U18CO018

Shekhaliya Shubham

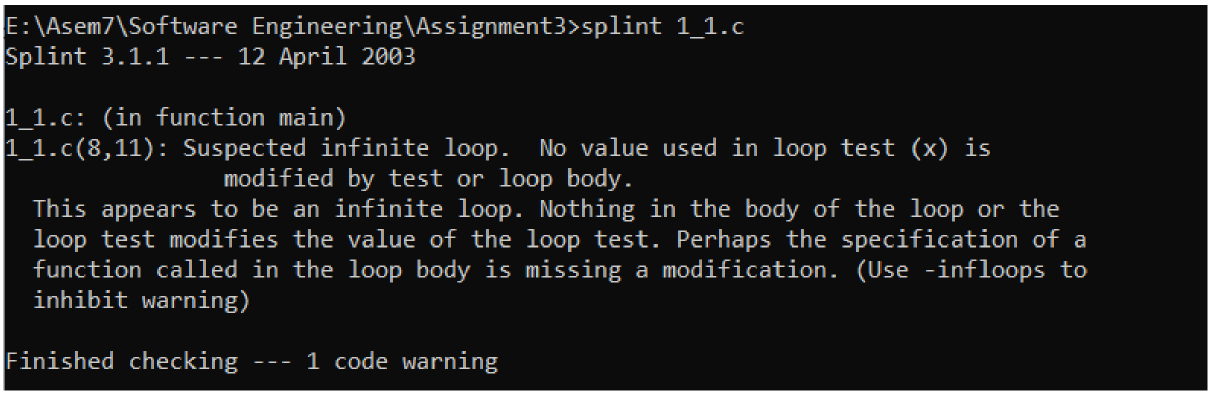
Software Engineering

Lab Assignment-3

1. Implement the following problematic control structures in C and compare the outputs of standard C compiler and the Splint tool.

- Likely infinite loops

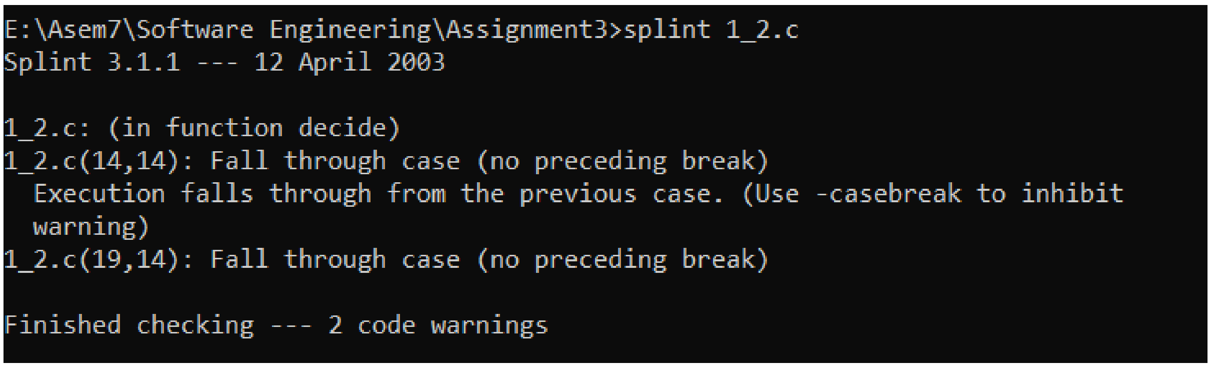
```c
#include<stdio.h>

int main() {

    int x = 10;
    int f = 0;

    while(x > 0) {
        f = 1-f;
    }

    return 0;
}
```



```
E:\Asem7\Software Engineering\Assignment3>splint 1_1.c
Splint 3.1.1 --- 12 April 2003

1_1.c: (in function main)
1_1.c(8,11): Suspected infinite loop.  No value used in loop test (x) is
                modified by test or loop body.
  This appears to be an infinite loop. Nothing in the body of the loop or the
  loop test modifies the value of the loop test. Perhaps the specification of a
  function called in the loop body is missing a modification. (Use -infloops to
  inhibit warning)

Finished checking --- 1 code warning
```

