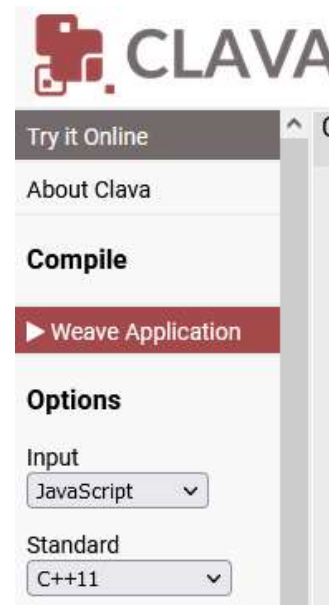
The screenshot displays the CLAVA online compiler interface. On the left is a sidebar with navigation links: 'Try it Online', 'About Clava', 'Compile', 'Weave Application', 'Options', 'Text Editor', and 'Resources'. The 'Options' section includes dropdowns for 'Input' (set to JavaScript) and 'Standard' (set to C++11), along with a 'Font size' of 12 and a note to 'Press F11 to toggle fullscreen'. The main area is divided into three panes. The top-left pane, titled 'C/C++ Source Code', contains the following code:

```
1- /*
2-  Simple matrix multiplication example.
3- */
4-
5- #include <stdio.h>
6- #include <math.h>
7- #include <stdlib.h>
8-
9- /*
10-  matrix multiplication
11- */
12- void matrix_mult(const double* A , const double* B, double*
13-
14-     for(int ii=0; ii<N; ii++) {
15-         for(int jj=0; jj<K; jj++) {
16-             //C[i][j] = 0;
17-             C[K*ii + jj] = 0;
18-         }
19-     }
```

The top-right pane is titled 'JavaScript' and contains a single line of code: '1'. The bottom-left pane, titled 'Result', shows the message: '1 Nothing yet, press "Weave Application".'. The bottom-right pane is titled 'Output Log' and is currently empty.

Tutorial – Hello World

- JavaScript box
 - `println("Hello World")`
- Press “Weave Application”



Tutorial – Node types (Ex1)

Use Query API to get and print the AST root node

Tutorial – Node types (Ex1)

Use Query API to get and print the AST root node

- Check API Documentation

| Resources |
|--|
| Download Clava |
| Language Specification |
| API Documentation |
| GitHub |

Tutorial – Node types (Ex1)

Use Query API to get and print the AST root node

- Check API Documentation
- Find “Query” class
- Find out how to get the root node

| Resources |
|--|
| Download Clava |
| Language Specification |
| API Documentation |
| GitHub |

Tutorial – Node types (Ex1)

Use Query API to get and print the AST root node

```
laraImport ("weaver.Query");  
  
println (Query.root ());
```

Output Log

'program'

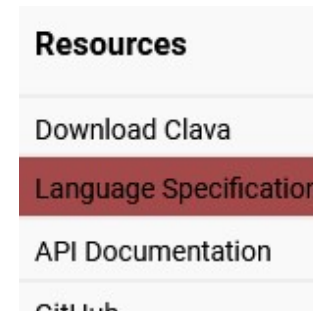
Tutorial – Node Attributes (Ex2)

Print code of root node

Tutorial – Node Attributes (Ex2)

Print code of root node

- Language Specification
 - Toggle Language Specification
 - ...or open Language Specification page
- Find attribute 'code'
- To access attribute code: `node.code`



Tutorial – Node Attributes (Ex2)

Print code of root node

```
laraImport ("weaver.Query");  
  
println (Query.root ().code);
```

Output Log

```
**** File 'weaved.cpp' ****/
```

```
#include <stdio.h>  
#include <math.h>  
#include <stdlib.h>
```

```
/*  
Simple matrix multiplication example.  
*/
```

```
/*  
matrix multiplication  
*/
```

```
void matrix_mult(double const *A, double c  
for(int ii = 0; ii < N; ii++) {
```

Tutorial – Node Attributes (Ex3)

Print filenames of the files of the program

Tutorial – Node Attributes (Ex3)

Print filenames of the files of the program

- Get files from an attribute in program
- Get filename from an attribute in file

Tutorial – Node Attributes (Ex3)

Print filenames of the files of the program

```
laraImport ("weaver.Query");  
  
println (Query.root().files  
    .map (file => file.filename))
```



Output Log
weaved.cpp

Tutorial – AST Structure (Ex4)

What are the types of the children of a 'function' node?

Tutorial – AST Structure (Ex4)

What are the types of the children of a ‘function’ node?

