Software Engineering (CS401)

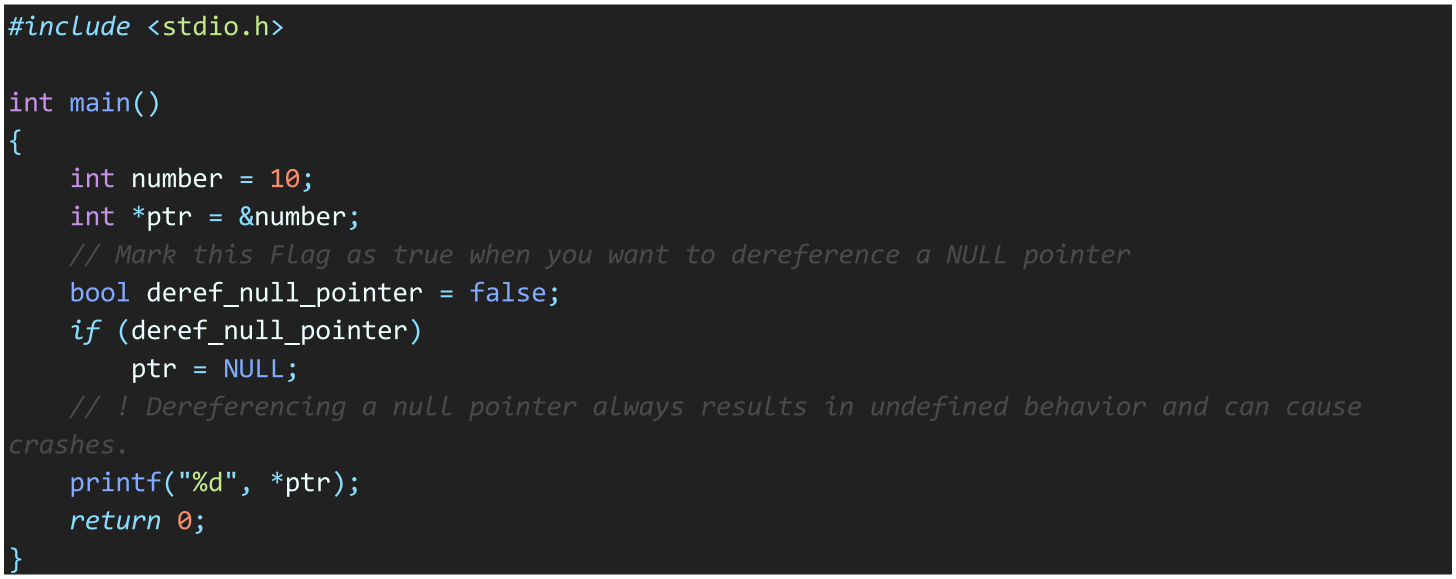
Assignment 1

U19CS012

Q1.) From the Errors listed below, Design a C program Fragment