- Fall through switch cases

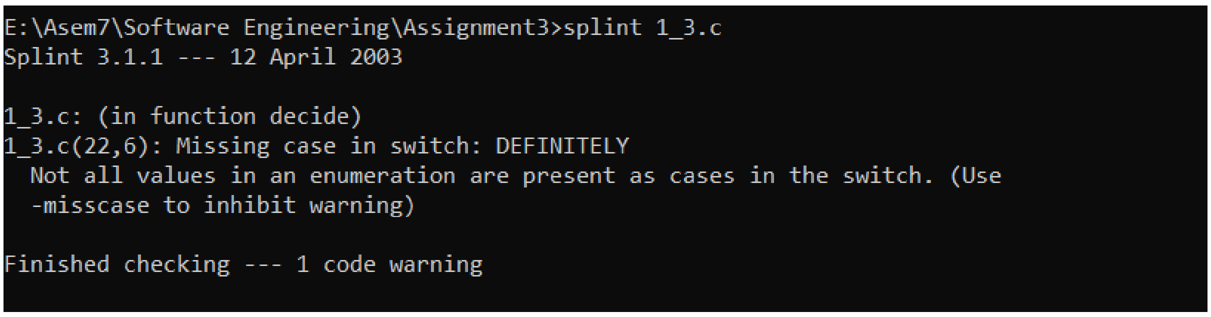
```c
typedef enum {
    YES,
    NO,
    DEFINITELY,
    PROBABLY,
    MAYBE
} ynm;

void decide(ynm y) {
    switch (y) {
        case PROBABLY:
        case NO:
            printf("No!");
        case MAYBE:
            printf("Maybe");
        /*@fallthrough@*/
        case YES:
            printf("Yes!");
        case DEFINITELY:
            printf("Definitely!");
    }
}
```



```
E:\Asem7\Software Engineering\Assignment3>splint 1_2.c
Splint 3.1.1 --- 12 April 2003

1_2.c: (in function decide)
1_2.c(14,14): Fall through case (no preceding break)
  Execution falls through from the previous case. (Use -casebreak to inhibit
  warning)
1_2.c(19,14): Fall through case (no preceding break)

Finished checking --- 2 code warnings
```

- Missing switch cases

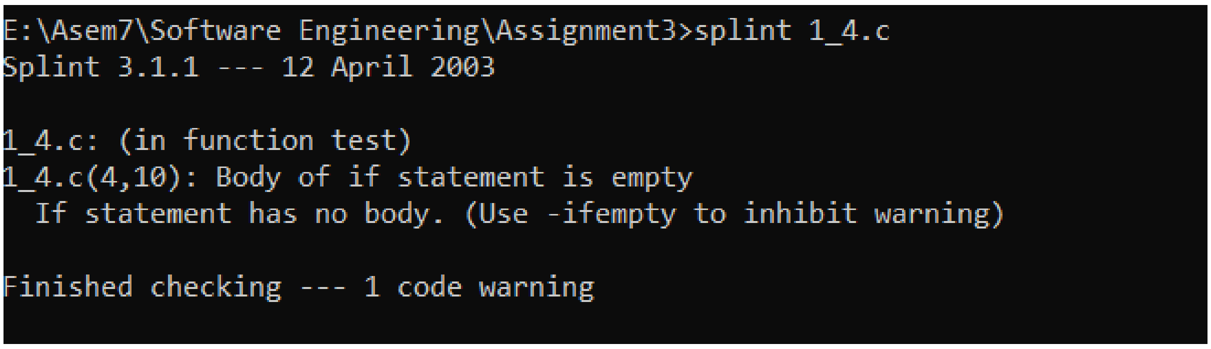
```c
typedef enum {
    YES,
    NO,
    DEFINITELY,
    PROBABLY,
    MAYBE
} ynm;

void decide(ynm y) {
    switch (y) {
        case PROBABLY:
            break;
        case NO:
            printf("No!");
            break;
        case MAYBE:
            printf("Maybe");
            break;
        case YES:
            printf("Yes!");
            break;
    }
}
```



```
E:\Asem7\Software Engineering\Assignment3>splint 1_3.c
Splint 3.1.1 --- 12 April 2003

1_3.c: (in function decide)
1_3.c(22,6): Missing case in switch: DEFINITELY
  Not all values in an enumeration are present as cases in the switch. (Use
  -misscase to inhibit warning)

Finished checking --- 1 code warning
```

