```
#include <stdio.h>

int try_it(long i) {
 int a[2]={0x111,0x222};
 int b[2]={0x333,0x444};
 int c[2]={0x555,0x666};
 return b[i];
}
int main() {
 printf("0x%x\n",try_it(-1));
 printf("0x%x\n",try_it(2));
}

% gcc -O0 file.c && ./a.out
```

```
#include <stdio.h>

int try_it(long i) {
 int a[2]={0x111,0x222};
 int b[2]={0x333,0x444};
 int c[2]={0x555,0x666};
 return b[i];
}
int main() {
 printf("0x%x\n",try_it(-1));
 printf("0x%x\n",try_it(2));
}

% gcc -O0 file.c && ./a.out
0x666
0x111
```

```
0x080483c4 <+0>:     push   %ebp
0x080483c5 <+1>:     mov    %esp,%ebp
0x080483c7 <+3>:     sub    $0x20,%esp
0x080483ca <+6>:     movl   $0x111,-0x8(%ebp)
0x080483d1 <+13>:    movl   $0x222,-0x4(%ebp)
0x080483d8 <+20>:    movl   $0x333,-0x10(%ebp)
0x080483df <+27>:    movl   $0x444,-0xc(%ebp)
0x080483e6 <+34>:    movl   $0x555,-0x18(%ebp)
0x080483ed <+41>:    movl   $0x666,-0x14(%ebp)
0x080483f4 <+48>:    mov    0x8(%ebp),%eax
0x080483f7 <+51>:    mov    -0x10(%ebp,%eax,4),%eax
0x080483fb <+55>:    leave
0x080483fc <+56>:    ret
```

```
#include <stdio.h>

int try_it(long i) {
 int a[2]={0x111,0x222};
 int b[2]={0x333,0x444};
 int c[2]={0x555,0x666};
 return b[i];
}
int main() {
 printf("0x%x\n",try_it(-1));
 printf("0x%x\n",try_it(2));
}

% gcc -O1 file.c && ./a.out
```

```
#include <stdio.h>

int try_it(long i) {
 int a[2]={0x111,0x222};
 int b[2]={0x333,0x444};
 int c[2]={0x555,0x666};
 return b[i];
}
int main() {
 printf("0x%x\n",try_it(-1));
 printf("0x%x\n",try_it(2));
}

% gcc -O1 file.c && ./a.out
0x8048459
0xbff24788
```

```
0x080483c4 <+0>:     push   %ebp
0x080483c5 <+1>:     mov    %esp,%ebp
0x080483c7 <+3>:     sub    $0x10,%esp
0x080483ca <+6>:     movl   $0x333,-0x8(%ebp)
0x080483d1 <+13>:    movl   $0x444,-0x4(%ebp)
0x080483d8 <+20>:    mov    0x8(%ebp),%eax
0x080483db <+23>:    mov    -0x8(%ebp,%eax,4),%eax
0x080483df <+27>:    leave
0x080483e0 <+28>:    ret
```

This International Standard imposes no requirements on the behavior of programs that contain undefined behavior.

– the C Standard

From the list of undefined behaviors:

An array subscript is out of range, even if an object is apparently accessible with the given subscript (as in the lvalue expression a[1][7] given the declaration int a[4][5]) (6.5.6).

Permissible undefined behavior ranges from ignoring the situation completely with unpredictable results, to having demons fly out of your nose.

– John F. Woods, 1992 (comp.std.c)

Nasal Demons

Aliasing

```
#include <stdio.h>

void alias(double *f, long *i) {
 *i=0x17;
 *f=0.0;
 printf("0x%lx\n",*i);
}

int main() {
 long x=0;
 alias((double*)&x,&x);
}

% clang -O1 file.c && ./a.out
```

```
#include <stdio.h>

void alias(double *f, long *i) {
 *i=0x17;
 *f=0.0;
 printf("0x%lx\n",*i);
}

int main() {
 long x=0;
 alias((double*)&x,&x);
}

% clang -O1 file.c && ./a.out
0x0
```

```
a.out`alias:
a.out[0x100000f20] <+0>:  pushq  %rbp
a.out[0x100000f21] <+1>:  movq   %rsp, %rbp
a.out[0x100000f24] <+4>:  movq   $0x17, (%rsi)
a.out[0x100000f2b] <+11>: movq   $0x0, (%rdi)
a.out[0x100000f32] <+18>: movq   (%rsi), %rsi
a.out[0x100000f35] <+21>: leaq   0x5a(%rip), %rdi ;
a.out[0x100000f3c] <+28>: xorl   %eax, %eax
a.out[0x100000f3e] <+30>: popq   %rbp
a.out[0x100000f3f] <+31>: jmp    0x100000f74 ; print
a.out[0x100000f44] <+36>: nopw   %cs:(%rax,%rax)
```

```
#include <stdio.h>

void alias(double *f, long *i) {
 *i=0x17;
 *f=0.0;
 printf("0x%lx\n",*i);
}

int main() {
 long x=0;
 alias((double*)&x,&x);
}

% clang -O2 file.c && ./a.out
```

```
#include <stdio.h>

void alias(double *f, long *i) {
 *i=0x17;
 *f=0.0;
 printf("0x%lx\n",*i);
}

int main() {
 long x=0;
 alias((double*)&x,&x);
}

% clang -O2 file.c && ./a.out
0x17
```

```
a.out`alias:
a.out[0x100000f20] <+0>:  pushq  %rbp
a.out[0x100000f21] <+1>:  movq   %rsp, %rbp
a.out[0x100000f24] <+4>:  movq   $0x17, (%rsi)
a.out[0x100000f2b] <+11>: movq   $0x0, (%rdi)
a.out[0x100000f32] <+18>: leaq   0x55(%rip), %rdi ;
a.out[0x100000f39] <+25>: movl   $0x17, %esi
a.out[0x100000f3e] <+30>: xorl   %eax, %eax
a.out[0x100000f40] <+32>: popq   %rbp
a.out[0x100000f41] <+33>: jmp    0x100000f6c ; print
a.out[0x100000f46] <+38>: nopw   %cs:(%rax,%rax)
```

From the list of undefined behaviors:

*The operand of the unary * operator has an invalid value (6.5.3.2).*

(I do not completely understand the standard on this point, but I think the intended meaning is that the value `0.0` has been assigned to the **`f`** pointer, which is thus assigned to the **`i`** pointer, but since `0.0` isn't an integer, it is an invalid value and thus this program is undefined.)

