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**

*#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;

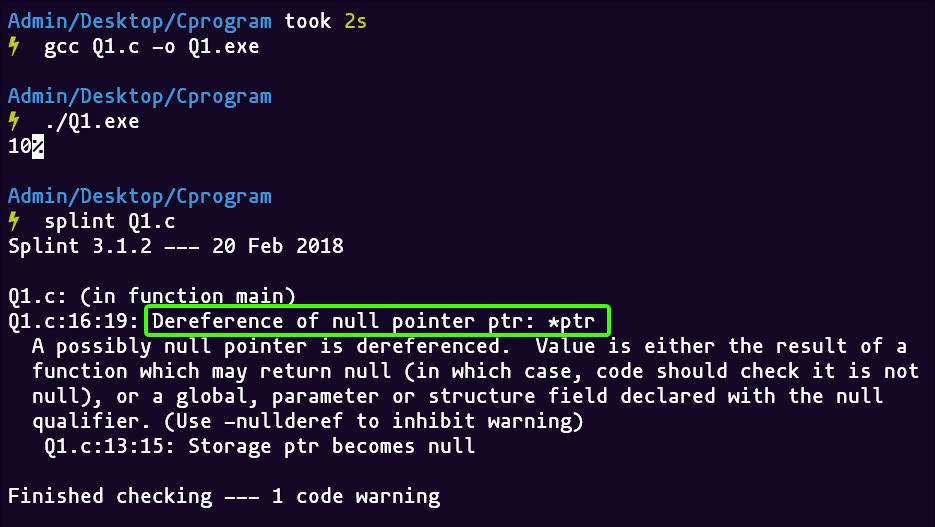
*// ! Dereferencing a null pointer always results in undefined behavior and can cause crashes.*

    printf("%d", \*ptr);

*return* 0;

}

**Output**



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

**Code**

*#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;

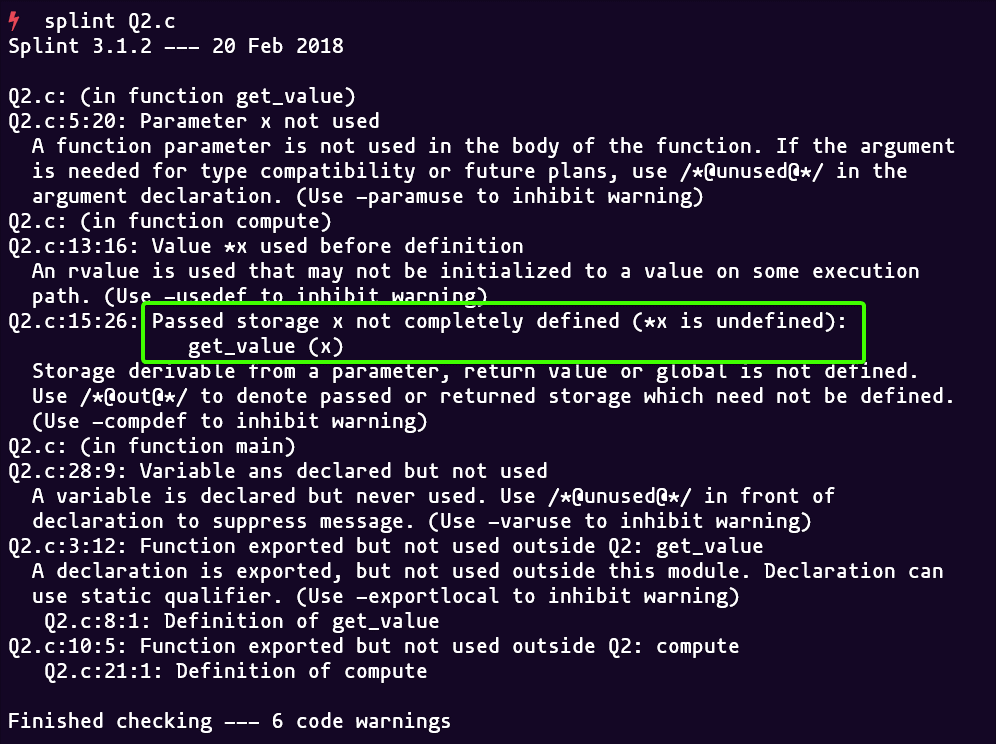
    int \*p = &k;

    int ans = compute(p, k);

*return* 0;

}

**Output**



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

**Code**

*#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()

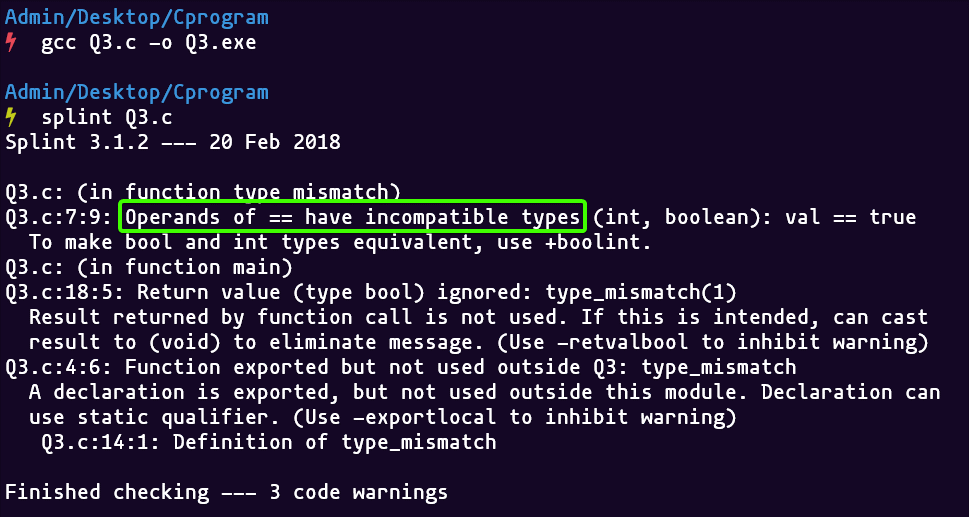
{

    type\_mismatch(1);