- Empty statement after an if, while or for

```c
void test() {
    int x = 0;
    if (x > 3)
        ;
    if (x > 3)
        x++;
}
```

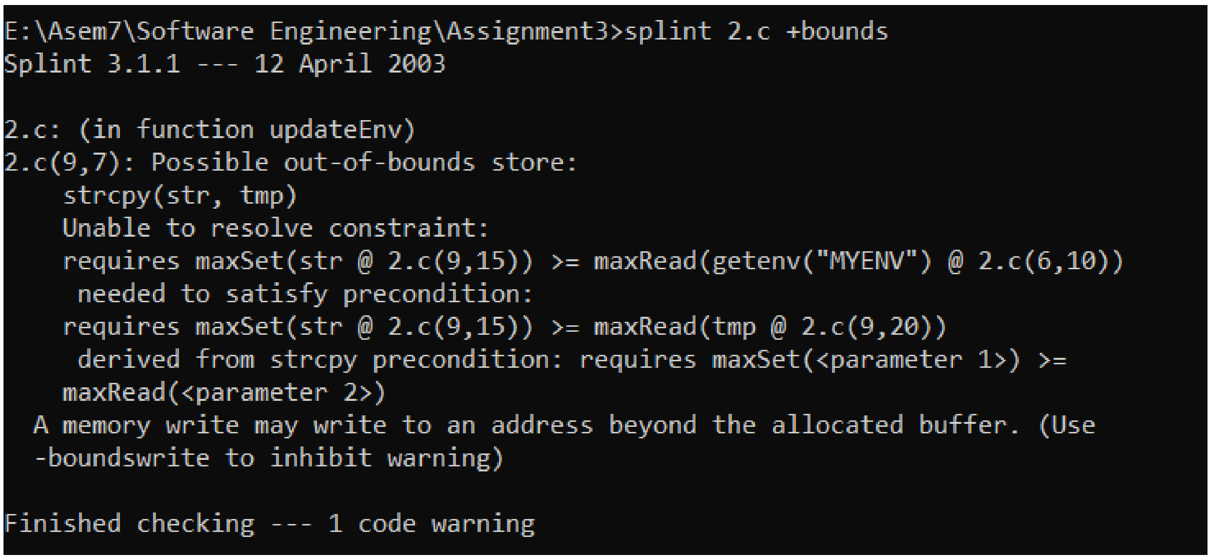


```
E:\Asem7\Software Engineering\Assignment3>splint 1_4.c
Splint 3.1.1 --- 12 April 2003

1_4.c: (in function test)
1_4.c(4,10): Body of if statement is empty
  If statement has no body. (Use -ifempty to inhibit warning)

Finished checking --- 1 code warning
```

2. What is buffer overflow? How it can be exploited? Write a C program to illustrate a buffer overflow attack?

Buffers are memory storage regions that temporarily hold data while it is being transferred from one location to another. A buffer overflow (or buffer overrun) occurs when the volume of data exceeds the storage capacity of the memory buffer. As a result, the program attempting to write the data to the buffer overwrites adjacent memory locations

```c
#include <stdio.h>

void updateEnv(char * str)
{
   char * tmp;
   tmp = getenv("MYENV");

   if (tmp != NULL)
      strcpy (str, tmp);
}

void updateEnvSafe (char * str, size_t strSize) /*@requires maxSet(str) >= str
Size@*/
{
   char * tmp;
   tmp = getenv("MYENV");

   if (tmp != NULL)
   {
      strncpy (str, tmp, strSize -1);
      str[strSize -1] = '/0';
   }
}
```