Arithmetic

```
#include <limits.h>
#include <stdio.h>

int is_int_max (long x) {
 return (x+1) < x;
}
int main() {
 printf("%i\n",is_int_max(0));
 printf("%i\n",is_int_max(LONG_MAX));
}

% clang -O0 file.c && ./a.out
```

```
#include <limits.h>
#include <stdio.h>

int is_int_max (long x) {
 return (x+1) < x;
}
int main() {
 printf("%i\n",is_int_max(0));
 printf("%i\n",is_int_max(LONG_MAX));
}

% clang -O0 file.c && ./a.out
0
1
```

```
a.out`is_int_max:
a.out[0x100000ef0] <+0>:  pushq  %rbp
a.out[0x100000ef1] <+1>:  movq   %rsp, %rbp
a.out[0x100000ef4] <+4>:  movq   %rdi, -0x8(%rbp)
a.out[0x100000ef8] <+8>:  movq   -0x8(%rbp), %rdi
a.out[0x100000efc] <+12>: addq   $0x1, %rdi
a.out[0x100000f03] <+19>: cmpq   -0x8(%rbp), %rdi
a.out[0x100000f07] <+23>: setl   %al
a.out[0x100000f0a] <+26>: andb   $0x1, %al
a.out[0x100000f0c] <+28>: movzbl %al, %eax
a.out[0x100000f0f] <+31>: popq   %rbp
a.out[0x100000f10] <+32>: retq
a.out[0x100000f11] <+33>: nopw   %cs:(%rax,%rax)
```

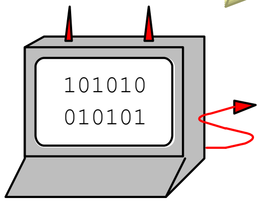
```
#include <limits.h>
#include <stdio.h>

int is_int_max (long x) {
 return (x+1) < x;
}
int main() {
 printf("%i\n",is_int_max(0));
 printf("%i\n",is_int_max(LONG_MAX));
}

% clang -O1 file.c && ./a.out
```

```
#include <limits.h>
#include <stdio.h>

int is_int_max (long x) {
 return (x+1) < x;
}
int main() {
 printf("%i\n",is_int_max(0));
 printf("%i\n",is_int_max(LONG_MAX));
}

% clang -O1 file.c && ./a.out
0
0
```

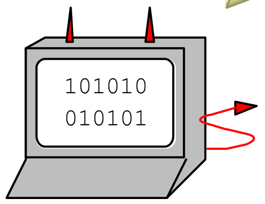
```
a.out`is_int_max:
a.out[0x100000f30] <+0>: pushq  %rbp
a.out[0x100000f31] <+1>: movq   %rsp, %rbp
a.out[0x100000f34] <+4>: xorl   %eax, %eax
a.out[0x100000f36] <+6>: popq   %rbp
a.out[0x100000f37] <+7>: retq
a.out[0x100000f38] <+8>: nopl   (%rax,%rax)
```

There are two situations I must consider:

- **`x+1`** does not overflow: I must return `0`.
- **`x+1`** does overflow: I can do whatever I want.



I know! I'll return 0!



From the list of undefined behaviors:

The value of the result of an integer arithmetic or conversion function cannot be represented (7.8.2.1, 7.8.2.2, 7.8.2.3, 7.8.2.4, 7.20.6.1, 7.20.6.2, 7.20.1).

Conversions (casts)

```
#include <stdio.h>

unsigned long f() {
 /* simplified from real implementation */
 /* of Racket's hashing function */
 double d = -0.5;
 return d * (1 << 30);
}
int main() {
 printf("0x%lx\n",f());
}

% clang -O0 file.c && ./a.out
```

```
#include <stdio.h>

unsigned long f() {
 /* simplified from real implementation */
 /* of Racket's hashing function */
 double d = -0.5;
 return d * (1 << 30);
}
int main() {
 printf("0x%lx\n",f());
}

% clang -O0 file.c && ./a.out
0xffffffffe0000000
```

```
a.out`f:
a.out[0x100000f20] <+0>:  pushq  %rbp
a.out[0x100000f21] <+1>:  movq   %rsp, %rbp
a.out[0x100000f24] <+4>:  movabsq $-0x402000000000000
a.out[0x100000f2e] <+14>: movq   %rax, -0x8(%rbp)
a.out[0x100000f32] <+18>: movq   $-0x20000000, %rax
a.out[0x100000f39] <+25>: popq   %rbp
a.out[0x100000f3a] <+26>: retq
a.out[0x100000f3b] <+27>: nopl   (%rax,%rax)
```

```
#include <stdio.h>

unsigned long f() {
 /* simplified from real implementation */
 /* of Racket's hashing function */
 double d = -0.5;
 return d * (1 << 30);
}
int main() {
 printf("0x%lx\n",f());
}

% clang -O3 file.c && ./a.out
```

```
#include <stdio.h>

unsigned long f() {
 /* simplified from real implementation */
 /* of Racket's hashing function */
 double d = -0.5;
 return d * (1 << 30);
}
int main() {
 printf("0x%lx\n",f());
}

% clang -O3 file.c && ./a.out
0x7fff5a985c48
```

```
a.out`f:
a.out[0x100000f50] <+0>: pushq  %rbp
a.out[0x100000f51] <+1>: movq   %rsp, %rbp
a.out[0x100000f54] <+4>: popq   %rbp
a.out[0x100000f55] <+5>: retq
a.out[0x100000f56] <+6>: nopw   %cs:(%rax,%rax)
```

From the list of undefined behaviors:

Conversion to or from an integer type produces a value outside the range that can be represented (6.3.1.4).

Complete List of Undefined Behaviors in C99

1: A nonempty source file does not end in a new-line character which is not immediately preceded by a backslash character or ends in a partial preprocessing token or comment (5.1.1.2).

2: Token concatenation produces a character sequence matching the syntax of a universal character name (5.1.1.2).

3: A program in a hosted environment does not define a function named main using one of the specified forms (5.1.2.2.1).

4: A character not in the basic source character set is encountered in a source file, except in an identifier, a character constant, a string literal, a header name, a comment, or a preprocessing token that is never converted to a token (5.2.1).

5: An identifier, comment, string literal, character constant, or header name contains an invalid multibyte character or does not begin and end in the initial shift state (5.2.1.2).

6: The same identifier has both internal and external linkage in the same translation unit (6.2.2).

7: An object is referred to outside of its lifetime (6.2.4).

8: The value of a pointer to an object whose lifetime has ended is used (6.2.4).

9: The value of an object with automatic storage duration is used while it is indeterminate (6.2.4, 6.7.8, 6.8).

10: A trap representation is read by an lvalue expression that does not have character type (6.2.6.1).

11: A trap representation is produced by a side effect that modifies any part of the object using an lvalue expression that does not have character type (6.2.6.1).

12: The arguments to certain operators are such that could produce a negative zero result, but the implementation does not support negative zeros (6.2.6.2).

13: Two declarations of the same object or function specify types that are not compatible (6.2.7).

