Memory Safety: Unsafe Buffer Usage

Martin Uecker, Graz University of Technology

Number: n3395
Date: 2024-11-23

This paper introduces preliminary wording for memory safe modes as proposed previously in N3211, and specific wording for constraints. This paper only focuses on bounds checking and should be seen as preview to gather feedback. In general, the proposed memory safe modes are designed to be similar to existing compiler diagnostics (e.g. `-Wunsafe-buffer-usage, -Wdangling-pointer, -Warray-bounds,` etc.), GCC's diagnostic pragmas and the pragmas controlling floating-point modes, as well as sanitizers. It should be stressed that the proposed modes are strict subsets of the unrestricted language, i.e. the removal of (or ignoring) all pragmas will not make a strictly conforming program invalid or cause a change observable behavior.

The aim of this paper is to make the proposal more concrete and illustrate how these safe modes could potentially be integrated into the standard (or a TS that amends the standard). In particular, as a first step towards spatial memory safety, this paper forbids pointer arithmetic, unsafe dereferencing, and restricts subscription of arrays. The main ideas are to

1. maintain the invariant that all pointers are dereferencable, and
2. make any memory operations that violates a bound a (run-time) constraint violation.

It is understood that this will require many revisions, more constraints, and other enhancements. This paper is mainly intended as a preview to gather feedback about the general approach. Experience from various ongoing initiatives to improve memory safety in C is most welcome will be taken into account when preparing future versions of this document.

Wording

5.2.2.4 Multi-threaded executions and data races

5.2.2.6 Safe Execution Modes

This document defines the execution modes related to memory safety: STATIC, DYNAMIC, and OFF which are selected at translation time using a pragma (see 6.7.10). The initial mode for each translation unit is OFF. The memory safe modes apply on the level of complete statements and declarations and the mode that applies depends on which mode is active when the first token of the statement or declaration is processed. While a memory safe mode is active additional constraints are in effect at translation time. For the DYNAMIC safe mode, additional run-time constraints are in effect that cause the program to perform a trap when the constraint is violated.

6.5.3.2 Array subscripting

Constraints

Constraints (safety)

2 In STATIC or DYNAMIC safe mode, the operand with pointer type shall be the result of array-to-pointer conversion of a complete array or be a function parameter of pointer type that is declared as an array where the keyword static appears within the [and]. In STATIC safe mode, these arrays shall have a known constant length and the operand of integer type shall be an integer constant expression that is non-negative and smaller than the length.

Runtime Constraint (safety)

3 In DYNAMIC safe mode, if the operand of integer type is negative or equal or larger than the length of the array, a trap is performed.

6.5.4.3 Address and indirection operators

Constraints (safety)

5 In STATIC safe mode, for the indirection operator, the operand shall be an expression formed from the address-of operator applied to an object of complete type, or shall be the identifier a function parameter of pointer type that is declared as an array where the keyword static appears within the [and].

NOTE: These constraints ensure that the pointer is not a null pointer.

Runtime Constraint (safety)

6 In DYNAMIC safe mode, for the indirection operator, if the operand is a null pointer, a trap is performed.

6.5.5 Cast operators

Constraints (safety)

5 In STATIC or DYNAMIC safe mode, there shall be no conversions to a pointer type, except where where permitted by the constraints of 6.5.16.1.

6.5.7 Additive operators

Constraints (safety)

5 In STATIC or DYNAMIC safe mode, none of the operands shall have pointer type, except for array subscripting.

6.5.17.2 Simple assignment

Constraints (safety)

2 In STATIC or DYNAMIC safe mode, the type of the operand to the right shall only be a pointer to a qualified or unqualified version of void if the type of the operand on the left has such a type. In STATIC safe mode, the operand to the right shall not be a null pointer constant.

6.7.10. Pragma Directive

standard-pragma:
    # pragma STDC FP_CONTRACT on-off-switch
    # pragma STDC FENV_ACCESS on-off-switch
    # pragma STDC FENV_DEC_ROUND dec-direction
    # pragma STDC FENV_ROUND direction
    # pragma STDC CX_LIMITED_RANGE on-off-switch
    # pragma STDC SAFETY safe-mode

safe-mode: one of
    DYNAMIC STATIC OFF