to compare the outputs of **Splint** and the **Standard C** Compiler.

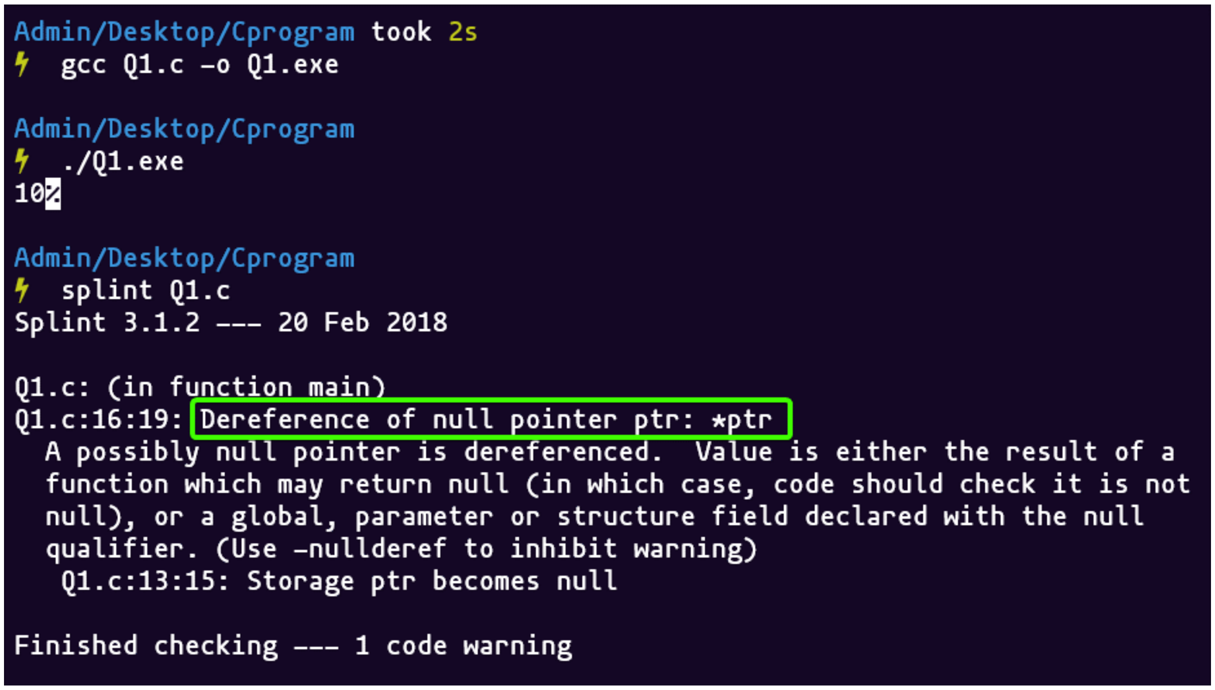
A.) **Dereferencing** a Possibly NULL Pointer.

