

# **Code::Blocks: Open Source IDE for Fortran**

**Darius Markauskas**

# Code::Blocks IDE

- Open source project: <http://codeblocks.org>
- First commits to SVN by “mandrav” in 2004
- Development using: C++, wxWidgets, Scintilla
- Code organized into a core and plugins
- Runs on Linux, Windows, Mac (?)
- Oriented towards C++ and Fortran

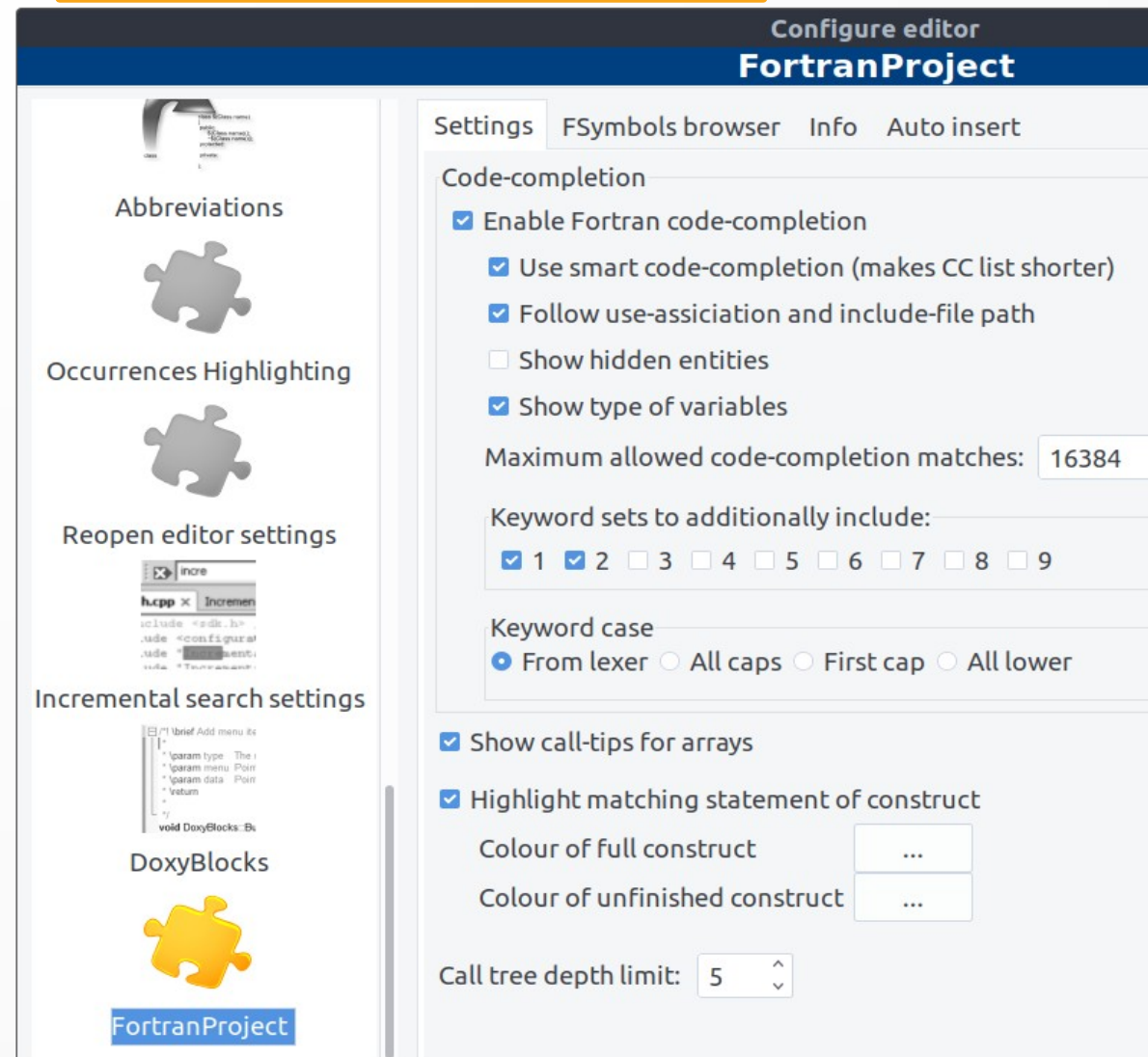


# FortranProject plugin

- Makes C::B useful for Fortran
- There are other parts in IDE too, where Fortran specific code is included: Fortran compilers, SmartIndentFortran plugin...
- Was started in 2010 by “darmar” (me)
- About 29k of code lines (C::B >400k)
- Custom build for Linux and Windows and more useful information for Fortran users on:

