CC0 User's Guide

Ishan Bhargava

Thursday 10th September, 2020
Compiler revision 712

1 Introduction

This document will describe the compiler options available when using CC0, as well as other configuration options. Some of these apply to anyone using CC0 to complete homework assignments, such as warnings. Other options are available for the sake of flexibility.

2 Dynamically-checked contracts

C0 code frequently uses contracts to establish invariants at runtime. Howveer, some of these checks can be very expensive to run. Therefore, CC0 uses the `-d` flag to control whether or not these run.

Starting in September 2020, additional functionality has been added behind `-d`. The runtime will now track function calls. When a crash occurs, the function calls that led to it will be printed out. These helps pinpoint where the problem is.

```
Clac top level
clac>> 1 0 /
c0rt: division by 0
c0rt: in a function called from:
          eval (clac.c0: 237.39-237.50)
          top_level (clac-main.c0: 52.12-52.26)
          main (clac-main.c0: 82.3-82.15)
          (program start)
```

3 Warnings

CC0 has a few warnings implemented which are almost always indicative of typos. These helps cut down on time spent hunting for small errors such as == instead of =, or a misspelled variable name.

Passing the `-W` flag (distinct from `-w`) will enable all of these by default, however you can individually turn them on as well. It is almost always beneficial to leave all of these enabled.

For example:

```
% cc0 -Wunused-expression main.c0
```

This table describes the effect of each warning and provides an example of code which would trigger it.

Name	Explanation

`unreachable-code`	Warns about code which is impossible to execute. For example,
	`int foo(string s) {`
	`  if (string_equal(s, "bad")) {`
	`  error("Argument 's' should not be 'bad'");`
	`  printf("s=%s\n", s); // Triggers warning`
	`  }`
	`  else {`
	`  return string_length(s);`
	`  }`
	
	`  println("Finished foo"); // Triggers warning`
	`}`

`unused-expression` Warns about expressions which don't have any effect. This usually indicates some kind of typo. For example,

```
int clamp(int x, int min, int max) {
  // Both '==' operations below have no effect
  if (x < min) x == min;
  if (x > max) x == max;

  return x;
}
```

`unused-variable` Warns about variables which are declared but never used. This warning can be disabled for a certain variable by prefixing the name with an underscore.

```
int main() {
  int val1 = foo();
  int val2 = bar(val1); // val2 is never used
  int val3 = baz(val1);

  return val3;
}
```

4 Environment variables

The following environment variables affect the C0 runtime (C0RT). Unless you are trying to stress-test the language, the defaults should suffice in almost every case. These are not available when running the program with C0VM, coin, or with a different runtime (e.g. bare, unsafe). It is not necessary to recompile the program after changing one of these.

There are two ways of setting these options. To set it for a single run:

```
% C0_STACK_LIMIT=20000 ./a.out
```

To set it persistently, modify your shell's configuration file e.g. `.bashrc`, `.zshrc`.

```
export C0_STACK_LIMIT=20000
```

Invalid option values (e.g. something that's not a number or out of range) will be reported when the program starts.

Name	Default	Explanation
`C0_STACK_LIMIT`	86306	Sets the maximum recursion depth. If exceeded, the program will print a message explaining what has happened, and then terminate. The default is chosen to be sufficiently high as to not interfere with correct programs, while not segfaulting on programs with infinite recursion. Setting this value too high may result in segfaults with no explanation
`C0_MAX_ARRAYSIZE`	1 GiB	Sets the maximum C0 array allocation size. This value includes array metadata: element size, element count, and pointer to actual data. While it should be safe to set this value higher, the effects of this on the garbage collector are unknown.
`C0_ENABLE_BACKTRACE`	1	If set to a nonzero value and the program is compiled with `-d`, then a backtrace will be printed if the program crashes. Otherwise, it will not be printed.
`C0_BACKTRACE_LIMIT`	50	Controls how many entries will be printed in a backtrace. Note that repeated recursive entries will be collapsed into one entry.