- Use the attribute ‘dump’ to print the AST

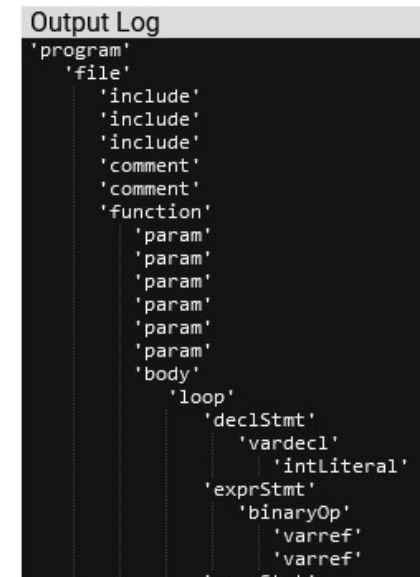
Tutorial – AST Structure (Ex4)

What are the types of the children of a ‘function’ node?

R.: param and body

```
laraImport ("weaver.Query") ;
```

```
println (Query.root () .dump) ;
```



```
Output Log
'program'
  'file'
    'include'
    'include'
    'include'
    'comment'
    'comment'
    'function'
      'param'
      'param'
      'param'
      'param'
      'param'
      'param'
      'body'
        'loop'
          'declStmt'
            'vardecl'
              'intLiteral'
          'exprStmt'
            'binaryOp'
              'varref'
              'varref'
```

Tutorial – AST Navigation (Ex5)

Print the name of all functions that start with the letter 'm'

Tutorial – AST Navigation (Ex5)

Print the name of all functions that start with the letter 'm'

- Check documentation of “Query” class
- Check the method “search()” of Query

Tutorial – AST Navigation (Ex5)

Print the name of all functions that start with the letter 'm'

R.: matrix_mult, main



Output Log
matrix_mult,main

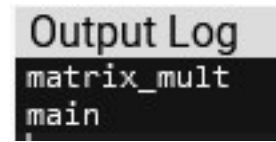
```
laraImport ("weaver.Query") ;
```

```
println (Query.search ("function", {name: /^m.*\/}))  
    .get ()  
    .map (f => f.name))
```

Tutorial – AST Navigation (Ex5)

Print the name of all functions that start with the letter ‘m’

R.: matrix_mult, main



```
Output Log
matrix_mult
main
```

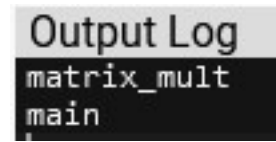
```
laraImport ("weaver.Query") ;
```

```
for(const f of Query.search("function", {name: /^m.*$/}))
{
    println(f.name)
}
```

Tutorial – AST Navigation (Ex5)

Print the name of all functions that start with the letter 'm'

R.: matrix_mult, main



```
Output Log
matrix_mult
main
```

```
laraImport ("weaver.Query") ;
```

```
for(const f of Query.search("function",
                             {name: n => n.startsWith("m")})) {
  println(f.name)
}
```

Tutorial – AST Navigation (Ex6)

Print all <function> - <call> pairs

Tutorial – AST Navigation (Ex6)

Print all <function> - <call> pairs

- E.g. `main->test_matrix_mul, test_matrix_mul->matrix_mult,...`
- `Query.search()` returns a `weaver.Selector`
- Chained searches
 - Method 1: `Selector.search() + Selector.chain()`
 - Method 2: Nested `for` + `Query.searchFrom()`

Tutorial – AST Navigation (Ex6)

Print all <function> - <call> pairs

Output Log

init_matrix->rand,print_matrix_result->printf,

```
laraImport("weaver.Query");

println(Query.search("function")
        .search("call")
        .chain()
        .map(ch => ch.function.name
                  + "->" + ch.call.name)
)
```

Tutorial – AST Navigation (Ex6)

Print all <function> - <call> pairs

```
laraImport("weaver.Query");
```

```
for(const f of Query.search("function")) {  
  for(const c of Query.searchFrom(f, "call")) {  
    println(f.name+"->" + c.name)  
  }  
}
```

Output Log

```
init_matrix->rand  
print_matrix_result->printf  
test_matrix_mul->malloc
```

Tutorial – AST Transformation (Ex7)

Insert a comment before each call, with format “// <call_name>”

Tutorial – AST Transformation (Ex7)

Insert a comment before each call, with format “// <call_name>”

- “Actions” section in Language Specification
- Actions change the AST
- Actions for inserting code
 - `insertBefore (node | String)`
 - `insertAfter (node | String)`

Tutorial – AST Transformation (Ex7)

Insert a comment before each call, with format “// <call_name>”

```
laraImport ("weaver.Query") ;
```

```
Query.search ("call").get ()
```

```
    .forEach (c => c.insertBefore ("// "+c.name))
```

```
int main() {  
    // To make results repeatable  
    // srand  
    srand(0);  
    // test_matrix_mul  
    test_matrix_mul();  
}
```