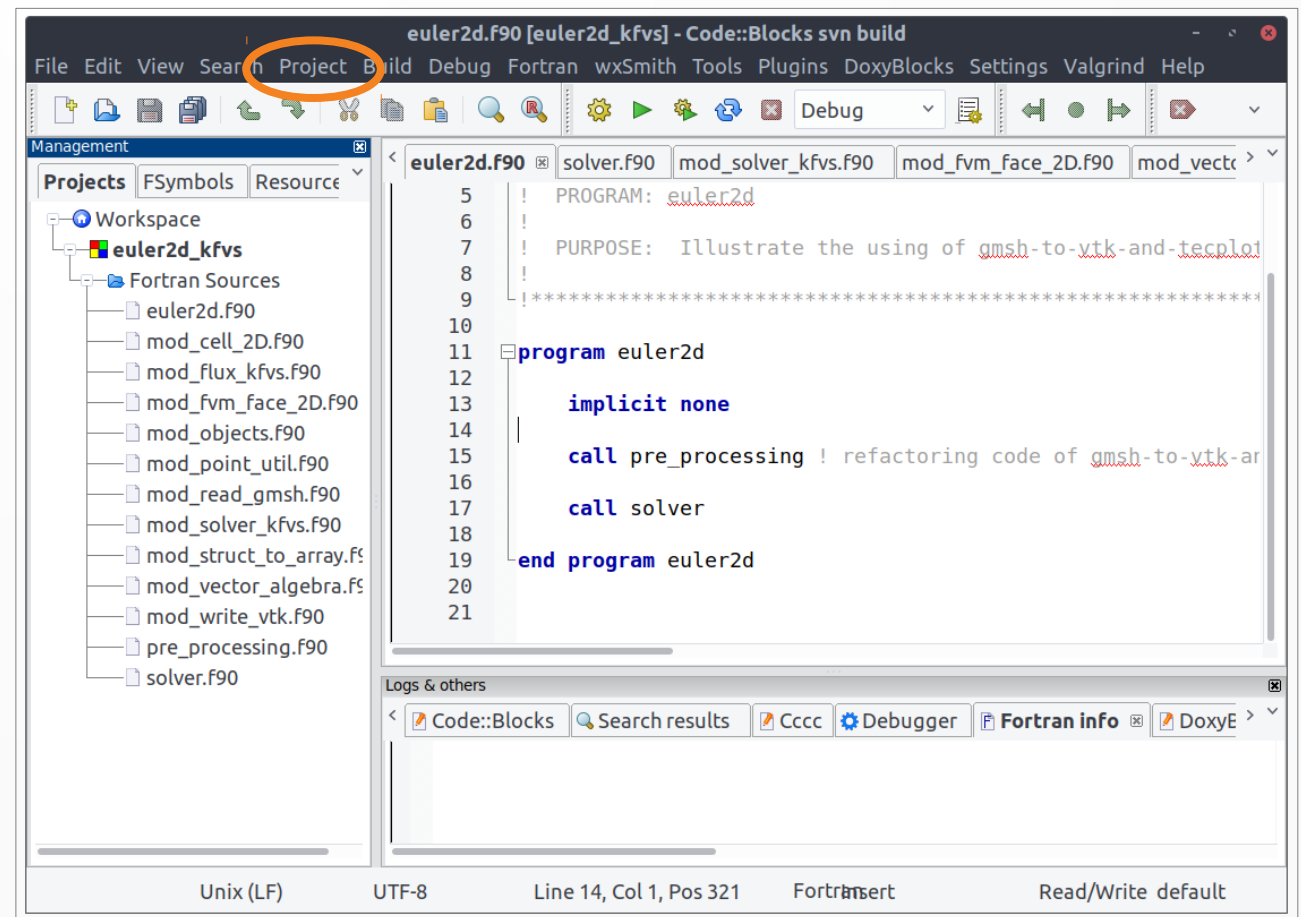
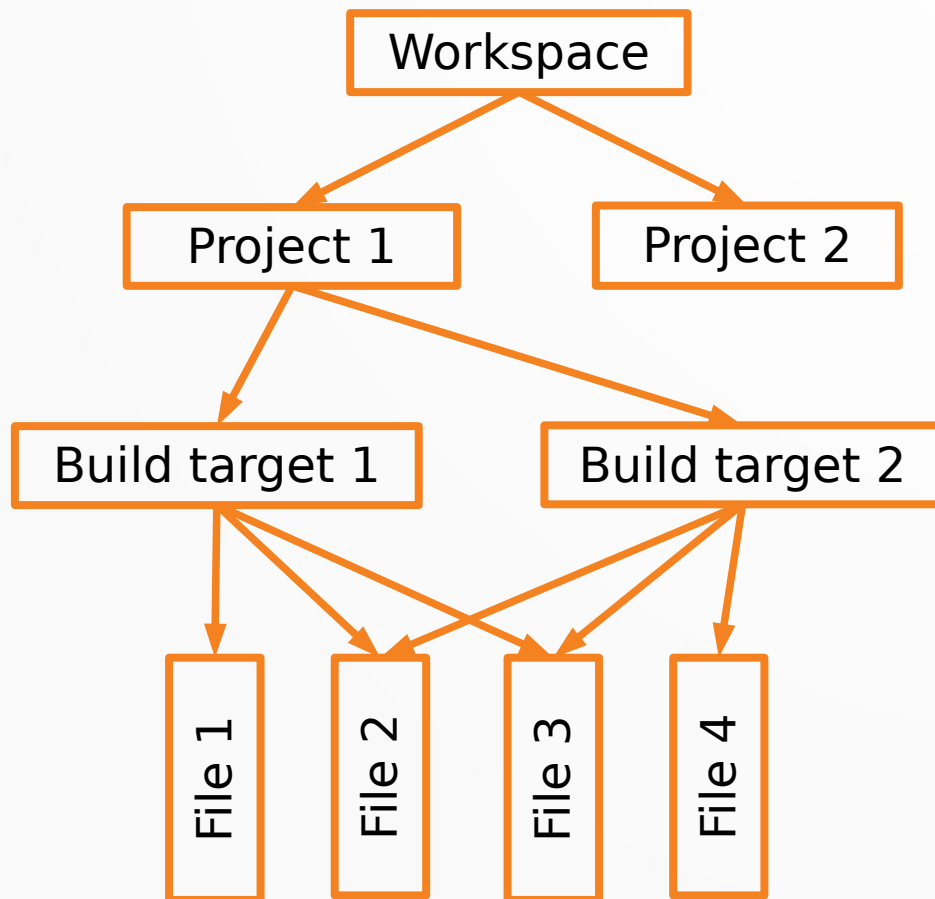
<http://cbfortran.sourceforge.net>

## Settings->Editor...



# Code organization using the IDE

- User files are grouped into workspaces, projects and targets





# Editor

- Syntax highlighting (for free and fixed source code forms)
- Code folding
- Occurrences highlighting: highlights selected word in editor
- Fortran construct highlighter
- Keyboard shortcuts adjustment (Settings→Editor→Keyboard shortcuts)

```
1
2  PROGRAM BSP
3
4  INTEGER MAXIND
5  PARAMETER (MAXIND=10)
6  REAL ARR(MAXIND)
7  C ACHTUNG! Array startet mit dem Index
8  C ARR(0) waere ein Fehler!
9  ARR(1) = 1.5
10 ARR(2) = 2.5
11 ARR(10) = 10.5
12
13 WRITE (*,*) ARR(1)
```

```
m = 0
do j = 1, size(tree)
  n = tree(j)
  if (n > m) then
    call this%start (trim(name(n)))
    m = n
  else
    timer => this%current ! save point
    call this%stop (trim(name(n)))
    timer%time = time(n) ! overwrite
  end if
end do
```

```
343
346  if (n > m) then
347  else
348    timer => this%current ! save point
349    call this%stop (trim(name(n)))
350    timer%time = time(n) ! overwrite
351  end if
352
353
```

