U18CO018

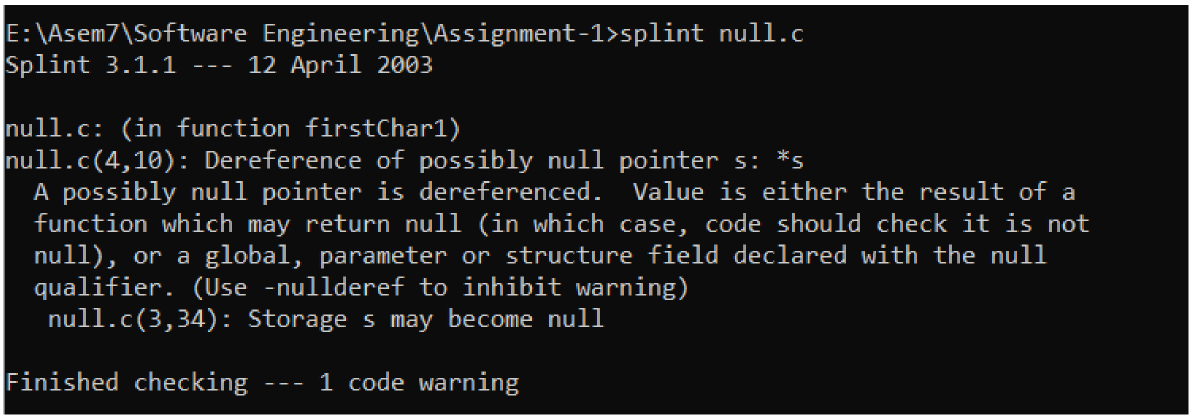
Shekhaliya Shubham

Software Engineering

Assignment 1

● Dereferencing a possibly null pointer.

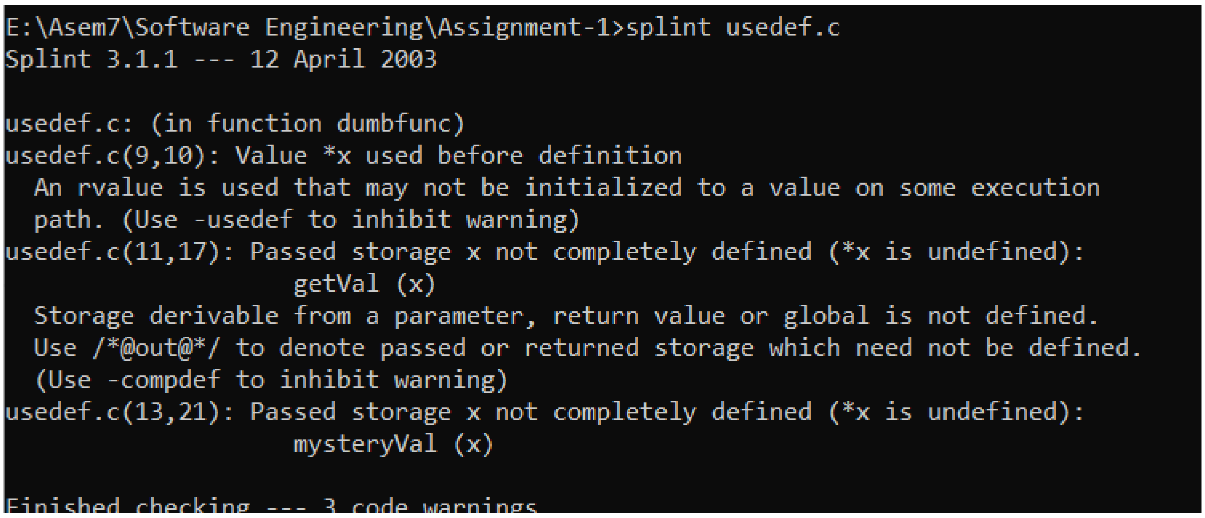
```c
#include <stdio.h>

char firstChar1(/*@null@*/ char *s) {
    return *s;
}

char firstChar2(/*@null@*/ char *s) {
    if (s == NULL)
        return '\0';
    return *s;
}
```



```
E:\Asem7\Software Engineering\Assignment-1>splint null.c
Splint 3.1.1 --- 12 April 2003

null.c: (in function firstChar1)
null.c(4,10): Dereference of possibly null pointer s: *s
  A possibly null pointer is dereferenced.  Value is either the result of a
  function which may return null (in which case, code should check it is not
  null), or a global, parameter or structure field declared with the null
  qualifier. (Use -nullderef to inhibit warning)
   null.c(3,34): Storage s may become null

Finished checking --- 1 code warning
```

