SEI CERT C CODING STANDARDS

[ENV01-C. Do not make assumptions about the size of an environment variable](https://wiki.sei.cmu.edu/confluence/display/c/ENV01-C.+Do+not+make+assumptions+about+the+size+of+an+environment+variable)

## **Noncompliant Code Example**

void f() {

**char** path[PATH\_MAX]; /\* Requires PATH\_MAX to be defined \*/

**strcpy**(path, **getenv**("PATH"));

  /\* Use path \*/

}

## **Compliant Solution**

void f() {

**char** \*path = NULL;

  /\* Avoid assuming $PATH is defined or has limited length \*/

  const **char** \*temp = **getenv**("PATH");

  if (temp != NULL) {

    path = (**char**\*) **malloc**(**strlen**(temp) + 1);

    if (path == NULL) {

      /\* Handle error condition \*/

    } else {

**strcpy**(path, temp);

    }

    /\* Use path \*/

  }

}

# [EXP34-C. Do not dereference null pointers](https://wiki.sei.cmu.edu/confluence/display/c/EXP34-C.+Do+not+dereference+null+pointers)

## **Noncompliant Code Example**

#include <png.h> /\* From libpng \*/

#include <string.h>

void func(png\_structp png\_ptr, **int** length, const void \*user\_data) {

  png\_charp chunkdata;

  chunkdata = (png\_charp)png\_malloc(png\_ptr, length + 1);

  /\* ... \*/

**memcpy**(chunkdata, user\_data, length);

  /\* ... \*/

 }

## **Compliant Solution**

#include <png.h> /\* From libpng \*/

#include <string.h>

 void func(png\_structp png\_ptr, **size\_t** length, const void \*user\_data) {

  png\_charp chunkdata;

  if (length == SIZE\_MAX) {

    /\* Handle error \*/

  }

  chunkdata = (png\_charp)png\_malloc(png\_ptr, length + 1);

  if (NULL == chunkdata) {

    /\* Handle error \*/

  }

  if (NULL == user\_data) {

    /\* Handle error \*/

  }

  /\* ... \*/

**memcpy**(chunkdata, user\_data, length);

  /\* ... \*/

 }

[DCL30-C. Declare objects with appropriate storage durations](https://wiki.sei.cmu.edu/confluence/display/c/DCL30-C.+Declare+objects+with+appropriate+storage+durations)

## **Noncompliant Code Example (Differing Storage Durations)**

#include <stdio.h>

const **char** \*p;

void dont\_do\_this(void) {

  const **char** c\_str[] = "This will change";

  p = c\_str; /\* Dangerous \*/

}

void innocuous(void) {

**printf**("%s\n", p);

}

**int** main(void) {

  dont\_do\_this();

  innocuous();

  return 0;

}

## **Compliant Solution (Same Storage Durations)**

void this\_is\_OK(void) {

  const **char** c\_str[] = "Everything OK";

  const **char** \*p = c\_str;

  /\* ... \*/

}

/\* p is in

# [DCL39-C. Avoid information leakage when passing a structure across a trust boundary](https://wiki.sei.cmu.edu/confluence/display/c/DCL39-C.+Avoid+information+leakage+when+passing+a+structure+across+a+trust+boundary)

# Noncompliant Code Example

#include <string.h>

struct test {

**int** a;

**char** b;

**int** c;

};

/\* Safely copy bytes to user space \*/

extern **int** copy\_to\_user(void \*dest, void \*src, **size\_t** size);

void do\_stuff(void \*usr\_buf) {

  struct test arg;

  /\* Set all bytes (including padding bytes) to zero \*/

**memset**(&arg, 0, sizeof(arg));

  arg.a = 1;

  arg.b = 2;

  arg.c = 3;

  copy\_to\_user(usr\_buf, &arg, sizeof(arg));

}

## **Compliant Solution**

#include <stddef.h>

#include <string.h>

struct test {

  int a;

  char b;

  int c;

};

/\* Safely copy bytes to user space \*/

extern int copy\_to\_user(void \*dest, void \*src, size\_t size);

void do\_stuff(void \*usr\_buf) {

  struct test arg = {.a = 1, .b = 2, .c = 3};

  /\* May be larger than strictly needed \*/

  unsigned char buf[sizeof(arg)];

  size\_t offset = 0;

  memcpy(buf + offset, &arg.a, sizeof(arg.a));

  offset += sizeof(arg.a);

  memcpy(buf + offset, &arg.b, sizeof(arg.b));

  offset += sizeof(arg.b);

  memcpy(buf + offset, &arg.c, sizeof(arg.c));

  offset += sizeof(arg.c);

  /\* Set all remaining bytes to zero \*/

  memset(buff + offset, 0, sizeof(arg) - offset);

  copy\_to\_user(usr\_buf, buf, offset /\* size of info copied \*/);

}

# INT02-C: Understand integer conversion rules.

# Noncompliant code

int si = -1;

unsigned int ui = 1;

printf("%d\n", si < ui);

Compliant solution

int si = -1;

unsigned ui = 1;

printf("%d\n", si < (int)ui);

INT04-C: Enforce limits on integer values originating from tainted sources.