# Code completion

- Is shown when you type or on “Ctrl+Space”
- Code completion for:
  - Keywords
  - Defined variables, procedures
  - Derived type components
  - Type-bound procedures
- Follows use-association
- Uses logic to make CC list shorter (smart code-completion)

Settings→Editor



General settings

## Code Completion

☒ Code completion

☐ Case sensitive

☐ Autoselect single match

Autolaunch after typing # letters: 3

☒ Documentation popup

Tooltips: enable

```
21  
22  call  
23  
24  
25  
26  
27  
28  
29
```

# Call-tips, tool-tips

- Call-tips: show information about dummy arguments (Shift+Ctrl+Space)
- Tool-tips: are shown when mouse is kept over item

```
call list%addfirst()
```

```
(val_data, size_data, success)  
integer :: val_data  
! data to add
```

```
388 time = timer%time  
389 if (running(this, timer)) then ! add the time since it was s  
390     logical function running(this, timer) count_max)  
391     !! Returns true if the target  
392     of the timer pointer is running. L(count_rate)  
393     el timer_tree_type,  
394         timer_tree_type.F90:467 -count-1))/real(cour  
395     end if  
396 end if  
397 end subroutine timer_tree_read  
398
```


s & others

Code::Blocks Search results Cccc Debugger Fortran info DoxyBlocks


```
!! Returns true if the target of the timer pointer is running.  
logical function running(this, timer)  
class(timer_tree), intent(in) :: this  
type(node), pointer :: timer  
!Module: timer_tree_type. File: timer_tree_type.F90:467
```

# Auto-complete, auto-insert

- Auto-complete: **Not** code-completion!
  - Replaces typed keyword with the predefined code
  - Invoked by typing one of the keywords and pressing “Ctrl+J” (Edit→Auto-complete)
  - Change, add new:  
Settings→Editor→Abbreviations
- Auto-insert:
  - Inserts “end...” after “do”, “if(...)then” etc.
  - Options: Settings→Editor→FortranProject, Auto insert tab



```
34  
35  
36  
37 sd  
38  
39  
40  
41  
42  
43  
44
```