● Using possibly undefined storage or returning storage that is not properly defined.

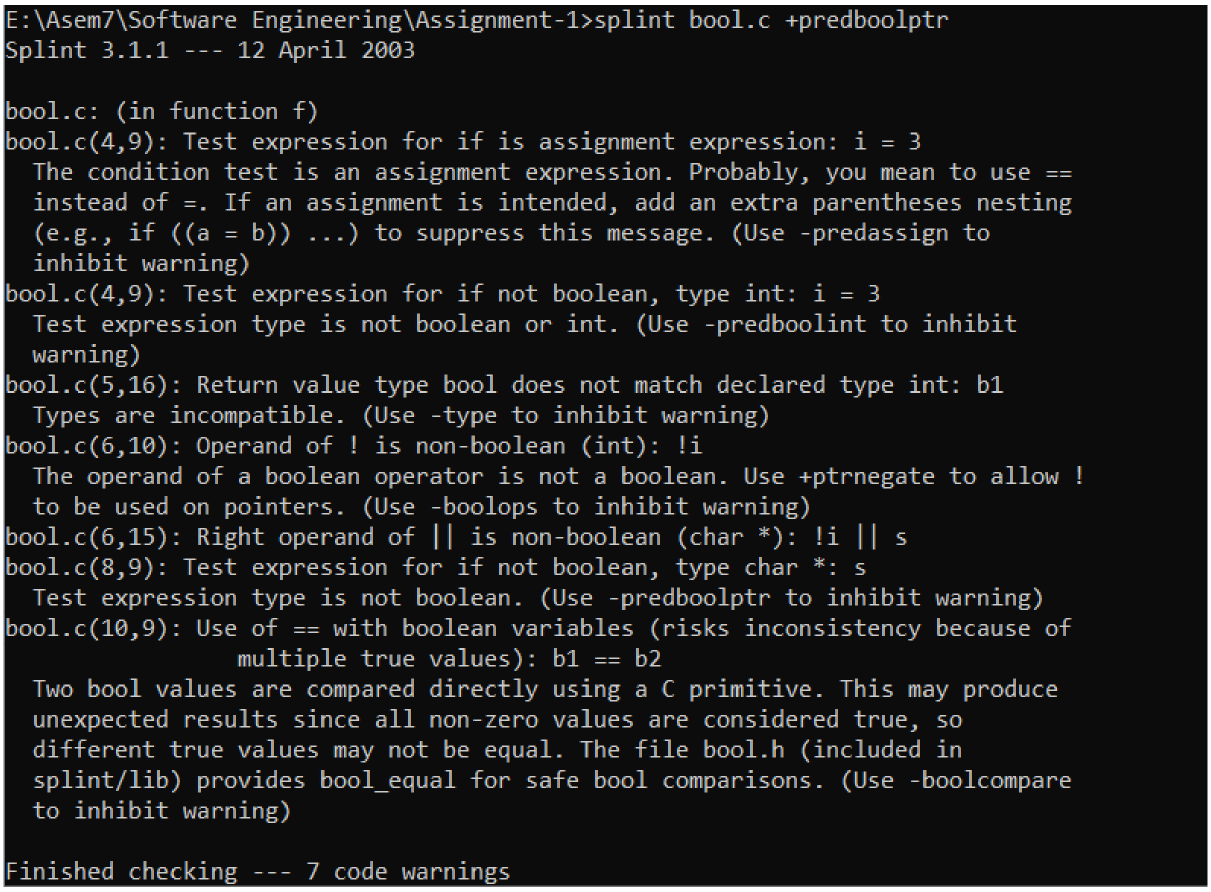
```c
#include <stdio.h>

void setVal(/*@out@*/ int *x);
int getVal(/*@in@*/ int *x);
int mysteryVal(int *x);

int dumbfunc(/*@out@*/ int *x, int i) {
    if (i > 3)
        return *x;
    else if (i > 1)
        return getVal(x);
    else if (i == 0)
        return mysteryVal(x);
    else {
        setVal(x);
        return *x;
    }
}
```



```
E:\Asem7\Software Engineering\Assignment-1>splint usedef.c
Splint 3.1.1 --- 12 April 2003

usedef.c: (in function dumbfunc)
usedef.c(9,10): Value *x used before definition
  An rvalue is used that may not be initialized to a value on some execution
  path. (Use -usedef to inhibit warning)
usedef.c(11,17): Passed storage x not completely defined (*x is undefined):
                    getVal (x)
  Storage derivable from a parameter, return value or global is not defined.
  Use /*@out@*/ to denote passed or returned storage which need not be defined.
  (Use -compdef to inhibit warning)
usedef.c(13,21): Passed storage x not completely defined (*x is undefined):
                    mysteryVal (x)

Finished checking --- 3 code warnings
```