14: Conversion to or from an integer type produces a value outside the range that can be represented (6.3.1.4).

15: Demotion of one real floating type to another produces a value outside the range that can be represented (6.3.1.5).

16: An lvalue does not designate an object when evaluated (6.3.2.1).

16: An lvalue does not designate an object when evaluated (6.3.2.1).

17: A non-array lvalue with an incomplete type is used in a context that requires the value of the designated object (6.3.2.1).

18: An lvalue having array type is converted to a pointer to the initial element of the array, and the array object has register storage class (6.3.2.1).

19: An attempt is made to use the value of a void expression, or an implicit or explicit conversion (except to void) is applied to a void expression (6.3.2.2).

20: Conversion of a pointer to an integer type produces a value outside the range that can be represented (6.3.2.3).

21: Conversion between two pointer types produces a result that is incorrectly aligned (6.3.2.3).

22: A pointer is used to call a function whose type is not compatible with the pointed-to type (6.3.2.3).

23: An unmatched ’ or " character is encountered on a logical source line during tokenization (6.4).

24: A reserved keyword token is used in translation phase 7 or 8 for some purpose other than as a keyword (6.4.1).

25: A universal character name in an identifier does not designate a character whose encoding falls into one of the specified ranges (6.4.2.1).

26: The initial character of an identifier is a universal character name designating a digit (6.4.2.1).

27: Two identifiers differ only in nonsignificant characters (6.4.2.1).

28: The identifier__func__is explicitly declared (6.4.2.2).

29: The program attempts to modify a string literal (6.4.5).

30: The characters ’, \, ", //, or /* occur in the sequence between the < and > delimiters, or the characters ’, \, //, or /* occur in the sequence between the " delimiters, in a header name preprocessing token (6.4.7).

31: Between two sequence points, an object is modified more than once, or is modified and the prior value is read other than to determine the value to be stored (6.5).

32: An exceptional condition occurs during the evaluation of an expression (6.5).

33: An object has its stored value accessed other than by an lvalue of an allowable type (6.5).

34: An attempt is made to modify the result of a function call, a conditional operator, an assignment operator, or a comma operator, or to access it after the next sequence point (6.5.2.2, 6.5.15, 6.5.16, 6.5.17).

the next sequence point (6.5.2.2, 6.5.15, 6.5.16, 6.5.17).

35: For a call to a function without a function prototype in scope, the number of arguments does not equal the number of parameters (6.5.2.2).

36: For call to a function without a function prototype in scope where the function is defined with a function prototype, either the prototype ends with an ellipsis or the types of the arguments after promotion are not compatible with the types of the parameters (6.5.2.2).

37: For a call to a function without a function prototype in scope where the function is not defined with a function prototype, the types of the arguments after promotion are not compatible with those of the parameters after promotion (with certain exceptions) (6.5.2.2).

38: A function is defined with a type that is not compatible with the type (of the expression) pointed to by the expression that denotes the called function (6.5.2.2).

39: The operand of the unary * operator has an invalid value (6.5.3.2).

40: A pointer is converted to other than an integer or pointer type (6.5.4).

41: The value of the second operand of the / or % operator is zero (6.5.5).

42: Addition or subtraction of a pointer into, or just beyond, an array object and an integer type produces a result that does not point into, or just beyond, the same array object (6.5.6).

43: Addition or subtraction of a pointer into, or just beyond, an array object and an integer type produces a result that points just beyond the array object and is used as the operand of a unary * operator that is evaluated (6.5.6).

44: Pointers that do not point into, or just beyond, the same array object are subtracted (6.5.6).

45: An array subscript is out of range, even if an object is apparently accessible with the given subscript (as in the lvalue expression a[1][7] given the declaration int a[4][5]) (6.5.6).

46: The result of subtracting two pointers is not representable in an object of type ptrdiff_t (6.5.6).

47: An expression is shifted by a negative number or by an amount greater than or equal to the width of the promoted expression (6.5.7).

48: An expression having signed promoted type is left-shifted and either the value of the expression is negative or the result of shifting would be not be representable in the promoted type (6.5.7).

49: Pointers that do not point to the same aggregate or union (nor just beyond the same array object) are compared using relational operators (6.5.8).

50: An object is assigned to an inexactly overlapping object or to an exactly overlapping object with incompatible type (6.5.16.1).

51: An expression that is required to be an integer constant expression does not have an integer type; has operands that are not integer constants

51: An expression that is required to be an integer constant expression does not have an integer type; has operands that are not integer constants, enumeration constants, character constants, sizeof expressions whose results are integer constants, or immediately-cast floating constants; or contains casts (outside operands to sizeof operators) other than conversions of arithmetic types to integer types (6.6).

52: A constant expression in an initializer is not, or does not evaluate to, one of the following: an arithmetic constant expression, a null pointer constant, an address constant, or an address constant for an object type plus or minus an integer constant expression (6.6).

53: An arithmetic constant expression does not have arithmetic type; has operands that are not integer constants, floating constants, enumeration constants, character constants, or sizeof expressions; or contains casts (outside operands to sizeof operators) other than conversions of arithmetic types to arithmetic types (6.6).

54: The value of an object is accessed by an array-subscript [], member-access . or −>, address &, or indirection * operator or a pointer cast in creating an address constant (6.6).

55: An identifier for an object is declared with no linkage and the type of the object is incomplete after its declarator, or after its init-declarator if it has an initializer (6.7).

56: A function is declared at block scope with an explicit storage-class specifier other than extern (6.7.1).

57: A structure or union is defined as containing no named members (6.7.2.1).

58: An attempt is made to access, or generate a pointer to just past, a flexible array member of a structure when the referenced object provides no elements for that array (6.7.2.1).

59: When the complete type is needed, an incomplete structure or union type is not completed in the same scope by another declaration of the tag that defines the content (6.7.2.3).

60: An attempt is made to modify an object defined with a const-qualified type through use of an lvalue with non-const-qualified type (6.7.3).

61: An attempt is made to refer to an object defined with a volatile-qualified type through use of an lvalue with non-volatile-qualified type (6.7.3).

62: The specification of a function type includes any type qualifiers (6.7.3).

63: Two qualified types that are required to be compatible do not have the identically qualified version of a compatible type (6.7.3).

64: An object which has been modified is accessed through a restrict-qualified pointer to a const-qualified type, or through a restrict-qualified pointer and another pointer that are not both based on the same object (6.7.3.1).

65: A restrict-qualified pointer is assigned a value based on another restricted pointer whose associated block neither began execution before the block associated with this pointer, nor ended before the assignment (6.7.3.1).

66: A function with external linkage is declared with an inline function specifier, but is not also defined in the same translation unit (6.7.4).