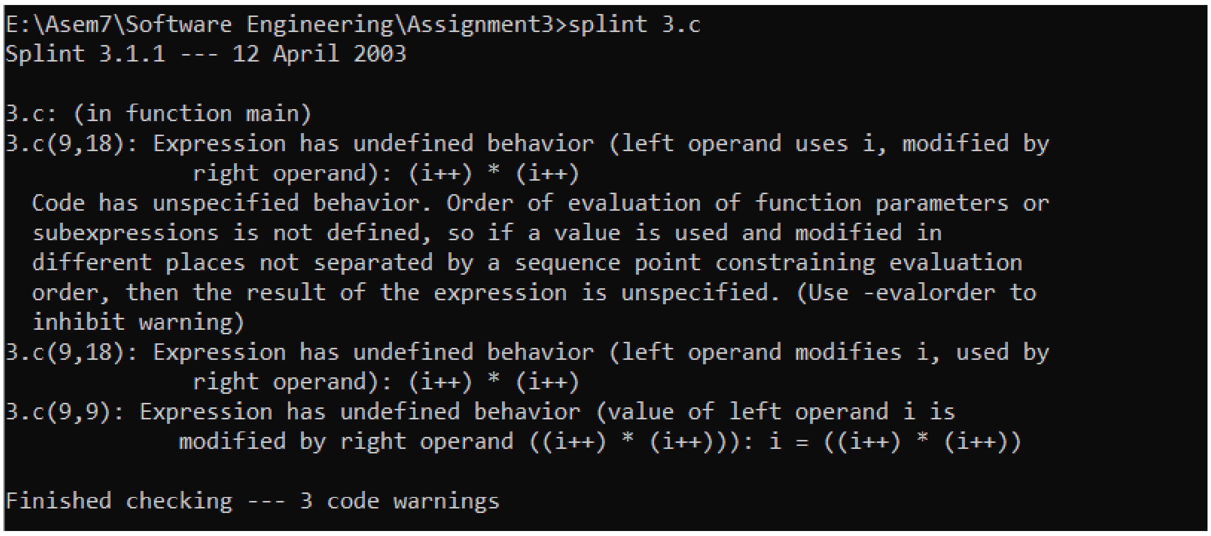


```
E:\Asem7\Software Engineering\Assignment3>splint 2.c +bounds
Splint 3.1.1 --- 12 April 2003

2.c: (in function updateEnv)
2.c(9,7): Possible out-of-bounds store:
    strcpy(str, tmp)
    Unable to resolve constraint:
    requires maxSet(str @ 2.c(9,15)) >= maxRead(getenv("MYENV") @ 2.c(6,10))
     needed to satisfy precondition:
    requires maxSet(str @ 2.c(9,15)) >= maxRead(tmp @ 2.c(9,20))
     derived from strcpy precondition: requires maxSet(<parameter 1>) >=
    maxRead(<parameter 2>)
  A memory write may write to an address beyond the allocated buffer. (Use
  -boundswrite to inhibit warning)

Finished checking --- 1 code warning
```

The above program shows two ways of updating the environment variables. The first function doesn’t update it safely and can lead to buffer overflow attack whereas the second function is the correct method to update the environment variable and doesn’t cause any buffer overflow attack.

3. Macro implementations or invocations can be dangerous. Justify this statement by giving an example in C language.

```c
#include <stdio.h>

int square(/*@sef@*/ int x);
#define square(x) ((x) * (x))

int main()
{
    int i = 1;
    i = square(i++);
    i = square(i);

    return 0;
}
```



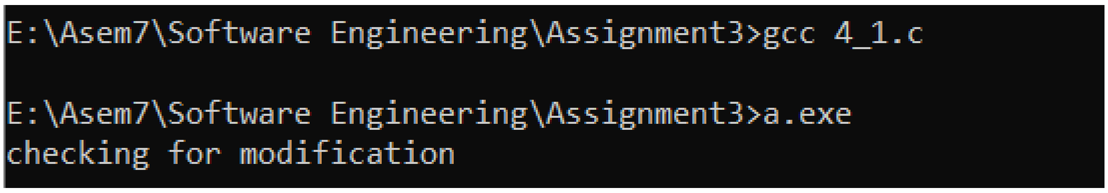
```
E:\Asem7\Software Engineering\Assignment3>splint 3.c
Splint 3.1.1 --- 12 April 2003

3.c: (in function main)
3.c(9,18): Expression has undefined behavior (left operand uses i, modified by
              right operand): (i++) * (i++)
  Code has unspecified behavior. Order of evaluation of function parameters or
  subexpressions is not defined, so if a value is used and modified in
  different places not separated by a sequence point constraining evaluation
  order, then the result of the expression is unspecified. (Use -evalorder to
  inhibit warning)
3.c(9,18): Expression has undefined behavior (left operand modifies i, used by
              right operand): (i++) * (i++)
3.c(9,9): Expression has undefined behavior (value of left operand i is
             modified by right operand ((i++) * (i++))): i = ((i++) * (i++))

Finished checking --- 3 code warnings
```

4. What do you mean by interface faults. Write a set of C programs to implement interface faults and perform their detection using Splint tool. Check whether they are detected by the standard C compiler or not.