● Type mismatches, with greater precision and flexibility than provided by C compilers.

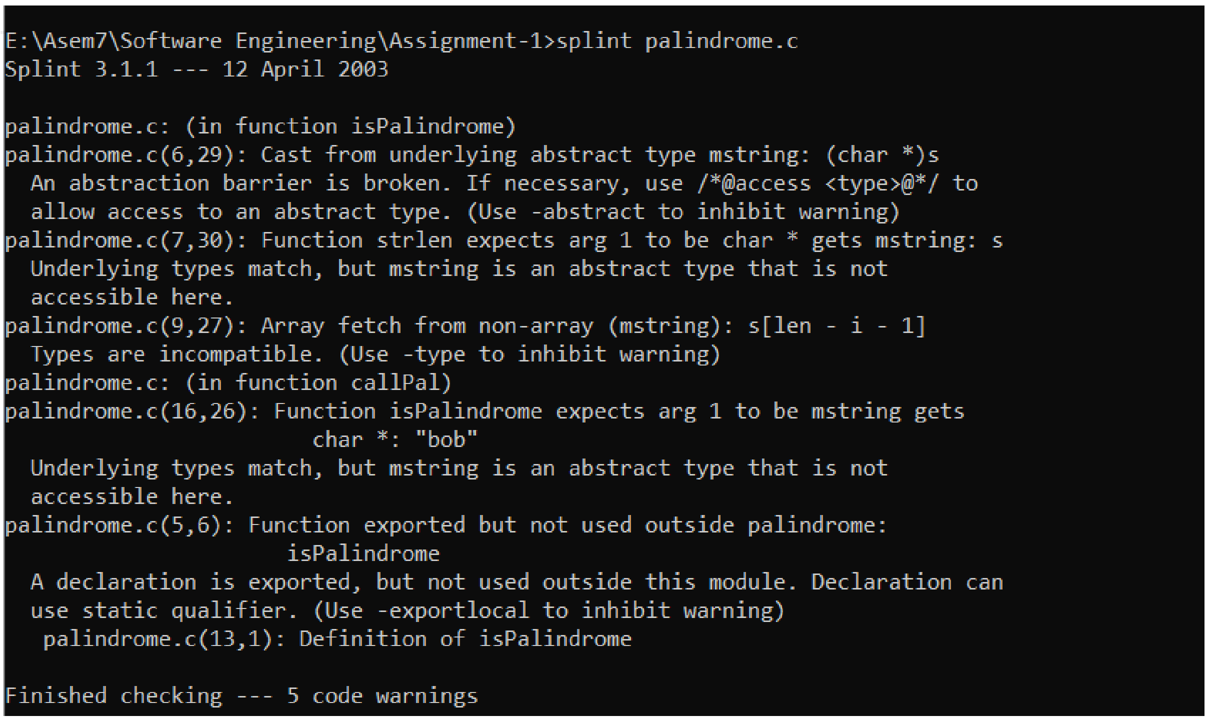
```c
#include <stdbool.h>

int f(int i, char* s, bool b1, bool b2) {
    if (i = 3)
        return b1;
    if (!i || s)
        return i;
    if (s)
        return 7;
    if (b1 == b2)
        return 3;
    return 2;
}
```



```
E:\Asem7\Software Engineering\Assignment-1>splint bool.c +predboolptr
Splint 3.1.1 --- 12 April 2003

bool.c: (in function f)
bool.c(4,9): Test expression for if is assignment expression: i = 3
  The condition test is an assignment expression. Probably, you mean to use ==
  instead of =. If an assignment is intended, add an extra parentheses nesting
  (e.g., if ((a = b)) ...) to suppress this message. (Use -predassign to
  inhibit warning)
bool.c(4,9): Test expression for if not boolean, type int: i = 3
  Test expression type is not boolean or int. (Use -predboolint to inhibit
  warning)
bool.c(5,16): Return value type bool does not match declared type int: b1
  Types are incompatible. (Use -type to inhibit warning)
bool.c(6,10): Operand of ! is non-boolean (int): !i
  The operand of a boolean operator is not a boolean. Use +ptrnegate to allow !
  to be used on pointers. (Use -boolops to inhibit warning)
bool.c(6,15): Right operand of || is non-boolean (char *): !i || s
bool.c(8,9): Test expression for if not boolean, type char *: s
  Test expression type is not boolean. (Use -predboolptr to inhibit warning)
bool.c(10,9): Use of == with boolean variables (risks inconsistency because of
                 multiple true values): b1 == b2
  Two bool values are compared directly using a C primitive. This may produce
  unexpected results since all non-zero values are considered true, so
  different true values may not be equal. The file bool.h (included in
  splint/lib) provides bool_equal for safe bool comparisons. (Use -boolcompare
  to inhibit warning)

Finished checking --- 7 code warnings
```