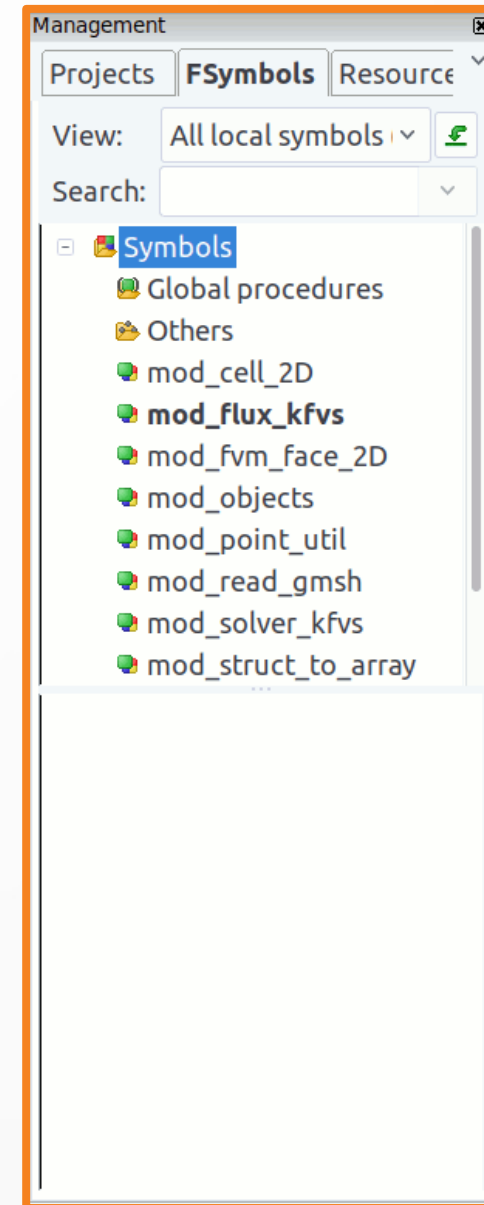


```
18  
19 |  
20  
21  
22
```



# Symbols Browser

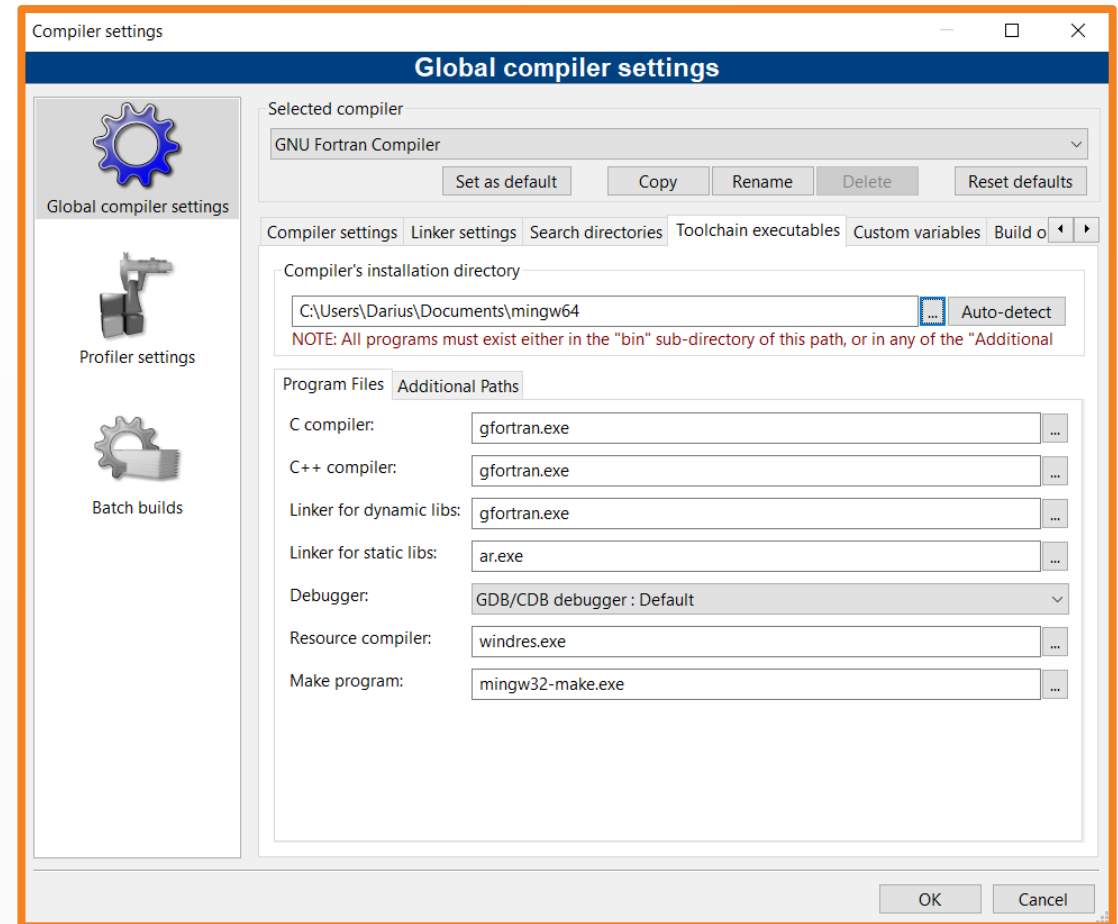
- Displays defined items in a workspace, an active project or a current file:
  - Global and module procedures
  - Modules
  - Submodules
  - Local variables
- Recognizes public/private items (■) (■)
- Item, where the cursor is, is marked in bold
- Double-click to go to the declaration



# Compilation

- Code compilation from within IDE
- IDE's build-in build system is used
- Takes care of dependencies between Fortran files
- If possible, several files are compiled at the same time
- Possibility to use external makefiles
- Compiler support: Gfortran, Intel, PGI, Oracle Fortran
- Additional compilers can be added by users

