Debugging with Cmajor Debugger

Seppo Laakko

June 21, 2015

1 Introduction

Cmajor programs can be debugged using the Cmajor source level debugger (cmdb). Currently the program to be debugged and libraries it uses must be first compiled with Cmajor compiler (cmc) using the “C backend” option (-backend=c) and the debug configuration. In Windows programs can be debugged directly in Cmajor Development Environment (IDE) that talks to cmdb behind the scenes. Cmajor debugger in turn talks to GNU debugger (gdb) behind the scenes.

2 Debugging Commands

Table 1 lists available debugging commands, their abbreviations and IDE equivalents.

Table 1: Debugging Commands

Command	**Abbreviation**	**IDE command**
start		
quit	q	`Stop Debugging`
help	h	`Help	Debugging`
continue	c	`Debug	Continue`
next	n	`Debug	Step Over`
step	s	`Debug	Step Into`
out	o	`Debug	Step Out`
break [FILE:]LINE	b [FILE:]LINE	`Debug	Toggle breakpoint`
clear N	cl N	`Debug	Toggle breakpoint`
callstack	ca	
frame N	f N	
list [FILE:]LINE	l [FILE:]LINE	
list*	l*	
list	l	
inspect EXPR	i EXPR	`Debug	Inspect...`
show breakpoints		
set break on throw (on / off)		
empty line	ENTER	

Descriptions of the commands:

- start

  Starts a debugging session.
- quit

  Ends debugging session and exits.
- continue

  Runs program until it stops to a breakpoint or exits.
- next

  Goes to next source line by stepping over function calls.
- step

  Goes to next source line or steps into a function.
- out

  Goes out of function.
- break [FILE:]LINE

  Sets a breakpoint to a source line.
- clear N

  Clears breakpoint number N.
- callstack

  Shows current call stack.
- frame N

  Sets current call frame for inspecting.
- list [FILE:]LINE

  Lists source code around line LINE.
- list*

  Lists source code around current position.
- list

  Lists next lines.
- inspect EXPR

  Inspects value of expression EXPR.
- show breakpoints

  Shows list of breakpoints.
- set break on throw on

  Sets implicit breakpoint in each throw statement.

- set break on throw off

  Clears implicit breakpoint in each throw statement.

- empty line

  Repeats last command.

3 Inspect Expressions

Inspect expression is a content expression.

⟨*inspect-expr*⟩ ::= ⟨*content-expr*⟩

3.1 Content Expression

Content expression is either an *at* character followed by a prefix expression, or a sole prefix expression.

⟨*content-expr*⟩ ::= '@' ⟨*prefix-expr*⟩ | ⟨*prefix-expr*⟩

at-prefixed content expression evaluates a content of a class or pointer to a class. Specializations of *System.Collections.List*, *System.Collections.Set* and *System.Collections.Map*, receive special treatment from the debugger to print the contents of them. *at*-prefixed content expression for an ordinary class prints values of its member variables. *at*-prefixed content expression for a pointer evaluates the dynamic type of the class pointed and prints its content.

3.2 Prefix Expression

A prefix expression consists of a prefix operator '*' followed by another prefix expression, or a postfix expression.

⟨*prefix-expr*⟩ ::= (`*` ⟨*prefix-expr*⟩)
  | ⟨*postfix-expr*⟩

The '*' prefix operator derefences a following pointer expression.

3.3 Postfix Expression

A postfix expression consists of a primary expression followed by zero or more postfix operations.

⟨*postfix-expr*⟩ ::= ⟨*primary-expr*⟩ (`.` ⟨*member-id*⟩ | `->` ⟨*member-id*⟩)*

⟨*member-id*⟩ ::= ⟨*identifier*⟩ | `base`

⟨*identifier*⟩ ::= (letter | '_') (letter | digit | '_')*

A '.' postfix operation selects a member of a class.

A '$->$' postfix operation selects a member of an expression that has a type of a pointer to class.

3.4 Primary Expression

A primary expression is a this-pointer, a name of a local variable a handle expression or a parenthesized perfix expression.

⟨*primary-expr*⟩ ::= `this` | ⟨*identifier*⟩ | `$` ⟨*int*⟩ ⟨*type-expr-part*⟩ | `(` ⟨*prefix-expr*⟩ `)`

⟨*type-expr-part*⟩ ::= '[' (anychar - ']')+ ']'

A handle expression consists of a '$' character followed by a handle returned by GDB and a type expression enclosed in brackets. Handle expression refers to previously evaluated expressions.

4 Usage

```
usage: cmdb [options] program [arguments...]
options:
-----------------------------------------
-ide:               use IDE mode
-file=FILE          read commands from FILE
```