66: A function with external linkage is declared with an inline function specifier, but is not also defined in the same translation unit (6.7.4).

67: Two pointer types that are required to be compatible are not identically qualified, or are not pointers to compatible types (6.7.5.1).

68: The size expression in an array declaration is not a constant expression and evaluates at program execution time to a nonpositive value (6.7.5.2).

69: In a context requiring two array types to be compatible, they do not have compatible element types, or their size specifiers evaluate to unequal values (6.7.5.2).

70: A declaration of an array parameter includes the keyword static within the [and] and the corresponding argument does not provide access to the first element of an array with at least the specified number of elements (6.7.5.3).

71: A storage-class specifier or type qualifier modifies the keyword void as a function parameter type list (6.7.5.3).

72: In a context requiring two function types to be compatible, they do not have compatible return types, or their parameters disagree in use of the ellipsis terminator or the number and type of parameters (after default argument promotion, when there is no parameter type list or when one type is specified by a function definition with an identifier list) (6.7.5.3).

73: The value of an unnamed member of a structure or union is used (6.7.8).

74: The initializer for a scalar is neither a single expression nor a single expression enclosed in braces (6.7.8).

75: The initializer for a structure or union object that has automatic storage duration is neither an initializer list nor a single expression that has compatible structure or union type (6.7.8).

76: The initializer for an aggregate or union, other than an array initialized by a string literal, is not a brace-enclosed list of initializers for its elements or members (6.7.8).

77: An identifier with external linkage is used, but in the program there does not exist exactly one external definition for the identifier, or the identifier is not used and there exist multiple external definitions for the identifier (6.9).

78: A function definition includes an identifier list, but the types of the parameters are not declared in a following declaration list (6.9.1).

79: An adjusted parameter type in a function definition is not an object type (6.9.1).

80: A function that accepts a variable number of arguments is defined without a parameter type list that ends with the ellipsis notation (6.9.1).

81: The } that terminates a function is reached, and the value of the function call is used by the caller (6.9.1).

82: An identifier for an object with internal linkage and an incomplete type is declared with a tentative definition (6.9.2).

83: The token defined is generated during the expansion of a #if or #elif preprocessing directive, or the use of the defined unary operator does not

83: The token defined is generated during the expansion of a #if or #elif preprocessing directive, or the use of the defined unary operator does not match one of the two specified forms prior to macro replacement (6.10.1).

84: The #include preprocessing directive that results after expansion does not match one of the two header name forms (6.10.2).

85: The character sequence in an #include preprocessing directive does not start with a letter (6.10.2).

86: There are sequences of preprocessing tokens within the list of macro arguments that would otherwise act as preprocessing directives (6.10.3).

87: The result of the preprocessing operator # is not a valid character string literal (6.10.3.2).

88: The result of the preprocessing operator ## is not a valid preprocessing token (6.10.3.3).

89: The #line preprocessing directive that results after expansion does not match one of the two well-defined forms, or its digit sequence specifies zero or a number greater than 2147483647 (6.10.4).

90: A non-STDC #pragma preprocessing directive that is documented as causing translation failure or some other form of undefined behavior is encountered (6.10.6).

91: A #pragma STDC preprocessing directive does not match one of the well-defined forms (6.10.6).

92: The name of a predefined macro, or the identifier defined, is the subject of a #define or #undef preprocessing directive (6.10.8).

93: An attempt is made to copy an object to an overlapping object by use of a library function, other than as explicitly allowed (e.g., memmove) (clause 7).

94: A file with the same name as one of the standard headers, not provided as part of the implementation, is placed in any of the standard places that are searched for included source files (7.1.2).

95: A header is included within an external declaration or definition (7.1.2).

96: A function, object, type, or macro that is specified as being declared or defined by some standard header is used before any header that declares or defines it is included (7.1.2).

97: A standard header is included while a macro is defined with the same name as a keyword (7.1.2).

98: The program attempts to declare a library function itself, rather than via a standard header, but the declaration does not have external linkage (7.1.2).

99: The program declares or defines a reserved identifier, other than as allowed by 7.1.4 (7.1.3).

100: The program removes the definition of a macro whose name begins with an underscore and either an uppercase letter or another underscore (7.1.3).

101: An argument to a library function has an invalid value or a type not expected by a function with variable number of arguments (7.1.4).

102: The pointer passed to a library function array parameter does not have a value such that all address computations and object accesses are valid (7.1.4).

103: The macro definition of assert is suppressed in order to access an actual function (7.2).

104: The argument to the assert macro does not have a scalar type (7.2).

105: The CX_LIMITED_RANGE, FENV_ACCESS, or FP_CONTRACT pragma is used in any context other than outside all external declarations or preceding all explicit declarations and statements inside a compound statement (7.3.4, 7.6.1, 7.12.2).

106: The value of an argument to a character handling function is neither equal to the value of EOF nor representable as an unsigned char (7.4).

107: A macro definition of errno is suppressed in order to access an actual object, or the program defines an identifier with the name errno (7.5).

108: Part of the program tests floating-point status flags, sets floating-point control modes, or runs under non-default mode settings, but was translated with the state for the FENV_ACCESS pragma "off" (7.6.1).

109: The exception-mask argument for one of the functions that provide access to the floating-point status flags has a nonzero value not obtained by bitwise OR of the floating-point exception macros (7.6.2).

110: The fesetexceptflag function is used to set floating-point status flags that were not specified in the call to the fegetexceptflag function that provided the value of the corresponding fexcept_t object (7.6.2.4).

111: The argument to fesetenv or feupdateenv is neither an object set by a call to fegetenv or feholdexcept, nor is it an environment macro (7.6.4.3, 7.6.4.4).

112: The value of the result of an integer arithmetic or conversion function cannot be represented (7.8.2.1, 7.8.2.2, 7.8.2.3, 7.8.2.4, 7.20.6.1, 7.20.6.2, 7.20.1).

113: The program modifies the string pointed to by the value returned by the setlocale function (7.11.1.1).

114: The program modifies the structure pointed to by the value returned by the localeconv function (7.11.2.1).

115: A macro definition of math_errhandling is suppressed or the program defines an identifier with the name math_errhandling (7.12).

116: An argument to a floating-point classification or comparison macro is not of real floating type (7.12.3, 7.12.14).

116: An argument to a floating point classification or comparison macro is not of real floating type (7.12.3, 7.12.14).

117: A macro definition of setjmp is suppressed in order to access an actual function, or the program defines an external identifier with the name setjmp (7.13).

118: An invocation of the setjmp macro occurs other than in an allowed context (7.13.2.1).

119: The longjmp function is invoked to restore a nonexistent environment (7.13.2.1).