Code



```c
#include <stdio.h>

int main()
{
    int number = 10;
    int *ptr = &number;
    // Mark this Flag as true when you want to dereference a NULL pointer
    bool deref_null_pointer = false;
    if (deref_null_pointer)
        ptr = NULL;
    // ! Dereferencing a null pointer always results in undefined behavior and can cause
crashes.
    printf("%d", *ptr);
    return 0;
}
```

Output



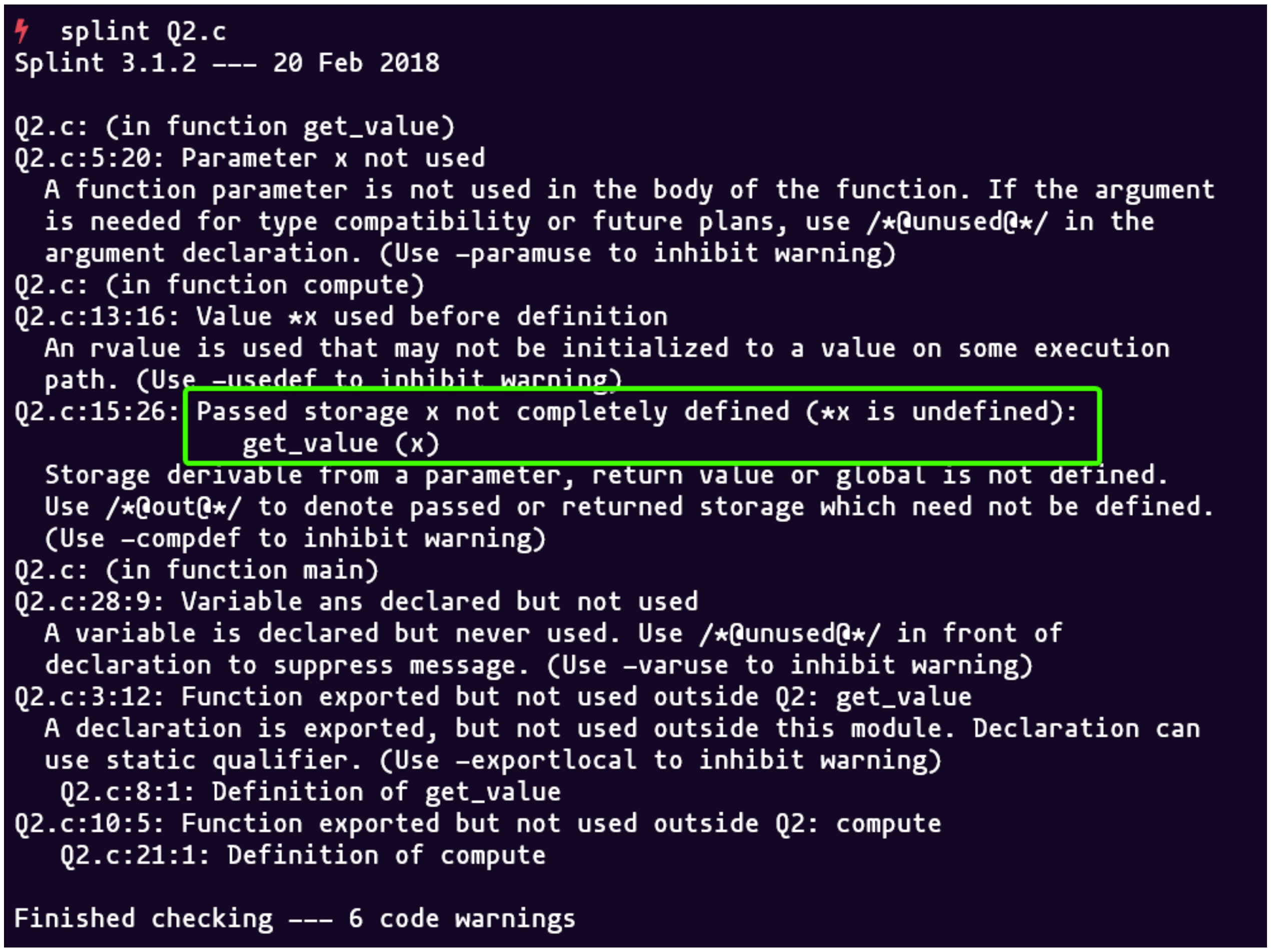
```
Admin/Desktop/Cprogram took 2s
ϟ  gcc Q1.c -o Q1.exe

Admin/Desktop/Cprogram
ϟ  ./Q1.exe
10%

Admin/Desktop/Cprogram
ϟ  splint Q1.c
Splint 3.1.2 --- 20 Feb 2018

Q1.c: (in function main)
Q1.c:16:19: Dereference of null pointer ptr: *ptr
  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)
   Q1.c:13:15: Storage ptr becomes null

Finished checking --- 1 code warning
```