● Violations of information hiding.

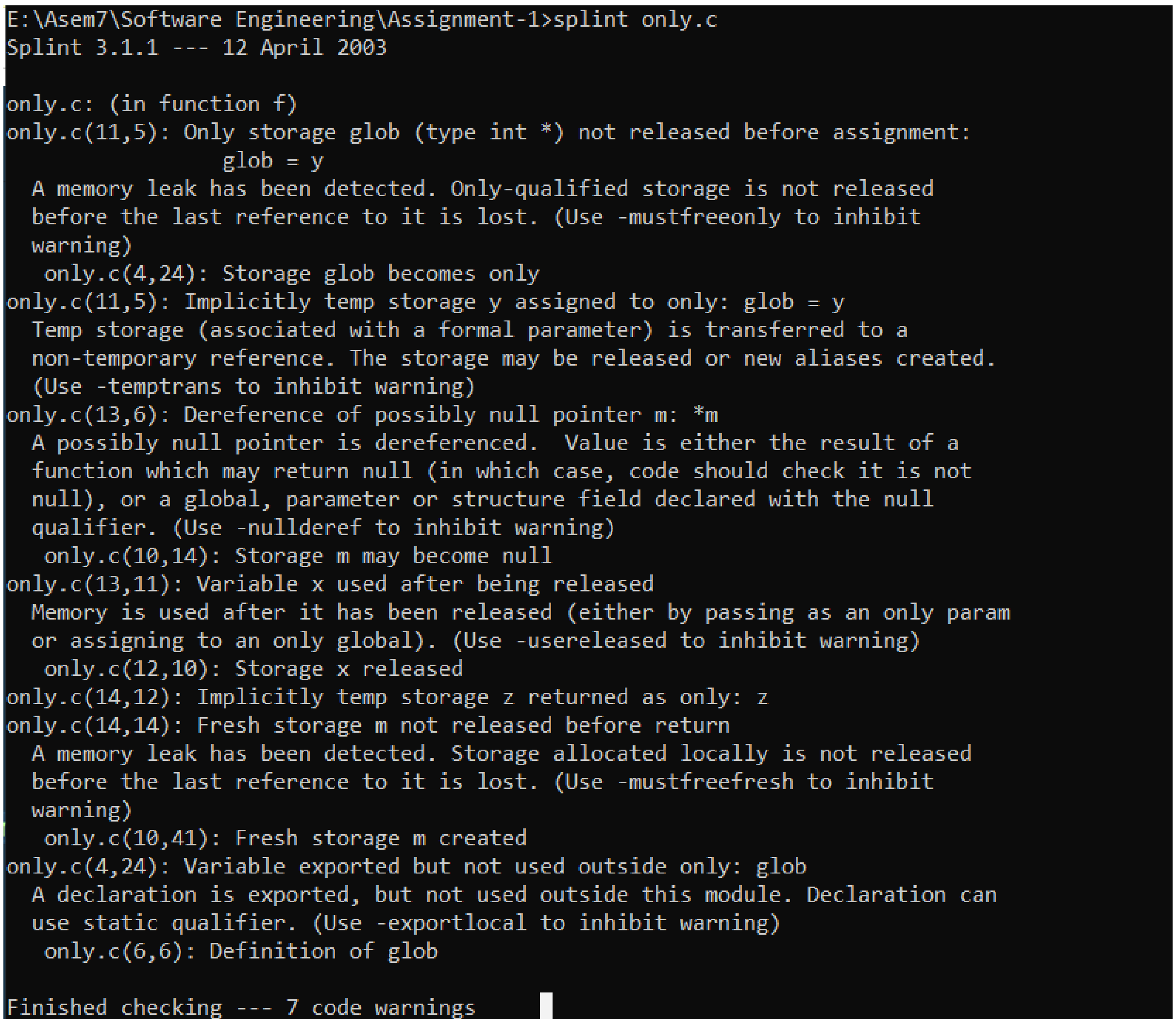
```
#include <stdbool.h>
#include <string.h>
#include "mstring.h"

bool isPalindrome(mstring s) {
    char *current = (char *)s;
    int i, len = (int)strlen(s);
    for (i = 0; i <= (len + 1) / 2; i++) {
        if (current[i] != s[len - i - 1])
            return false;
    }
    return true;
}

bool callPal(void) {
    return (isPalindrome("bob"));
}
```



```
E:\Asem7\Software Engineering\Assignment-1>splint palindrome.c
Splint 3.1.1 --- 12 April 2003

palindrome.c: (in function isPalindrome)
palindrome.c(6,29): Cast from underlying abstract type mstring: (char *)s
  An abstraction barrier is broken. If necessary, use /*@access <type>@*/ to
  allow access to an abstract type. (Use -abstract to inhibit warning)
palindrome.c(7,30): Function strlen expects arg 1 to be char * gets mstring: s
  Underlying types match, but mstring is an abstract type that is not
  accessible here.
palindrome.c(9,27): Array fetch from non-array (mstring): s[len - i - 1]
  Types are incompatible. (Use -type to inhibit warning)
palindrome.c: (in function callPal)
palindrome.c(16,26): Function isPalindrome expects arg 1 to be mstring gets
                        char *: "bob"
  Underlying types match, but mstring is an abstract type that is not
  accessible here.
palindrome.c(5,6): Function exported but not used outside palindrome:
                      isPalindrome
  A declaration is exported, but not used outside this module. Declaration can
  use static qualifier. (Use -exportlocal to inhibit warning)
   palindrome.c(13,1): Definition of isPalindrome

Finished checking --- 5 code warnings
```

● Memory management errors including uses of dangling references and memory leaks.