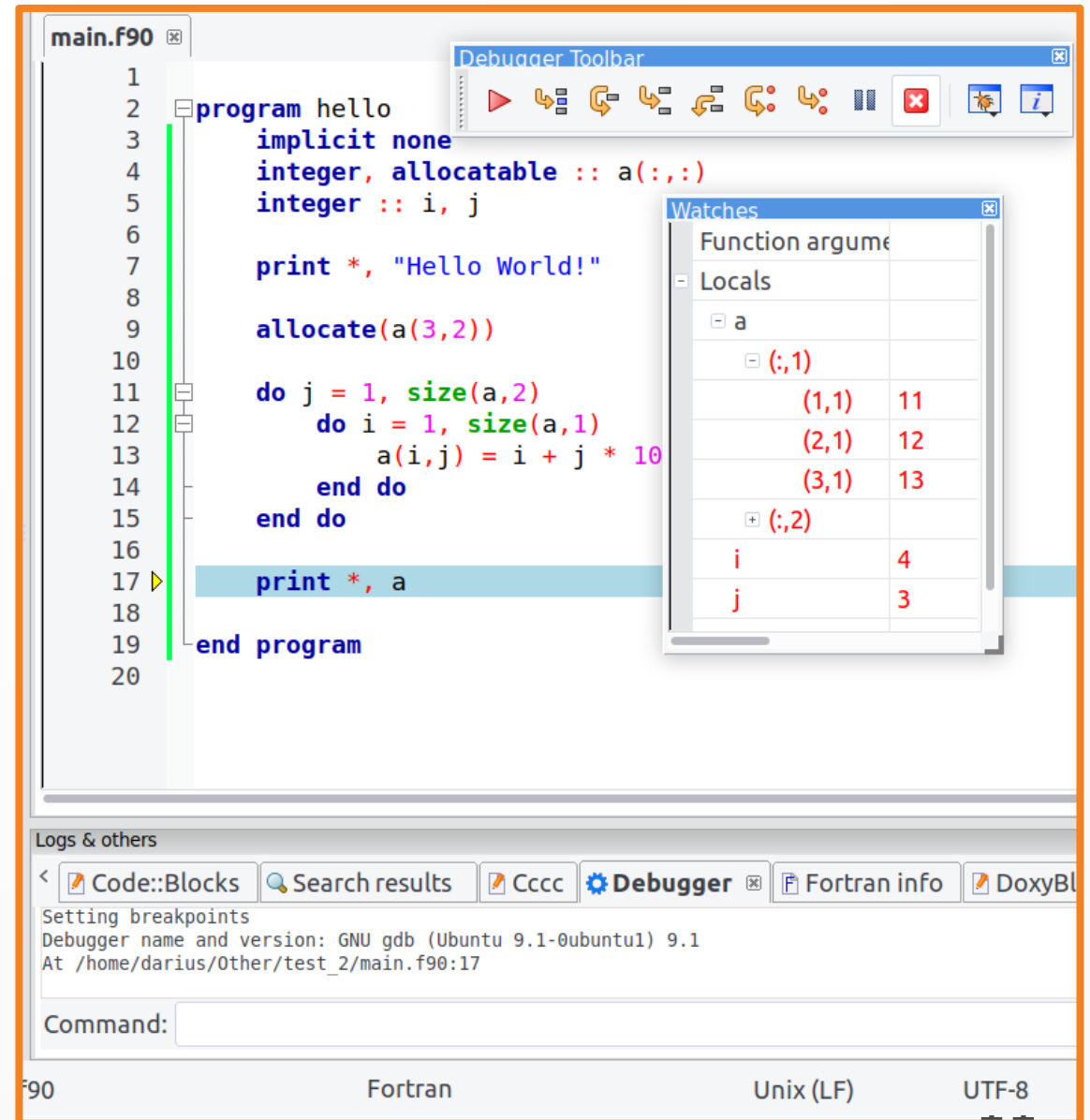
## Settings→Compiler



# Debugging

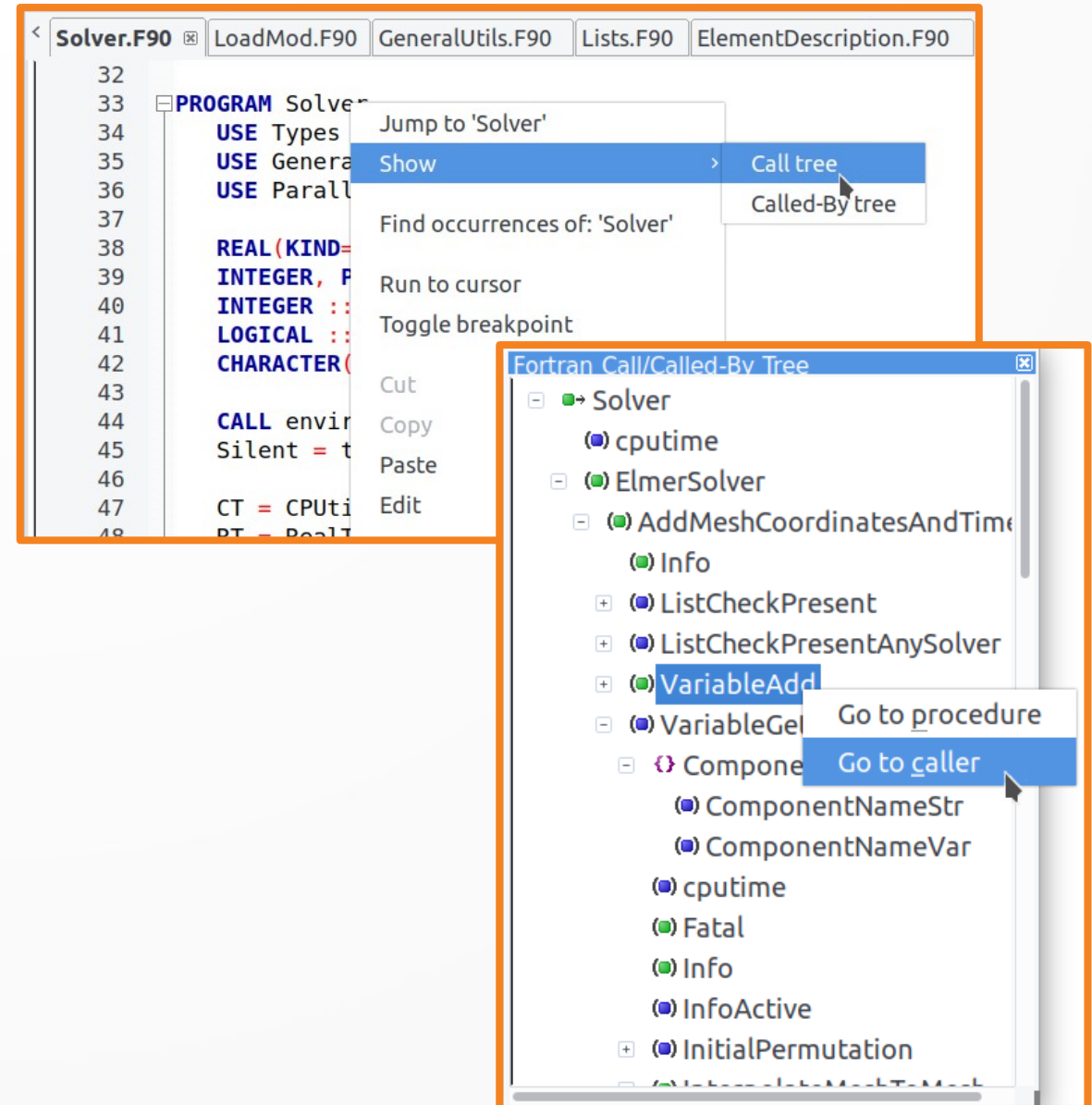
- Debugging with GDB debugger
- Watches window
- GDB command prompt
- CBFortran custom build:
  - Improvement through use of Python pretty printer
  - Possibility to visualize 1D and 2D arrays with Gnuplot
- More info:

<http://cbfortran.sourceforge.net/debugging>



# Call/Called-by tree

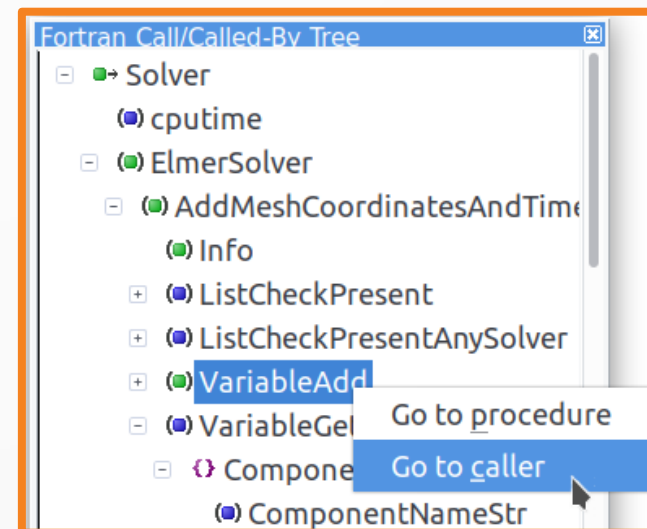
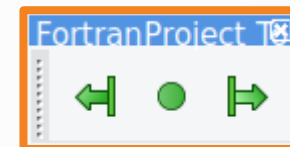
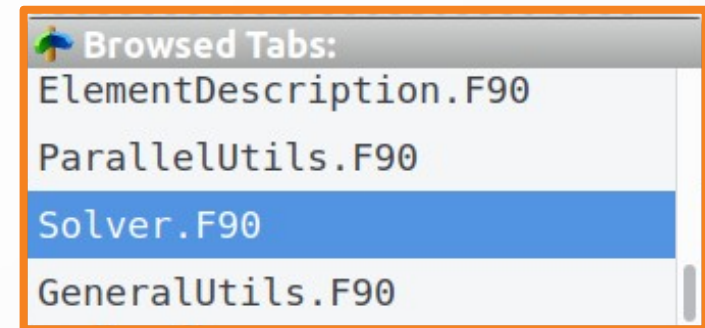
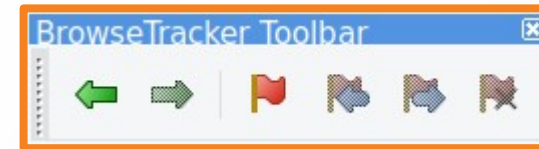
- Shows called/calling procedures in a tree
- Enables easy navigation in the code
- To show: right-click on a procedure or module name and choose “Show→Call tree/Called-By tree”
- If build takes too long, decrease “Call tree depth limit” on FortranProject setting dialog