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

Code

```c
#include <stdio.h>
extern void set_value(/*@out@*/ int *x);
extern int get_value(/*@in@*/ int *x);

int get_value(int *x)
{
    return 12;
}

int compute(/*@out@*/ int *x, int id)
{
    if (id > 1)
        return *x;
    else if (id > 5)
        return get_value(x);
    else
    {
        set_value(x);
        return *x;
    }
}

int main()
{
    int k = 10;
    int *p = &k;

    int ans = compute(p, k);

    return 0;
}
```

Output



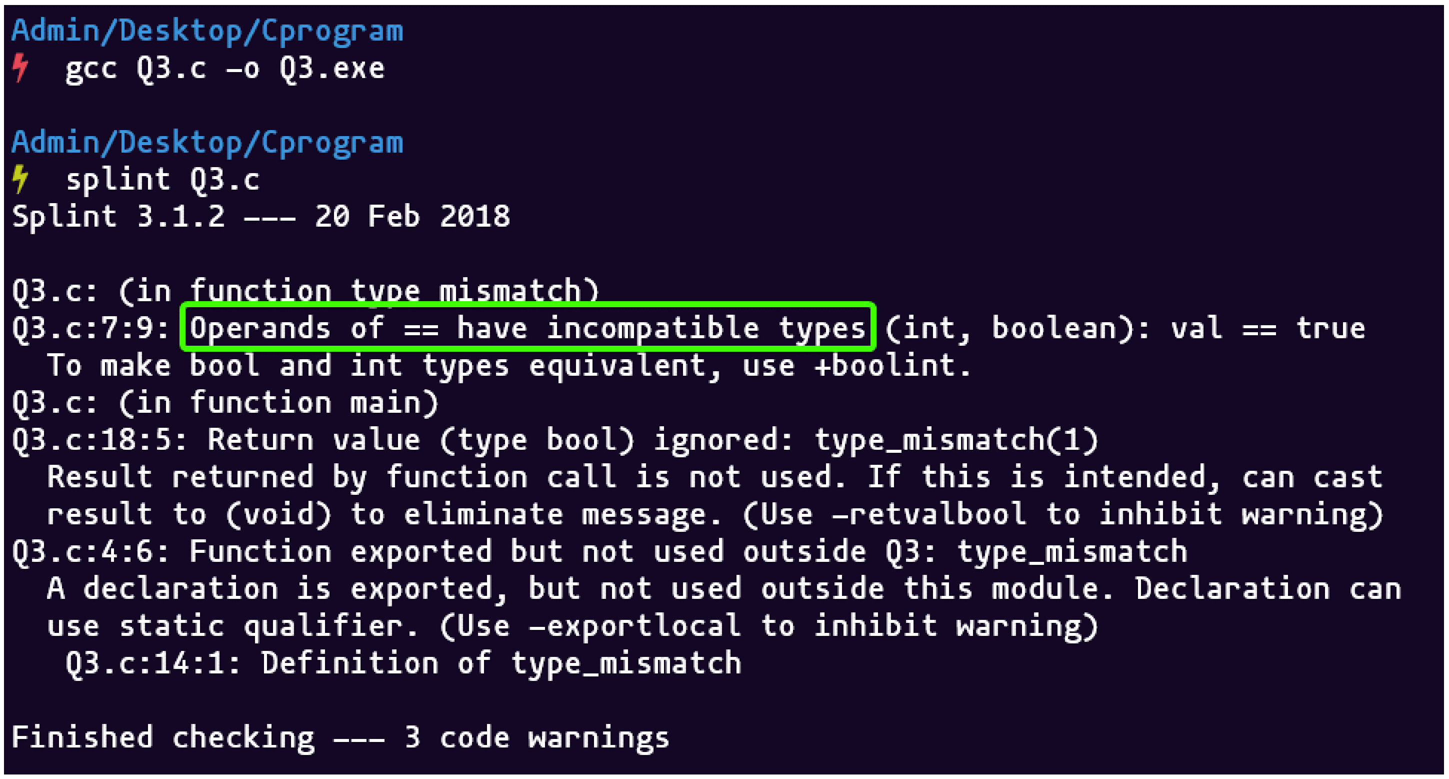
```
 splint Q2.c
Splint 3.1.2 --- 20 Feb 2018

Q2.c: (in function get_value)
Q2.c:5:20: Parameter x not used
  A function parameter is not used in the body of the function. If the argument
  is needed for type compatibility or future plans, use /*@unused@*/ in the
  argument declaration. (Use -paramuse to inhibit warning)
Q2.c: (in function compute)
Q2.c:13:16: 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)
Q2.c:15:26: Passed storage x not completely defined (*x is undefined):
               get_value (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)
Q2.c: (in function main)
Q2.c:28:9: Variable ans declared but not used
  A variable is declared but never used. Use /*@unused@*/ in front of
  declaration to suppress message. (Use -varuse to inhibit warning)
Q2.c:3:12: Function exported but not used outside Q2: get_value
  A declaration is exported, but not used outside this module. Declaration can
  use static qualifier. (Use -exportlocal to inhibit warning)
   Q2.c:8:1: Definition of get_value
Q2.c:10:5: Function exported but not used outside Q2: compute
   Q2.c:21:1: Definition of compute

Finished checking --- 6 code warnings
```

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

Code

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

bool type_mismatch(int val)
{
    // int comparison with bool
    if (val == true)
    {
        return true;
    }
    // Instead of False, returned 0
    return 0;
}