Tutorial – AST Transformation (Ex8)

Insert a printf before each call that prints “<call_name>@<line>”

Tutorial – AST Transformation (Ex8)

Insert a printf before each call that prints “<call_name>@<line>”

- JavaScript template literals/string might help
- Can use `Clava.rebuild()` to test generated code syntax
 - `laraImport("clava.Clava")`

Tutorial – AST Transformation (Ex8)

Insert a printf before each call that prints “<call_name>@<line>”

```
laraImport ("weaver.Query") ;
```

```
Query.search ("call")  
  .get ()  
  .forEach (c => c.insertBefore (  
    `printf ("${c.name}@${c.line}\\n") ;`  
  )  
)
```

```
int main() {  
  printf("srand@86\n");  
  // To make results repeatable  
  srand(0);  
  printf("test_matrix_mul@88\n");  
  test_matrix_mul();  
}
```


Tutorial – AST Transformation

- What about includes?

Tutorial – AST Transformation

- What about includes?
 - `call.ancestor("file").addInclude("stdio.h", true)`

Tutorial – AST Transformation

- What about includes?
 - `call.ancestor("file").addInclude("stdio.h", true)`
- What if C++?

Tutorial – AST Transformation

- What about includes?

- `call.ancestor("file").addInclude("stdio.h", true)`

- What if C++?

- `Clava.isCxx()`

Tutorial – AST Transformation

- What about includes?
 - `call.ancestor("file").addInclude("stdio.h", true)`
- What if C++?
 - `Clava.isCxx()`
- Seems a lot of work, is there a better way?

Tutorial – AST Transformation

- What about includes?
 - `call.ancestor("file").addInclude("stdio.h", true)`
- What if C++?
 - `Clava.isCxx()`
- Seems a lot of work, is there a better way?
 - Encapsulate complex functionality in an API

Tutorial – AST Transformation (Ex9)

***Insert a printf before each call that prints “<call_name>@<line>”,
using the Logger API (i.e. lara.code.Logger)***

Tutorial – AST Transformation (Ex9)

Insert a printf before each call that prints “<call_name>@<line>”, using the Logger API (i.e. lara.code.Logger)

- Instantiate a Logger object
- Methods `text()`, `ln()` and `log()` might be useful

Tutorial – AST Transformation (Ex9)

Insert a printf before each call that prints “<call_name>@<line>”, using the Logger API (i.e. lara.code.Logger)

```
laraImport ("weaver.Query");  
laraImport ("lara.code.Logger");  
const logger = new Logger();
```

```
int main() {  
    std::cout << "srand@86" << "\n";  
    // To make results repeatable  
    srand(0);  
    std::cout << "test_matrix_mul@88" << "\n";  
    test_matrix_mul();  
}
```

```
Query.search("call").get().forEach(c =>  
    logger.text(`${c.name}@${c.line}`)  
    .ln()  
    .log(c, true))
```

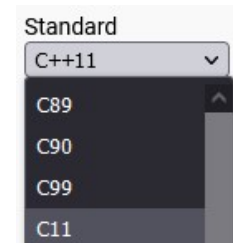
Tutorial – AST Transformation (Ex9)

Insert a printf before each call that prints “<call_name>@<line>”, using the Logger API (i.e. lara.code.Logger)

```
laraImport ("weaver.Query");  
laraImport ("lara.code.Logger");  
const logger = new Logger();
```

```
Query.search("call").get().forEach(c =>  
    logger.text(`${c.name}@${c.line}`)  
    .ln()  
    .log(c, true))
```

```
int main() {  
    printf("srand@86\n");  
    // To make results repeatable  
    srand(0);  
    printf("test_matrix_mul@88\n");  
    test_matrix_mul();  
}
```



Tutorial – AST Transformation (Ex10)

Use Timer API to measure execution time of `matrix_mult` calls

Tutorial – AST Transformation (Ex10)

Use Timer API to measure execution time of `matrix_mult` calls

- `Import lara.code.Timer`

Tutorial – AST Transformation (Ex10)

Use Timer API to measure execution time of `matrix_mult` calls

```
laraImport ("weaver.Query");  
laraImport ("lara.code.Timer");  
const timer = new Timer();
```

```
std::chrono::high_resolution_clock::  
// do: C = A*B  
matrix_mult(A, B, C, N, M, K);  
std::chrono::high_resolution_clock::  
auto clava_timing_duration_0 = std:::  
std::cout << "Time matrix_mult@77:"
```

```
Query.search("call", "matrix_mult")  
    .get()  
    .forEach(c =>  
        timer.time(c, `Time ${c.name}@${c.line}:`))
```