*return* 0;

}

**Output**



D.) Violations of Information Hiding.

**Code**

*#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()

{

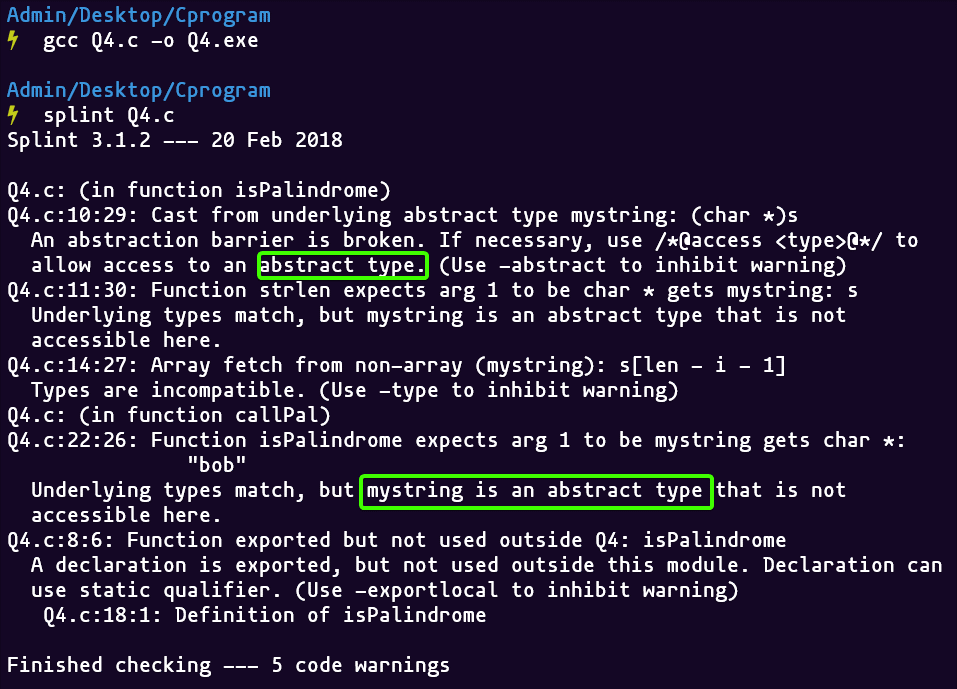
    printf("Checking Information Abstractions\n");

*// callPal();*

*return* 0;

}

**Output**



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

**Code**

*#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()

{

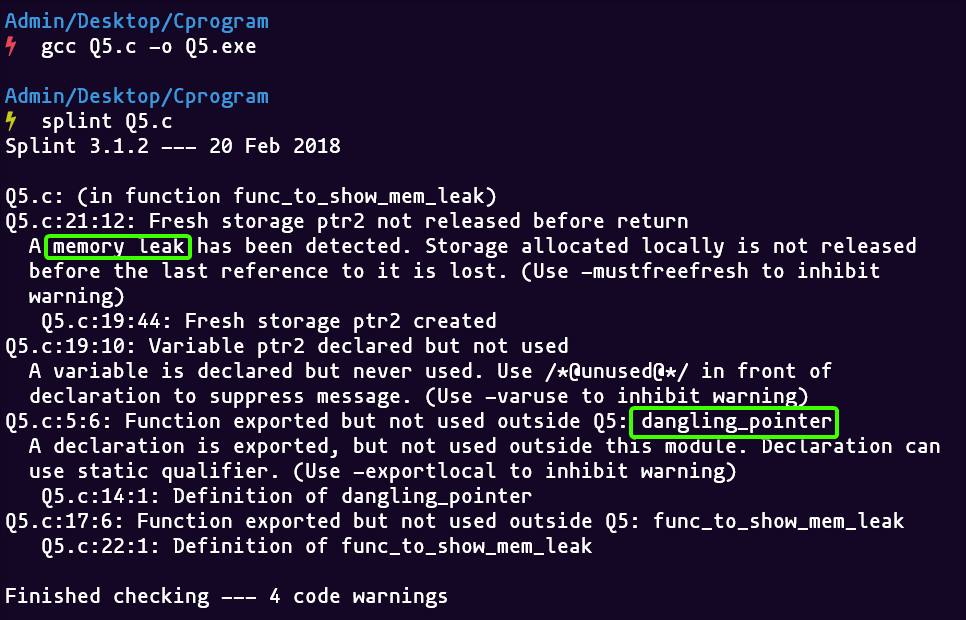
    dangling\_pointer();

    func\_to\_show\_mem\_leak();

*return* 0;

}

**Output**



F.) Dangerous Aliasing.

**Code**

*#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**