int main()
{
    type_mismatch(1);
    return 0;
}
```

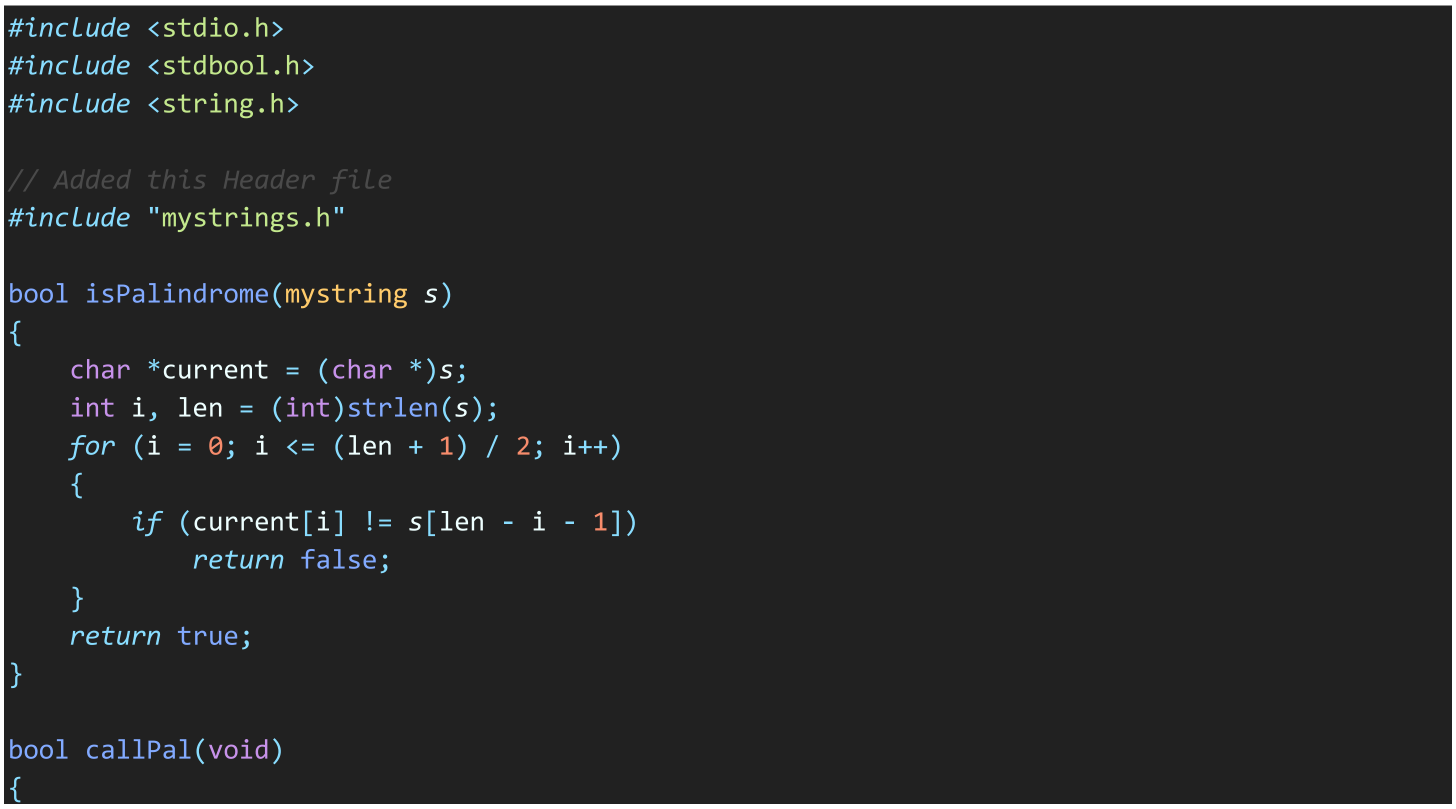
Output



```
Admin/Desktop/Cprogram
  gcc Q3.c -o Q3.exe

Admin/Desktop/Cprogram
  splint Q3.c
Splint 3.1.2 --- 20 Feb 2018

Q3.c: (in function type_mismatch)
Q3.c:7:9: Operands of == have incompatible types (int, boolean): val == true
  To make bool and int types equivalent, use +boolint.
Q3.c: (in function main)
Q3.c:18:5: Return value (type bool) ignored: type_mismatch(1)
  Result returned by function call is not used. If this is intended, can cast
  result to (void) to eliminate message. (Use -retvalbool to inhibit warning)
Q3.c:4:6: Function exported but not used outside Q3: type_mismatch
  A declaration is exported, but not used outside this module. Declaration can
  use static qualifier. (Use -exportlocal to inhibit warning)
   Q3.c:14:1: Definition of type_mismatch