120: After a longjmp, there is an attempt to access the value of an object of automatic storage class with non-volatile-qualified type, local to the function containing the invocation of the corresponding setjmp macro, that was changed between the setjmp invocation and longjmp call (7.13.2.1).

121: The program specifies an invalid pointer to a signal handler function (7.14.1.1).

122: A signal handler returns when the signal corresponded to a computational exception (7.14.1.1).

123: A signal occurs as the result of calling the abort or raise function, and the signal handler calls the raise function (7.14.1.1).

124: A signal occurs other than as the result of calling the abort or raise function, and the signal handler refers to an object with static storage duration other than by assigning a value to an object declared as volatile sig_atomic_t, or calls any function in the standard library other than the abort function, the _Exit function, or the signal function (for the same signal number) (7.14.1.1).

125: The value of errno is referred to after a signal occurred other than as the result of calling the abort or raise function and the corresponding signal handler obtained a SIG_ERR return from a call to the signal function (7.14.1.1).

126: A signal is generated by an asynchronous signal handler (7.14.1.1).

127: A function with a variable number of arguments attempts to access its varying arguments other than through a properly declared and initialized va_list object, or before the va_start macro is invoked (7.15, 7.15.1.1, 7.15.1.4).

128: The macro va_arg is invoked using the parameter ap that was passed to a function that invoked the macro va_arg with the same parameter (7.15).

129: A macro definition of va_start, va_arg, va_copy, or va_end is suppressed in order to access an actual function, or the program defines an external identifier with the name va_copy or va_end (7.15.1).

130: The va_start or va_copy macro is invoked without a corresponding invocation of the va_end macro in the same function, or vice versa (7.15.1, 7.15.1.2, 7.15.1.3, 7.15.1.4).

131: The type parameter to the va_arg macro is not such that a pointer to an object of that type can be obtained simply by postfixing a * (7.15.1.1).

132: The va_arg macro is invoked when there is no actual next argument, or with a specified type that is not compatible with the promoted type of the actual next argument, with certain exceptions (7.15.1.1).

actual next argument, with certain exceptions (7.15.1.1).

133: The va_copy or va_start macro is called to initialize a va_list that was previously initialized by either macro without an intervening invocation of the va_end macro for the same va_list (7.15.1.2, 7.15.1.4).

134: The parameter parmN of a va_start macro is declared with the register storage class, with a function or array type, or with a type that is not compatible with the type that results after application of the default argument promotions (7.15.1.4).

135: The member designator parameter of an offsetof macro is an invalid right

136: operand of the . operator for the type parameter, or designates a bit-field (7.17).

137: The argument in an instance of one of the integer-constant macros is not a decimal, octal, or hexadecimal constant, or it has a value that exceeds the limits for the corresponding type (7.18.4).

138: A byte input/output function is applied to a wide-oriented stream, or a wide character input/output function is applied to a byte-oriented stream (7.19.2).

139: Use is made of any portion of a file beyond the most recent wide character written to a wide-oriented stream (7.19.2).

140: The value of a pointer to a FILE object is used after the associated file is closed (7.19.3).

141: The stream for the fflush function points to an input stream or to an update stream in which the most recent operation was input (7.19.5.2).

142: The string pointed to by the mode argument in a call to the fopen function does not exactly match one of the specified character sequences (7.19.5.3).

143: An output operation on an update stream is followed by an input operation without an intervening call to the fflush function or a file positioning function, or an input operation on an update stream is followed by an output operation with an intervening call to a file positioning function (7.19.5.3).

144: An attempt is made to use the contents of the array that was supplied in a call to the setvbuf function (7.19.5.6).

145: There are insufficient arguments for the format in a call to one of the formatted input/output functions, or an argument does not have an appropriate type (7.19.6.1, 7.19.6.2, 7.24.2.1, 7.24.2.2).

146: The format in a call to one of the formatted input/output functions or to the strftime or wcsftime function is not a valid multibyte character sequence that begins and ends in its initial shift state (7.19.6.1, 7.19.6.2, 7.23.3.5, 7.24.2.1, 7.24.2.2, 7.24.5.1).

147: In a call to one of the formatted output functions, a precision appears with a conversion specifier other than those described (7.19.6.1, 7.24.2.1).

148: A conversion specification for a formatted output function uses an asterisk to denote an argument-supplied field width or precision, but the corresponding argument is not provided (7.19.6.1, 7.24.2.1).

149: A conversion specification for a formatted output function uses a # or 0 flag with a conversion specifier other than those described (7.19.6.1, 7.24.2.1).

150: A conversion specification for one of the formatted input/output functions uses a length modifier with a conversion specifier other than those described (7.19.6.1, 7.19.6.2, 7.24.2.1, 7.24.2.2).

151: An s conversion specifier is encountered by one of the formatted output functions, and the argument is missing the null terminator (unless a precision is specified that does not require null termination) (7.19.6.1, 7.24.2.1).

152: An n conversion specification for one of the formatted input/output functions includes any flags, an assignment-suppressing character, a field width, or a precision (7.19.6.1, 7.19.6.2, 7.24.2.1, 7.24.2.2).

153: A % conversion specifier is encountered by one of the formatted input/output functions, but the complete conversion specification is not exactly %% (7.19.6.1, 7.19.6.2, 7.24.2.1, 7.24.2.2).

154: An invalid conversion specification is found in the format for one of the formatted input/output functions, or the strftime or wcsftime function (7.19.6.1, 7.19.6.2, 7.23.3.5, 7.24.2.1, 7.24.2.2, 7.24.5.1).

155: The number of characters transmitted by a formatted output function is greater than INT_MAX (7.19.6.1, 7.19.6.3, 7.19.6.8, 7.19.6.10).

156: The result of a conversion by one of the formatted input functions cannot be represented in the corresponding object, or the receiving object does not have an appropriate type (7.19.6.2, 7.24.2.2).

157: A c, s, or [conversion specifier is encountered by one of the formatted input functions, and the array pointed to by the corresponding argument is not large enough to accept the input sequence (and a null terminator if the conversion specifier is s or [) (7.19.6.2, 7.24.2.2).

158: A c, s, or [conversion specifier with an l qualifier is encountered by one of the formatted input functions, but the input is not a valid multibyte character sequence that begins in the initial shift state (7.19.6.2, 7.24.2.2).

159: The input item for a %p conversion by one of the formatted input functions is not a value converted earlier during the same program execution (7.19.6.2, 7.24.2.2).

160: The vfprintf, vfscanf, vprintf, vscanf, vsnprintf, vsprintf, vsscanf, vfwprintf, vfwscanf, vswprintf, vswscanf, vwprintf, or vwscanf function is called with an improperly initialized va_list argument, or the argument is used (other than in an invocation of va_end) after the function returns (7.19.6.8, 7.19.6.9, 7.19.6.10, 7.19.6.11, 7.19.6.12, 7.19.6.13, 7.19.6.14, 7.24.2.5, 7.24.2.6, 7.24.2.7, 7.24.2.8, 7.24.2.9, 7.24.2.10).