# Navigation in code

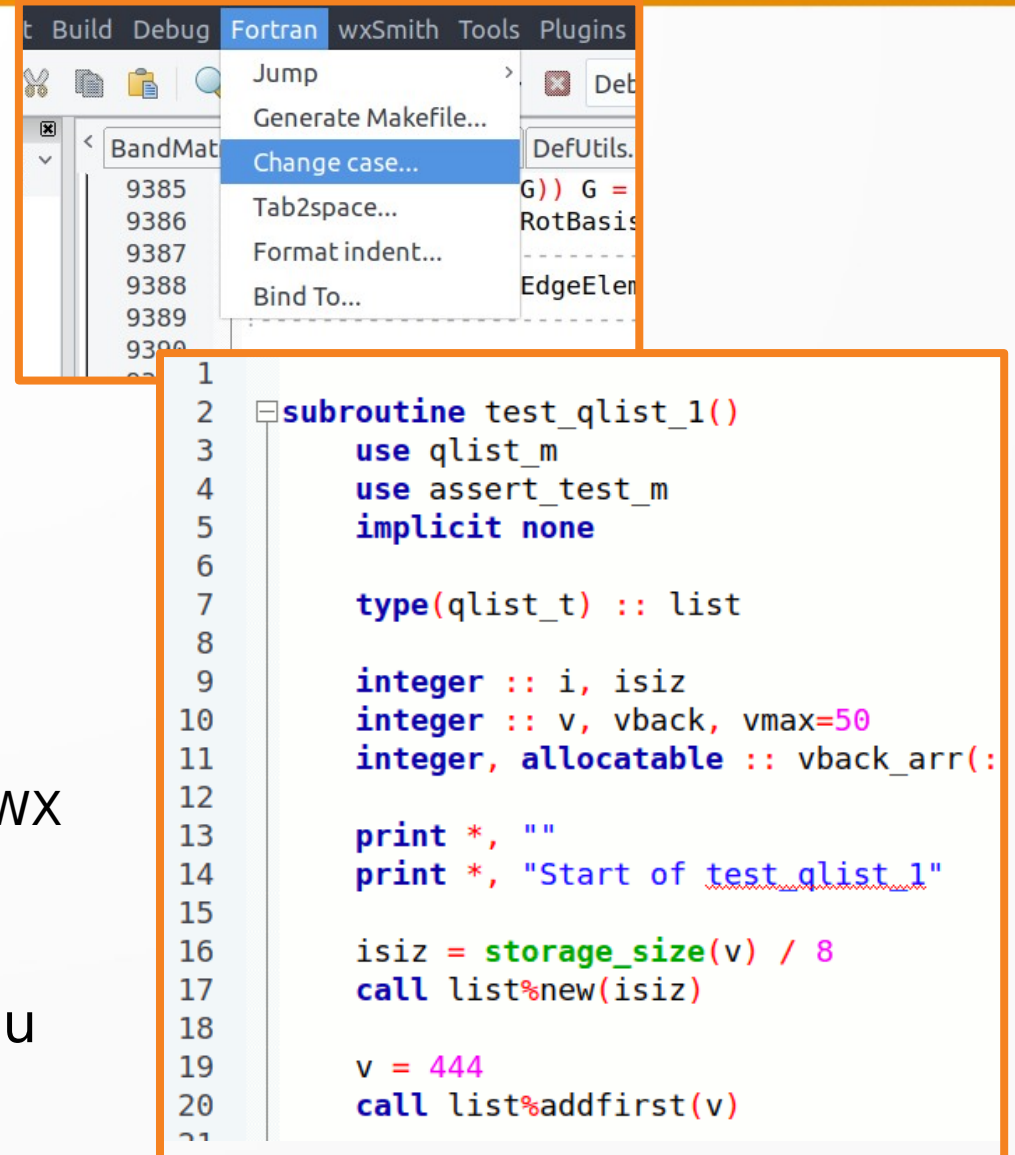
- BrowseTracker plugin:
  - Tracks mouse clicks
  - Menu: View→Jump→Jump Back / Jump Fwrd
  - BrowserTracker toolbar
- Go to the previous files: “Alt+Left”
- Right-click “Jump to ‘Name’”
- Call/Called-By tree