Finished checking --- 3 code warnings
```

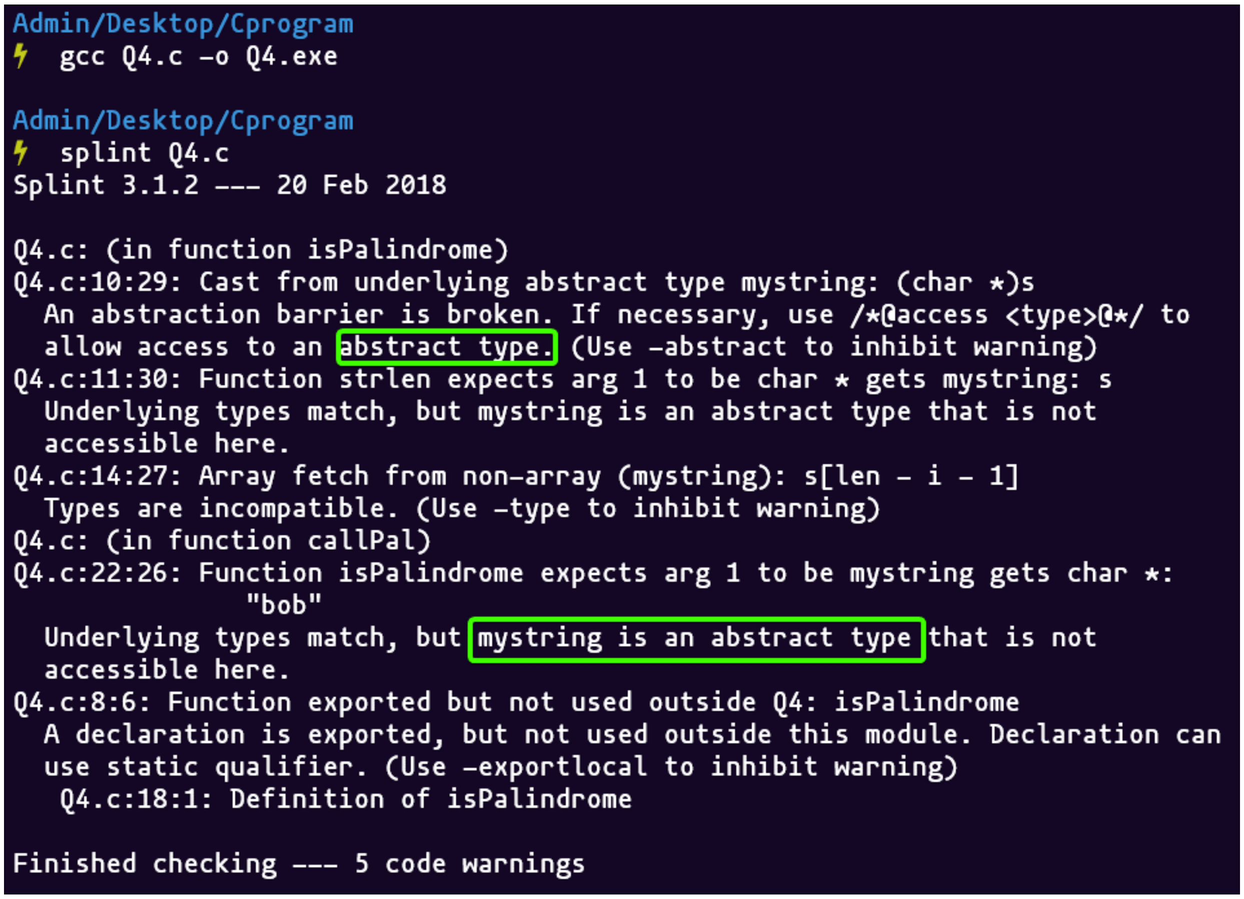
D.) Violations of Information Hiding.

Code



```c
#include <stdio.h>
#include <stdbool.h>
#include <string.h>

// Added this Header file
#include "mystrings.h"

bool isPalindrome(mystring 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"));
}