161: The contents of the array supplied in a call to the fgets, gets, or fgetws function are used after a read error occurred (7.19.7.2, 7.19.7.7, 7.24.3.2).

162: The file position indicator for a binary stream is used after a call to the ungetc function where its value was zero before the call (7.19.7.11).

163: The file position indicator for a stream is used after an error occurred during a call to the fread or fwrite function (7.19.8.1, 7.19.8.2).

164: A partial element read by a call to the fread function is used (7.19.8.1).

165: The fseek function is called for a text stream with a nonzero offset and either the offset was not returned by a previous successful call to the ftell function on a stream associated with the same file or whence is not SEEK_SET (7.19.9.2).

166: The fsetpos function is called to set a position that was not returned by a previous successful call to the fgetpos function on a stream associated with the same file (7.19.9.3).

167: A non-null pointer returned by a call to the calloc, malloc, or reallocfunction with a zero requested size is used to access an object (7.20.3).

168: The value of a pointer that refers to space deallocated by a call to the free or realloc function is used (7.20.3).

169: The pointer argument to the free or realloc function does not match a pointer earlier returned by calloc, malloc, or realloc, or the space has been deallocated by a call to free or realloc (7.20.3.2, 7.20.3.4).

170: The value of the object allocated by the malloc function is used (7.20.3.3).

171: The value of any bytes in a new object allocated by the realloc function beyond the size of the old object are used (7.20.3.4).

172: The program executes more than one call to the exit function (7.20.4.3).

173: During the call to a function registered with the atexit function, a call is made to the longjmp function that would terminate the call to the registered function (7.20.4.3).

174: The string set up by the getenv or strerror function is modified by the program (7.20.4.5, 7.21.6.2).

175: A command is executed through the system function in a way that is documented as causing termination or some other form of undefined behavior (7.20.4.6).

176: A searching or sorting utility function is called with an invalid pointer argument, even if the number of elements is zero (7.20.5).

177: The comparison function called by a searching or sorting utility function alters the contents of the array being searched or sorted, or returns ordering values inconsistently (7.20.5).

178: The array being searched by the bsearch function does not have its elements in proper order (7.20.5.1).

179: The current conversion state is used by a multibyte/wide character conversion function after changing the LC_CTYPE category (7.20.7).

180: A string or wide string utility function is instructed to access an array beyond the end of an object (7.21.1, 7.24.4).

180: A string or wide string utility function is instructed to access an array beyond the end of an object (7.21.1, 7.24.4).

181: A string or wide string utility function is called with an invalid pointer argument, even if the length is zero (7.21.1, 7.24.4).

182: The contents of the destination array are used after a call to the strxfrm, strftime, wcsxfrm, or wcsftime function in which the specified length was too small to hold the entire null-terminated result (7.21.4.5, 7.23.3.5, 7.24.4.4.4, 7.24.5.1).

183: The first argument in the very first call to the strtok or wcstok is a null pointer (7.21.5.8, 7.24.4.5.7).

184: The type of an argument to a type-generic macro is not compatible with the type of the corresponding parameter of the selected function (7.22).

185: A complex argument is supplied for a generic parameter of a type-generic macro that has no corresponding complex function (7.22).

186: The argument corresponding to an s specifier without an l qualifier in a call to the fwprintf function does not point to a valid multibyte character sequence that begins in the initial shift state (7.24.2.11).

187: In a call to the wcstok function, the object pointed to by ptr does not have the value stored by the previous call for the same wide string (7.24.4.5.7).

188: An mbstate_t object is used inappropriately (7.24.6).

189: The value of an argument of type wint_t to a wide character classification or case mapping function is neither equal to the value of WEOF nor representable as a wchar_t (7.25.1).

190: The iswctype function is called using a different LC_CTYPE category from the one in effect for the call to the wctype function that returned the description (7.25.2.2.1).

191: The towctrans function is called using a different LC_CTYPE category from the one in effect for the call to the wctrans function that returned the description (7.25.3.2.1).

Ack! Help!

- Runtime checking: **`-fsanitize=undefined`**, **`kcc`**
- Compile-time checking: **frama-c**

```c
#include <stdio.h>

int try_it(long i) {
 int a[2]={0x111,0x222};
 int b[2]={0x333,0x444};
 int c[2]={0x555,0x666};
 return b[i];
}
int main() {
 printf("0x%x\n",try_it(-1));
 printf("0x%x\n",try_it(2));
}
```

```
#include <stdio.h>

int try_it(long i) {
 int a[2]={0x111,0x222};
 int b[2]={0x333,0x444};
 int c[2]={0x555,0x666};
 return b[i];
}
int main() {
 printf("0x%x\n",try_it(-1));
 printf("0x%x\n",try_it(2));
}
```

clang -fsanitize=undefined

```
prog.c:7:9: runtime error: index -1 out of
bounds for type 'int [2]'
0x666
0x111
```

```c
#include <stdio.h>

int try_it(long i) {
 int a[2]={0x111,0x222};
 int b[2]={0x333,0x444};
 int c[2]={0x555,0x666};
 return b[i];
}
int main() {
 printf("0x%x\n",try_it(-1));
 printf("0x%x\n",try_it(2));
}
```

frama-c

```
file.c:7:[kernel] warning: accessing out of
bounds index {-1}. assert 0 ≤ i < 2;
```

kcc

```
#include <stdio.h>

int try_it(long i) {
 int a[2]={0x111,0x222};
 int b[2]={0x333,0x444};
 int c[2]={0x555,0x666};
 return b[i];
}
int main() {
 printf("0x%x\n",try_it(-1));
 printf("0x%x\n",try_it(2));
}
```

```
Error: UB-CEA1
Description: A pointer (or array subscript)
outside the bounds of an object.
Type: Undefined behavior.
See also: C11 sec. 6.5.6:8, J.2:1 item 46
  at try_it(/host/x.c:7)
  at main(/host/x.c:10)
  at <file-scope>(<unknown>)
Error: UB-CEE3
Description: Found pointer that refers outside
the bounds of an object + 1.
Type: Undefined behavior.
See also: C11 sec. 6.3.2.1:1, J.2:1 item 19
```

```
#include <limits.h>
#include <stdio.h>

int is_int_max (long x) {
 return (x+1) < x;
}
int main() {
 printf("%i\n",is_int_max(0));
 printf("%i\n",is_int_max(LONG_MAX));
}
```

```
#include <limits.h>
#include <stdio.h>

int is_int_max (long x) {
 return (x+1) < x;
}
int main() {
 printf("%i\n",is_int_max(0));
 printf("%i\n",is_int_max(LONG_MAX));
}
```