Demo 1: Modification

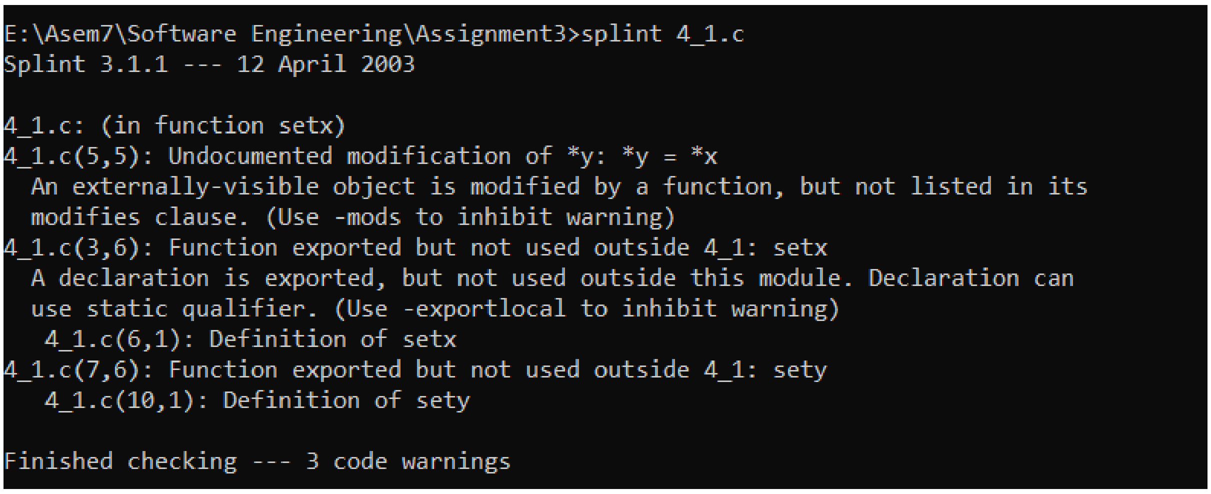
```
#include <stdio.h>

void setx(int *x, int *y) /*@modifies *x@*/
{
    *y = *x;
}
void sety(int *x, int *y) /*@modifies *y@*/
{
    setx(y, x);
}

int main()
{

    int x = 5;
    int y = 3;

    printf("checking for modification\n");

    sety(&y,&x);

    return 0;
}
```

GCC ouput



```
E:\Asem7\Software Engineering\Assignment3>gcc 4_1.c

E:\Asem7\Software Engineering\Assignment3>a.exe
checking for modification
```

Splint output

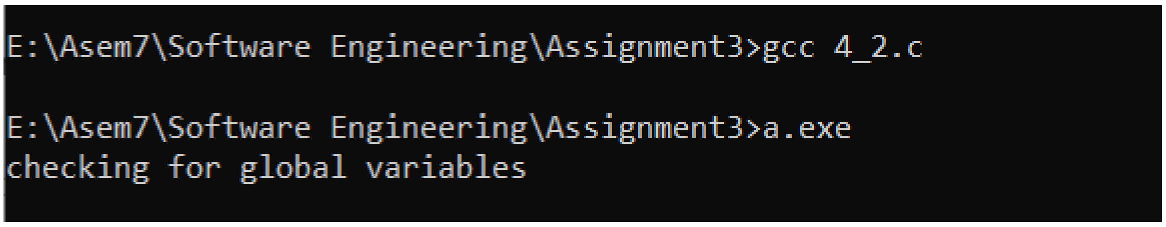


```
E:\Asem7\Software Engineering\Assignment3>splint 4_1.c
Splint 3.1.1 --- 12 April 2003

4_1.c: (in function setx)
4_1.c(5,5): Undocumented modification of *y: *y = *x
  An externally-visible object is modified by a function, but not listed in its
  modifies clause. (Use -mods to inhibit warning)
4_1.c(3,6): Function exported but not used outside 4_1: setx
  A declaration is exported, but not used outside this module. Declaration can
  use static qualifier. (Use -exportlocal to inhibit warning)
   4_1.c(6,1): Definition of setx
4_1.c(7,6): Function exported but not used outside 4_1: sety
   4_1.c(10,1): Definition of sety

Finished checking --- 3 code warnings
```