int main()
{
    printf("Checking Information Abstractions\n");
    // callPal();
    return 0;
}
```

Output



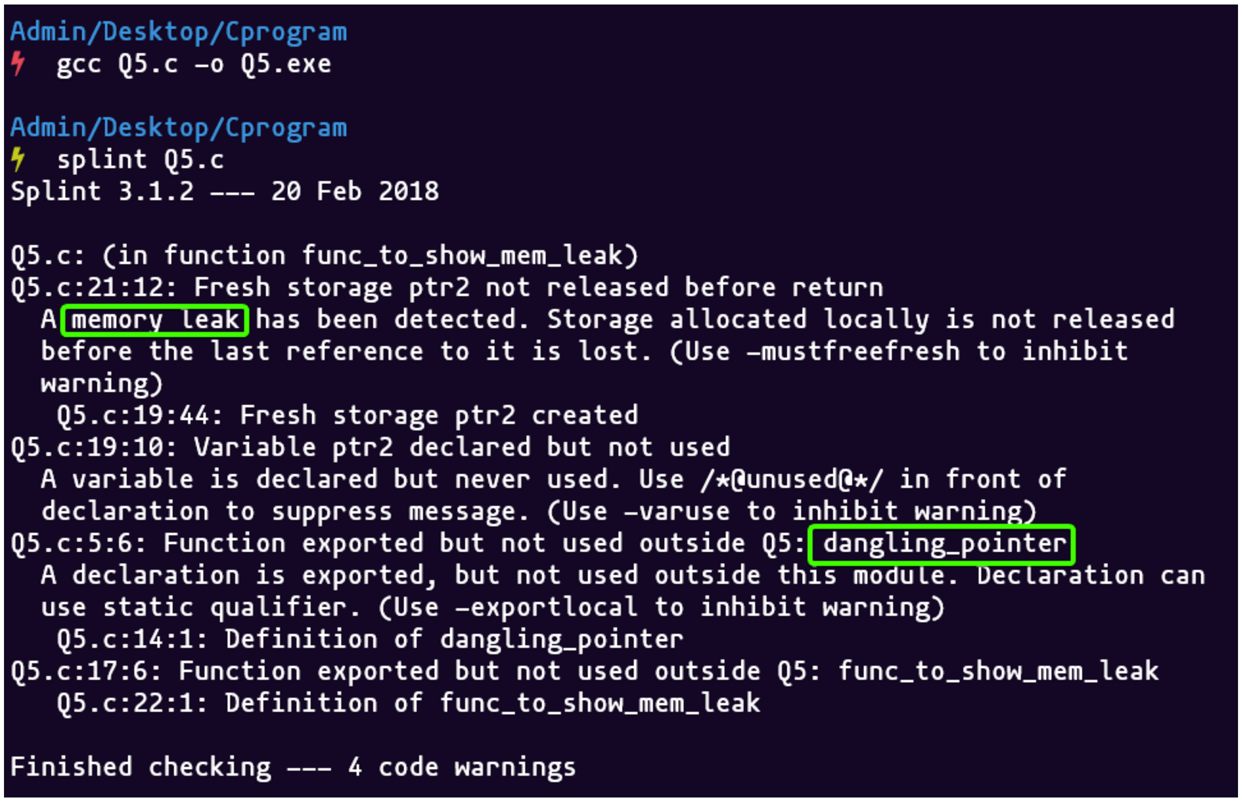
```
Admin/Desktop/Cprogram
ϟ  gcc Q4.c -o Q4.exe

Admin/Desktop/Cprogram
ϟ  splint Q4.c
Splint 3.1.2 --- 20 Feb 2018

Q4.c: (in function isPalindrome)
Q4.c:10:29: Cast from underlying abstract type mystring: (char *)s
  An abstraction barrier is broken. If necessary, use /*@access <type>@*/ to
  allow access to an abstract type. (Use -abstract to inhibit warning)
Q4.c:11:30: Function strlen expects arg 1 to be char * gets mystring: s
  Underlying types match, but mystring is an abstract type that is not
  accessible here.
Q4.c:14:27: Array fetch from non-array (mystring): s[len - i - 1]
  Types are incompatible. (Use -type to inhibit warning)
Q4.c: (in function callPal)
Q4.c:22:26: Function isPalindrome expects arg 1 to be mystring gets char *:
               "bob"
  Underlying types match, but mystring is an abstract type that is not
  accessible here.
Q4.c:8:6: Function exported but not used outside Q4: isPalindrome
  A declaration is exported, but not used outside this module. Declaration can
  use static qualifier. (Use -exportlocal to inhibit warning)
   Q4.c:18:1: Definition of isPalindrome

Finished checking --- 5 code warnings
```

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

Code

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

// Deallocating a memory pointed by ptr causes dangling pointer
void dangling_pointer()
{
    int *ptr = (int *)malloc(sizeof(int));
    // After below free call, ptr becomes a dangling pointer
    free(ptr);

    // No more a dangling pointer
    ptr = NULL;
}

// F(x) with memory leak
void func_to_show_mem_leak()
{
    int *ptr2 = (int *)malloc(sizeof(int));
    // return without deallocating ptr2
    return;
}

int main()
{
    dangling_pointer();

    func_to_show_mem_leak();

    return 0;
}
```