# Code refactoring

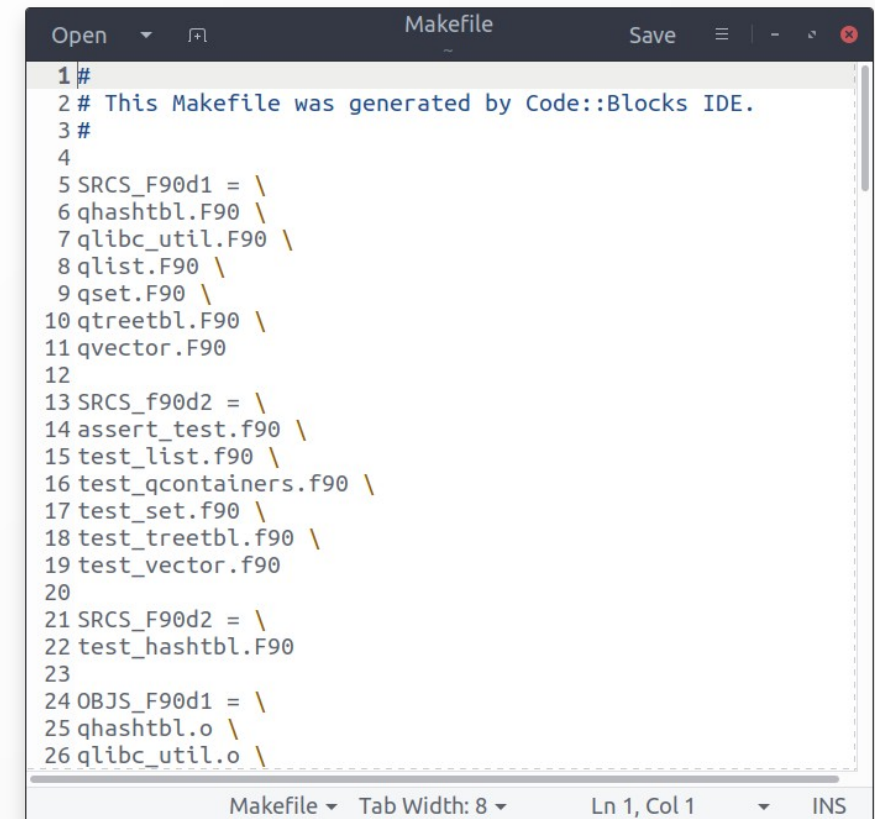
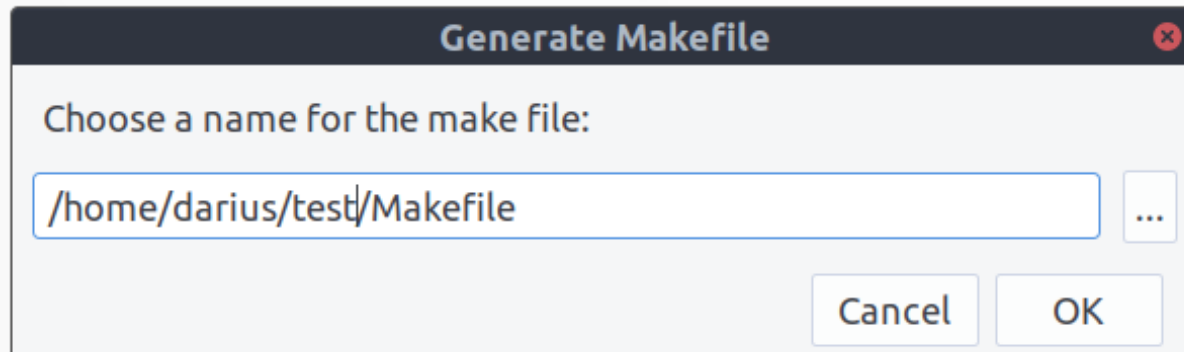
- Change case:
  - Change case for keywords and/or other names
- Tab2space:
  - replaces tabs with spaces
  - useful for fixed form source code
- Format indent:
  - adjusts indentation of the code
  - originally developed as a separate plugin by YWX ([wxFortranIndent@163.com](mailto:wxFortranIndent@163.com))

All refactoring tools are found in Fortran menu



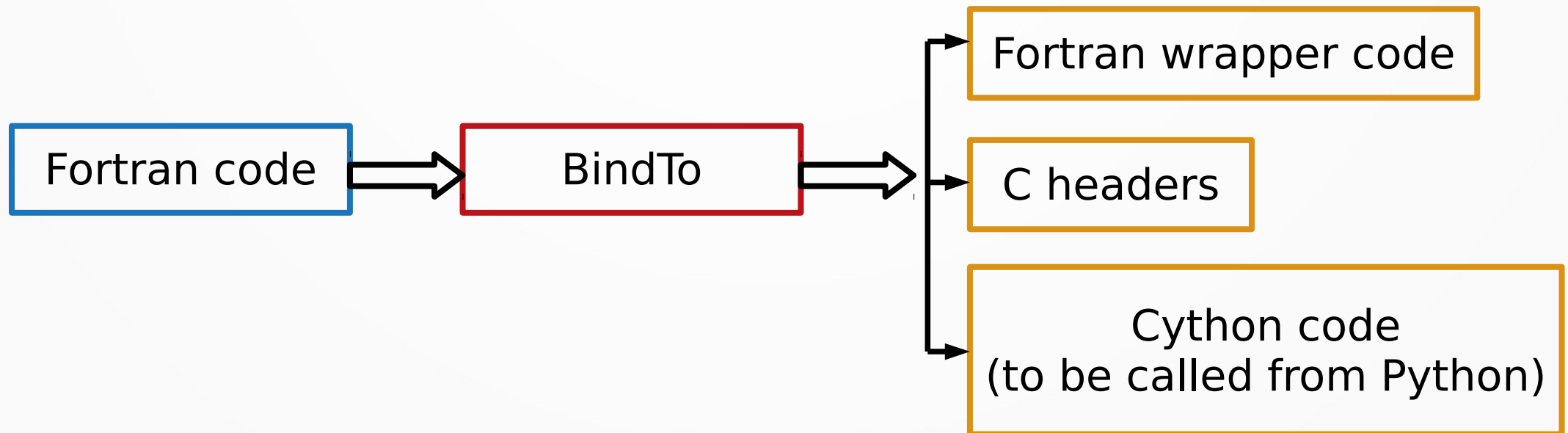
# Generation of makefile

- Generates a makefile for current project
- Access through Fortran menu
- Generated makefile can be used only on Linux (does anybody need it on Windows?)