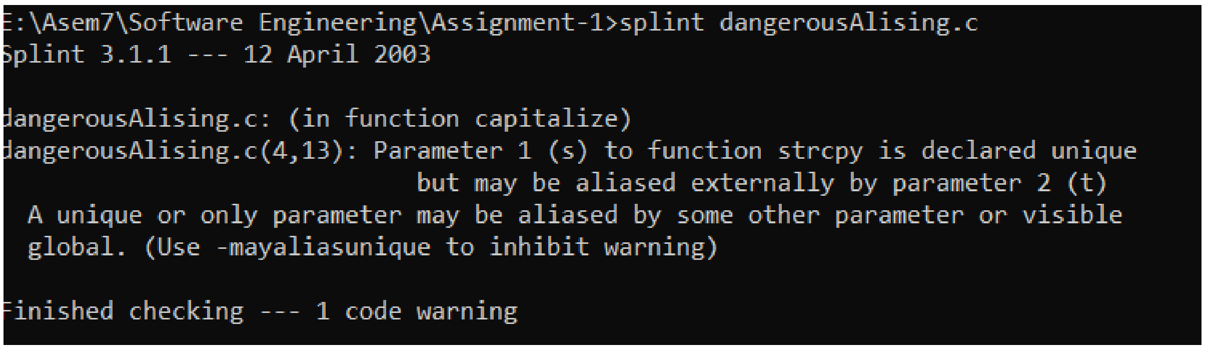
```c
#include <stdio.h>
#include <stdlib.h>

extern /*@only@*/ int *glob;

int *glob;

/*@only@*/ int *f(/*@only@*/ int *x, int *y, int *z)
/*@globals glob;@*/ {
    int *m = (int *)malloc(sizeof(int));
    glob = y; //Memory leak
    free(x);
    *m = *x;  //Use after free
    return z; //Memory leak detected
}
```



```
E:\Asem7\Software Engineering\Assignment-1>splint only.c
Splint 3.1.1 --- 12 April 2003

only.c: (in function f)
only.c(11,5): Only storage glob (type int *) not released before assignment:
                 glob = y
  A memory leak has been detected. Only-qualified storage is not released
  before the last reference to it is lost. (Use -mustfreeonly to inhibit
  warning)
   only.c(4,24): Storage glob becomes only
only.c(11,5): Implicitly temp storage y assigned to only: glob = y
  Temp storage (associated with a formal parameter) is transferred to a
  non-temporary reference. The storage may be released or new aliases created.
  (Use -temptrans to inhibit warning)
only.c(13,6): Dereference of possibly null pointer m: *m
  A possibly null pointer is dereferenced.  Value is either the result of a
  function which may return null (in which case, code should check it is not
  null), or a global, parameter or structure field declared with the null
  qualifier. (Use -nullderef to inhibit warning)
   only.c(10,14): Storage m may become null
only.c(13,11): Variable x used after being released
  Memory is used after it has been released (either by passing as an only param
  or assigning to an only global). (Use -usereleased to inhibit warning)
   only.c(12,10): Storage x released
only.c(14,12): Implicitly temp storage z returned as only: z
only.c(14,14): Fresh storage m not released before return
  A memory leak has been detected. Storage allocated locally is not released
  before the last reference to it is lost. (Use -mustfreefresh to inhibit
  warning)
   only.c(10,41): Fresh storage m created
only.c(4,24): Variable exported but not used outside only: glob
  A declaration is exported, but not used outside this module. Declaration can
  use static qualifier. (Use -exportlocal to inhibit warning)
   only.c(6,6): Definition of glob

Finished checking --- 7 code warnings
```

● Dangerous aliasing.

```c
# include <string.h>

void capitalize (/*@out@*/ char *s, char *t) {
    strcpy (s, t);
    *s = toupper (*s);
}
```



```
:\Asem7\Software Engineering\Assignment-1>splint dangerousAlising.c
plint 3.1.1 --- 12 April 2003

angerousAlising.c: (in function capitalize)
angerousAlising.c(4,13): Parameter 1 (s) to function strcpy is declared unique
                            but may be aliased externally by parameter 2 (t)
  A unique or only parameter may be aliased by some other parameter or visible
  global. (Use -mayaliasunique to inhibit warning)

inished checking --- 1 code warning
```