<u>Output</u>



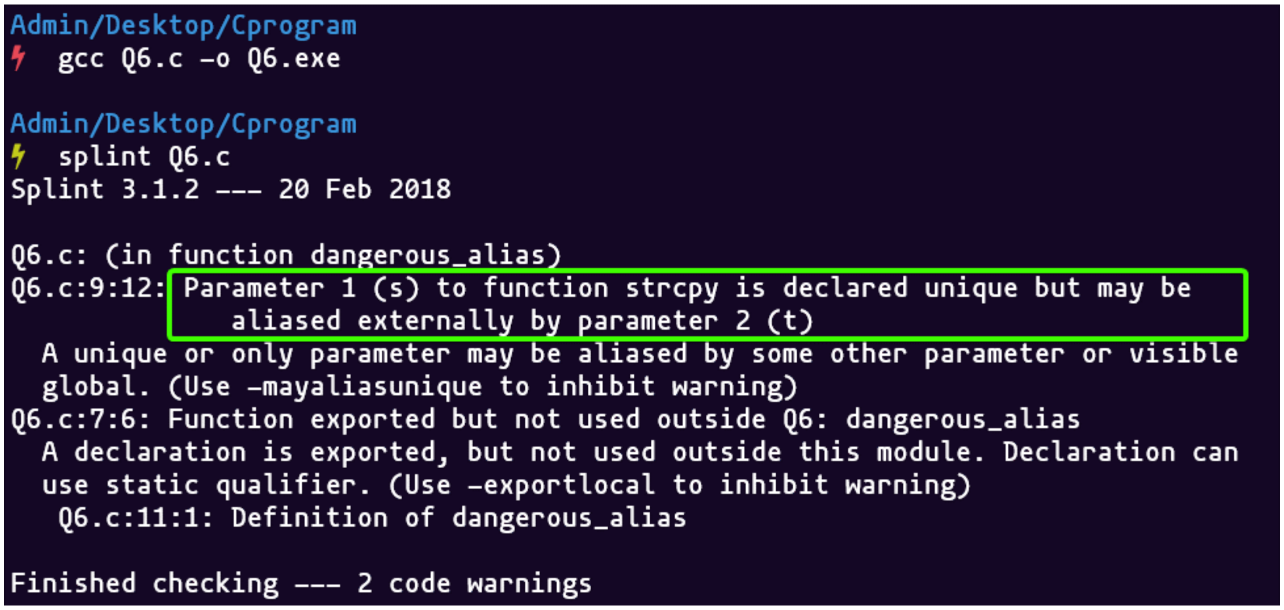
```
Admin/Desktop/Cprogram
ϟ  gcc Q5.c -o Q5.exe

Admin/Desktop/Cprogram
ϟ  splint Q5.c
Splint 3.1.2 --- 20 Feb 2018

Q5.c: (in function func_to_show_mem_leak)
Q5.c:21:12: Fresh storage ptr2 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)
   Q5.c:19:44: Fresh storage ptr2 created
Q5.c:19:10: Variable ptr2 declared but not used
  A variable is declared but never used. Use /*@unused@*/ in front of
  declaration to suppress message. (Use -varuse to inhibit warning)
Q5.c:5:6: Function exported but not used outside Q5: dangling_pointer
  A declaration is exported, but not used outside this module. Declaration can
  use static qualifier. (Use -exportlocal to inhibit warning)
   Q5.c:14:1: Definition of dangling_pointer
Q5.c:17:6: Function exported but not used outside Q5: func_to_show_mem_leak
   Q5.c:22:1: Definition of func_to_show_mem_leak

Finished checking --- 4 code warnings
```

F.) Dangerous Aliasing.

Code

```c
#include <stdio.h>
#include <ctype.h>
#include <string.h>

// Aliasing refers to the situation where the same memory location can be accessed using
different names.

void dangerous_alias(char *s, char *t)
{
    // Copying t to s
    strcpy(s, t);
    *s = toupper(*s);
}

int main()
{
    char *x = "hello";
    char *y = "world";

    dangerous_alias(x, y);

    printf("x = %c", *x);
    printf("y = %c", *y);

    return 0;
}
```

Output



```
Admin/Desktop/Cprogram
ϟ  gcc Q6.c -o Q6.exe

Admin/Desktop/Cprogram
ϟ  splint Q6.c
Splint 3.1.2 --- 20 Feb 2018

Q6.c: (in function dangerous_alias)
Q6.c:9:12: 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)
Q6.c:7:6: Function exported but not used outside Q6: dangerous_alias
  A declaration is exported, but not used outside this module. Declaration can
  use static qualifier. (Use -exportlocal to inhibit warning)
   Q6.c:11:1: Definition of dangerous_alias

Finished checking --- 2 code warnings
```