clang -fsanitize=undefined

```
prog.c:5:11: runtime error: signed integer
overflow: 9223372036854775807 + 1 cannot be
represented in type 'long'
0
1
```

```
#include <limits.h>
#include <stdio.h>

int is_int_max (long x) {
 return (x+1) < x;
}
int main() {
 printf("%i\n",is_int_max(0));
 printf("%i\n",is_int_max(LONG_MAX));
}
```

frama-c

```
file.c:5:[kernel] warning: signed overflow.
assert x+(long)1 ≤ 9223372036854775807;
```

```
#include <limits.h>
#include <stdio.h>

int is_int_max (long x) {
 return (x+1) < x;
}
int main() {
 printf("%i\n",is_int_max(0));
 printf("%i\n",is_int_max(LONG_MAX));
}
```

kcc

```
0
Error: UB-CCV1
Description: Signed integer overflow.
Type: Undefined behavior.
See also: C11 sec. 6.5:5, J.2:1 item 36
  at is_int_max(/host/x.c:5)
  at main(/host/x.c:9)
  at <file-scope>(<unknown>)
1
```

```
#include <stdio.h>

unsigned long f() {
 /* simplified from real implementation */
 /* of Racket's hashing function */
 double d = -0.5;
 return d * (1 << 30);
}
int main() {
 printf("0x%lx\n",f());
}
```

```
#include <stdio.h>

unsigned long f() {
 /* simplified from real implementation */
 /* of Racket's hashing function */
 double d = -0.5;
 return d * (1 << 30);
}
int main() {
 printf("0x%lx\n",f());
}
```

clang -fsanitize=undefined

```
(a14484886341448488634910.out+0x100000e2b):
runtime error: value -5.36871e+08 is outside
the range of representable values of type
'unsigned long'
0xffffffffe0000000
```

```
#include <stdio.h>

unsigned long f() {
 /* simplified from real implementation */
 /* of Racket's hashing function */
 double d = -0.5;
 return d * (1 << 30);
}
int main() {
 printf("0x%lx\n",f());
}
```

frama-c

```
file.c:7:[kernel] warning: overflow in
conversion of d * (double)(1 << 30)
({-536870912.}) from floating-point to
integer. assert -1 < d*(double)((int)(1<<30));
```

```c
#include <stdio.h>

unsigned long f() {
 /* simplified from real implementation */
 /* of Racket's hashing function */
 double d = -0.5;
 return d * (1 << 30);
}
int main() {
 printf("0x%lx\n",f());
}
```

kcc

```
Error: UB-CCV3
Description: Conversion to integer from float
outside the range that can be represented.
Type: Undefined behavior.
See also: C11 sec. 6.3.1.4:1, J.2:1 item 17
 at f(floatToInt.c:6)
 at main(floatToInt.c:10)
 at <file-scope>(<unknown>)
Execution failed (configuration dumped)
```

```
#include <stdio.h>

void alias(double *f, long *i) {
 *i=0x17;
 *f=0.0;
 printf("0x%lx\n",*i);
}

int main() {
 long x=0;
 alias((double*)&x,&x);
}
```

```
#include <stdio.h>

void alias(double *f, long *i) {
 *i=0x17;
 *f=0.0;
 printf("0x%lx\n",*i);
}

int main() {
 long x=0;
 alias((double*)&x,&x);
}
```

clang -fsanitize=undefined

```
0x0
```

frama-c

no assertion

```
#include <stdio.h>

void alias(double *f, long *i) {
 *i=0x17;
 *f=0.0;
 printf("0x%lx\n",*i);
}

int main() {
 long x=0;
 alias((double*)&x,&x);
}
```

```
#include <stdio.h>

void alias(double *f, long *i) {
 *i=0x17;
 *f=0.0;
 printf("0x%lx\n",*i);
}

int main() {
 long x=0;
 alias((double*)&x,&x);
}
```