Demo 2: Global variables

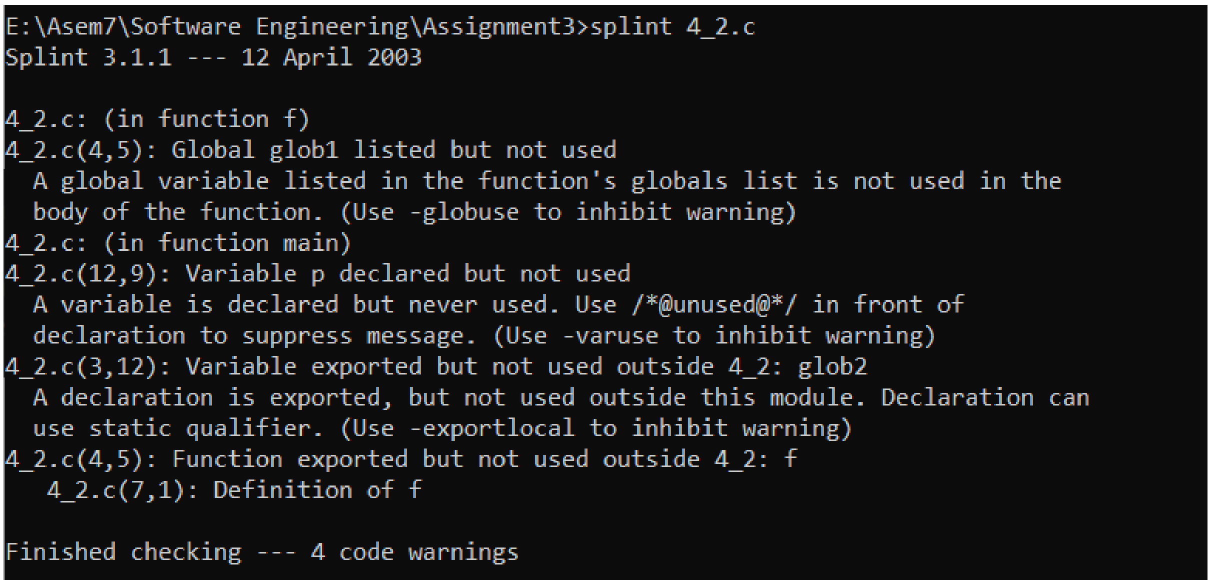
```c
#include <stdio.h>

int glob1, glob2;
int f(void) /*@globals glob1;@*/
{
    return glob2;
}

int main()
{

    int p = f();

    printf("checking for global variables\n");
    return 0;
}
```

GCC ouput



```
E:\Asem7\Software Engineering\Assignment3>gcc 4_2.c

E:\Asem7\Software Engineering\Assignment3>a.exe
checking for global variables
```

Splint output



```
E:\Asem7\Software Engineering\Assignment3>splint 4_2.c
Splint 3.1.1 --- 12 April 2003

4_2.c: (in function f)
4_2.c(4,5): Global glob1 listed but not used
  A global variable listed in the function's globals list is not used in the
  body of the function. (Use -globuse to inhibit warning)
4_2.c: (in function main)
4_2.c(12,9): Variable p 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)
4_2.c(3,12): Variable exported but not used outside 4_2: glob2
  A declaration is exported, but not used outside this module. Declaration can
  use static qualifier. (Use -exportlocal to inhibit warning)
4_2.c(4,5): Function exported but not used outside 4_2: f
   4_2.c(7,1): Definition of f

Finished checking --- 4 code warnings
```