Noncompliant solution:

char\*\* create\_table(void) {

  const char\* const lenstr = getenv("TABLE\_SIZE");

  const size\_t length = lenstr ? strtoul(lenstr, NULL, 10) : 0;

  if (length > SIZE\_MAX / sizeof(char \*))

    return NULL;   /\* Indicate error to caller \*/

  const size\_t table\_size = length \* sizeof(char \*);

  char\*\* const table = (char \*\*)malloc(table\_size);

  if (table == NULL)

    return NULL;   /\* Indicate error to caller \*/

  /\* Initialize table... \*/

  return table;

}

Compliant solution:

enum { MAX\_TABLE\_LENGTH = 256 };

char\*\* create\_table(void) {

  const char\* const lenstr = getenv("TABLE\_SIZE");

  const size\_t length = lenstr ? strtoul(lenstr, NULL, 10) : 0;

  if (length == 0 || length > MAX\_TABLE\_LENGTH)

    return NULL;   /\* Indicate error to caller \*/

  const size\_t table\_size = length \* sizeof(char \*);

  char\*\* const table = (char \*\*)malloc(table\_size);

  if (table == NULL)

    return NULL;   /\* Indicate error to caller \*/

  /\* Initialize table... \*/

  return table;

}

INT30-C. Ensure that unsigned integer operations do not wrap.

c

Noncompliant solution:

void func(unsigned int ui\_a, unsigned int ui\_b) {

  unsigned int usum = ui\_a + ui\_b;

  /\* ... \*/

}

Compliant solution:

precondition test:

include <limits.h>

void func(unsigned int ui\_a, unsigned int ui\_b) {

  unsigned int usum;

  if (UINT\_MAX - ui\_a < ui\_b) {

    /\* Handle error \*/

  } else {

    usum = ui\_a + ui\_b;

  }

  /\* ... \*/

}

Postcondition test :

void func(unsigned int ui\_a, unsigned int ui\_b) {

  unsigned int usum = ui\_a + ui\_b;

  if (usum < ui\_a) {

    /\* Handle error \*/

  }

  /\* ... \*/

}

INT31-C: Ensure that integer conversions do not result in lost or misinterpreted data.

## Noncompliant Code Example (Unsigned to Signed)

#include <limits.h>

void func(void) {

  unsigned long int u\_a = ULONG\_MAX;

  signed char sc;

  sc = (signed char)u\_a; /\* Cast eliminates warning \*/

  /\* ... \*/

}

## Compliant Solution (Unsigned to Signed)

#include <limits.h>

void func(void) {

  unsigned long int u\_a = ULONG\_MAX;

  signed char sc;

  if (u\_a <= SCHAR\_MAX) {

    sc = (signed char)u\_a;  /\* Cast eliminates warning \*/

  } else {

    /\* Handle error \*/

  }

}

INT33-C: Ensure that division and remainder operations do not result in divide-by-zero errors.

### Noncompliant Code Example :

#include <limits.h>

void func(signed long s\_a, signed long s\_b) {

  signed long result;

  if ((s\_a == LONG\_MIN) && (s\_b == -1)) {

    /\* Handle error \*/

  } else {

    result = s\_a / s\_b;

  }

  /\* ... \*/

}

### Compliant Solution

#include <limits.h>

void func(signed long s\_a, signed long s\_b) {

  signed long result;

  if ((s\_b == 0) || ((s\_a == LONG\_MIN) && (s\_b == -1))) {

    /\* Handle error \*/

  } else {

    result = s\_a / s\_b;

  }

  /\* ... \*/

}

MSC01-C: Strive for logical completeness.

## Noncompliant Code Example (if Chain):

if (a == b) {

  /\* ... \*/

}

else if (a == c) {

  /\* ... \*/

}

## Compliant Solution (if Chain)

if (a == b) {

  /\* ... \*/

}

else if (a == c) {

  /\* ... \*/

}

else {

  /\* Handle error condition \*/

}

**MSC12-C: Detect and remove code that has no effect or is never executed.**

## Noncompliant Code Example

int func(int condition) {

    char \*s = NULL;

    if (condition) {

        s = (char \*)malloc(10);

        if (s == NULL) {

           /\* Handle Error \*/

        }

        /\* Process s \*/

        return 0;

    }

    /\* ... \*/

    if (s) {

        /\* This code is unreachable \*/

    }

    return 0;

}

## Compliant Solution

int func(int condition) {

    char \*s = NULL;

    if (condition) {

        s = (char \*)malloc(10);

        if (s == NULL) {

           /\* Handle error \*/

        }

        /\* Process s \*/

    }

    /\* ... \*/

    if (s) {

        /\* This code is now reachable \*/

    }

    return 0;

}

MSC13-C: Detect and remove unused values.

## Noncompliant Code Example

int \*p1;

int \*p2;

p1 = foo();

p2 = bar();

if (baz()) {

  return p1;

}

else {

  p2 = p1;

}

return p2;

## Compliant Solution

int \*p1 = foo();

/\* Removable if bar() does not produce any side effects \*/

(void)bar();

/\* Removable if baz() does not produce any side effects \*/

(void)baz();

return p1;