# BindTo tool

- Generates a wrapper code for Fortran to be called from C and Python
- More about BindTo: <http://cbfortran.sourceforge.net/bindto/>





# Demonstration in Code::Blocks

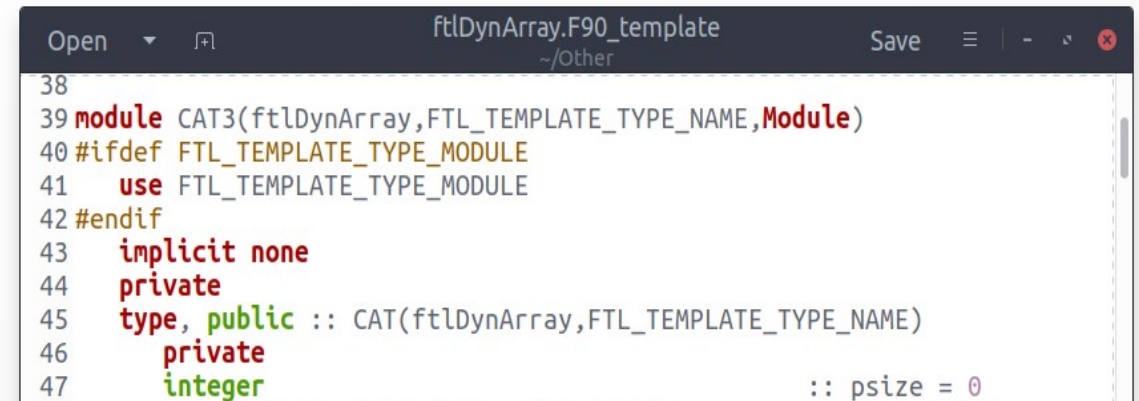
# Ongoing work: a problem

The Fortran Template Library (FTL)  
<https://github.com/SCM-NV/ftl>

Implements:  
generic containers, algorithms,  
string manipulation



```
1  
2 #define FTL_TEMPLATE_TYPE integer  
3 #define FTL_TEMPLATE_TYPE_NAME Int  
4 #define FTL_INSTANTIATE_TEMPLATE  
5 #include "ftlDynArray.F90_template"
```

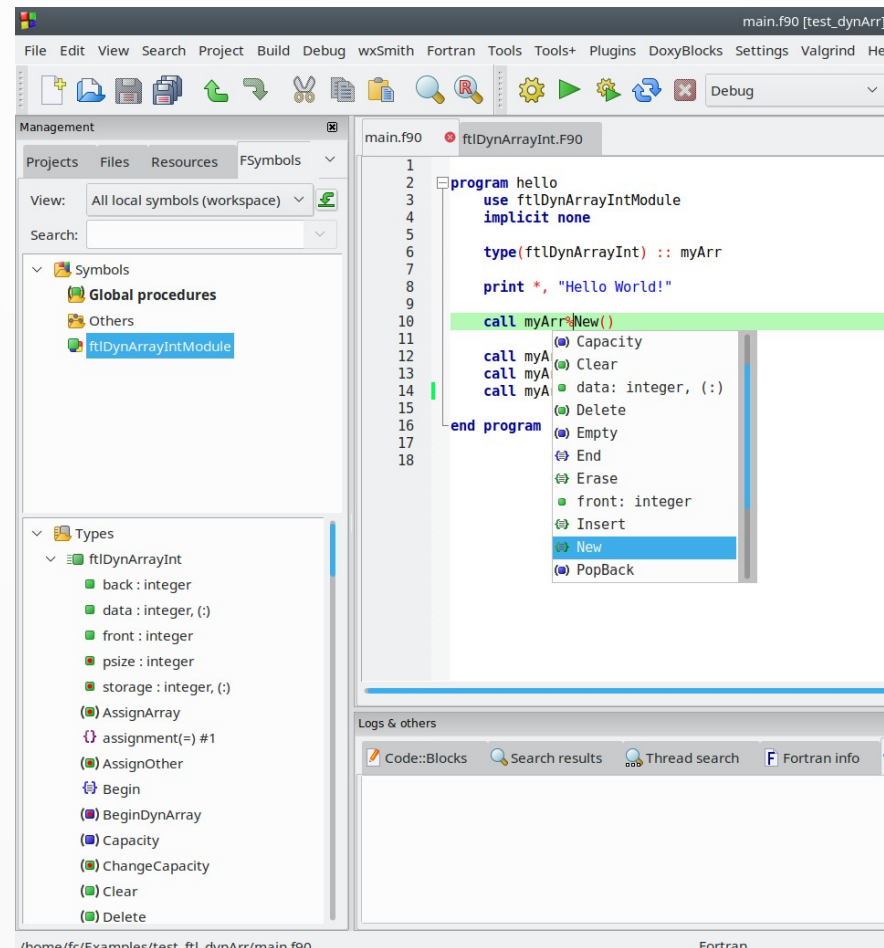


```
38  
39 module CAT3(ftlDynArray,FTL_TEMPLATE_TYPE_NAME,Module)  
40 #ifdef FTL_TEMPLATE_TYPE_MODULE  
41   use FTL_TEMPLATE_TYPE_MODULE  
42 #endif  
43   implicit none  
44   private  
45   type, public :: CAT(ftlDynArray,FTL_TEMPLATE_TYPE_NAME)  
46     private  
47     integer :: psize = 0
```

```
typedef std::vector<int> ftlDynArrayInt
```

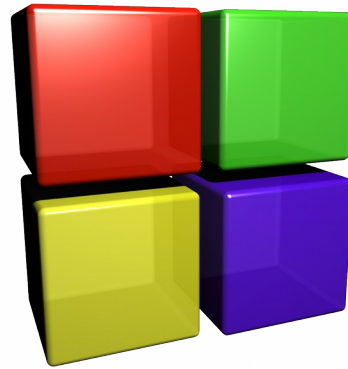
# Ongoing work: a solution

- Implementation of preprocessor directives



Milian Curcic: “Fortran should feel like play and not work”

Try



;)

**Thank you for your attention!**