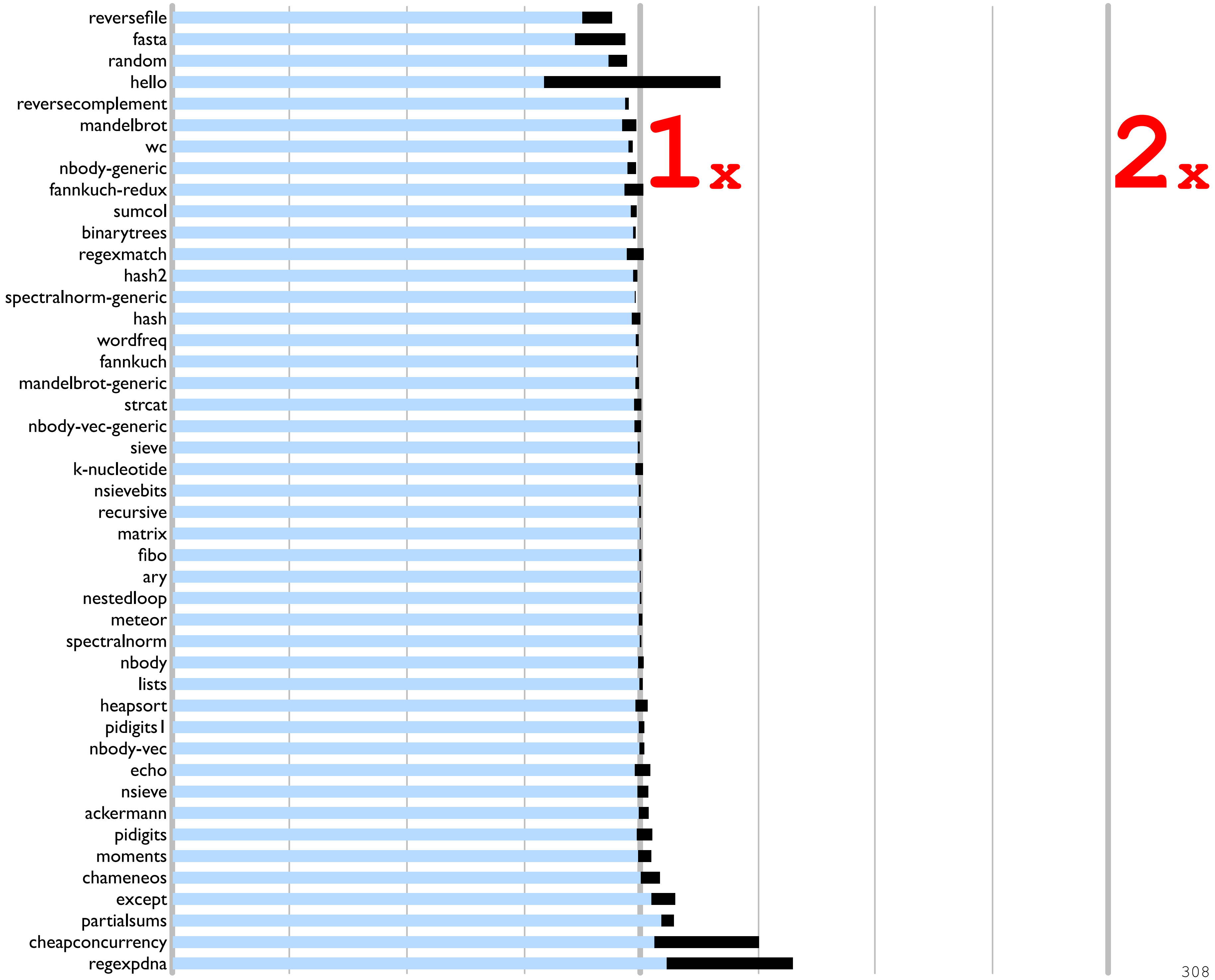
kcc

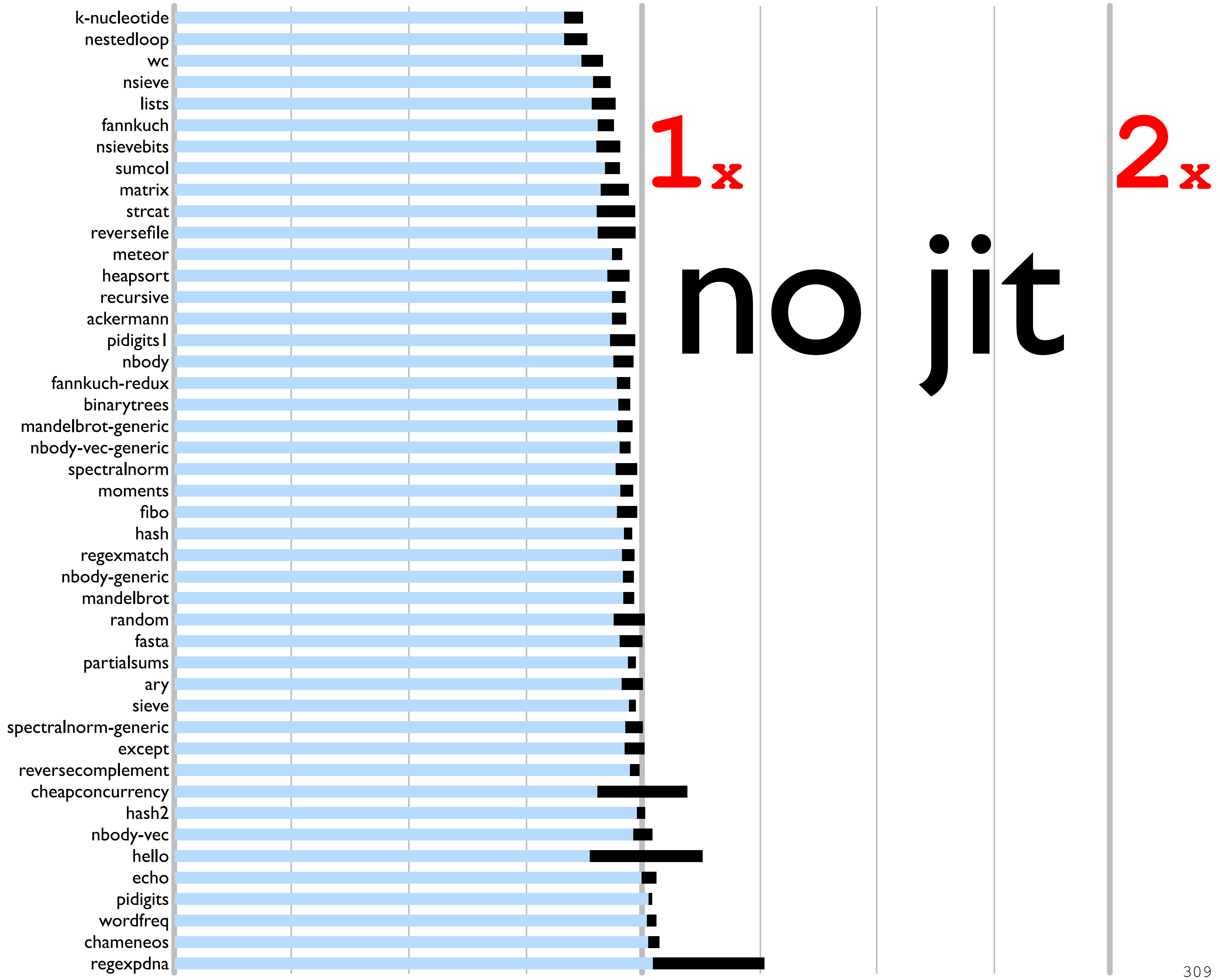
```
Error: UB-EIO10
Description: Type of lvalue not compatible
with the effective type of the object being
accessed.
Type: Undefined behavior.
See also: C11 sec. 6.5:7, J.2:1 item 37
  at alias(/host/x.c:5)
  at main(/host/x.c:11)
  at <file-scope>(<unknown>)
Execution failed (configuration dumped)
```

Any Speedups?

`--fwrapv`: Treat signed integer overflow as two's complement

`--fno-strict-aliasing`: normally, an object of one type is assumed never to reside at the same address as an object of a different type, unless the types are (almost) the same.


reversefile
fasta
random
hello
reversecomplement
mandelbrot
wc
nbody-generic
fannkuch-redux
sumcol
binarytrees
regexmatch
hash2
spectralnorm-generic
hash
wordfreq
fannkuch
mandelbrot-generic
strcat
nbody-vec-generic
sieve
k-nucleotide
nsievebits
recursive
matrix
fibo
ary
nestedloop
meteor
spectralnorm
nbody
lists
heapsort
pidigits1
nbody-vec
echo
nsieve
ackermann
pidigits
moments
chameneos
except
partialsums
cheapconcurrency
regexpdna
1x
2x



k-nucleotide
nestedloop
wc
nsieve
lists
fannkuch
nsievebits
sumcol
matrix
strcat
reversefile
meteor
heapsort
recursive
ackermann
pidigits1
nbody
fannkuch-redux
binarytrees
mandelbrot-generic
nbody-vec-generic
spectralnorm
moments
fibo
hash
regexmatch
nbody-generic
mandelbrot
random
fasta
partialsums
ary
sieve
spectralnorm-generic
except
reversecomplement
cheapconcurrency
hash2
nbody-vec
hello
echo
pidigits
wordfreq
chameneos
regexpdna
1x
2x
no jit