Software Engineering

Assignment 3

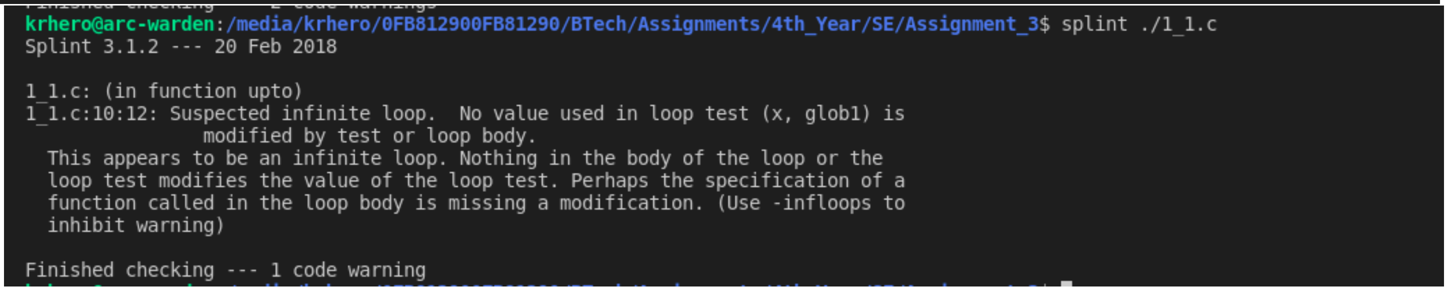
Student Details

Name: Krunal Rank
Adm. No: U18CO081

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

Likely infinite loops

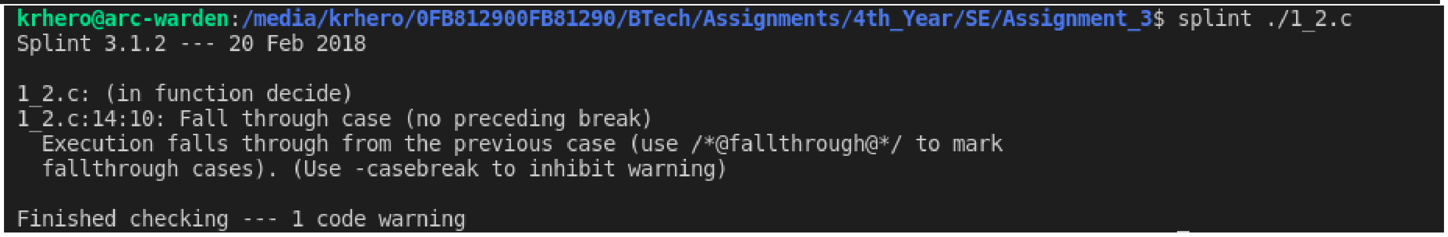
```c
#include <stdio.h>

extern int glob1, glob2;
extern int f(void) /*@globals glob1@*/ /*@modifies nothing@*/;
extern void g(void) /*@modifies glob2@*/;
extern void h(void);

void upto(int x)
{
   while (x > f())
       g();
   while (f() < 3)
       h();
}

int main()
{
   printf("Checking for likely infinite loops\n");
   return 0;
}
```



```
krhero@arc-warden:/media/krhero/0FB812900FB81290/BTech/Assignments/4th_Year/SE/Assignment_3$ splint ./1_1.c
Splint 3.1.2 --- 20 Feb 2018

1_1.c: (in function upto)
1_1.c:10:12: Suspected infinite loop.  No value used in loop test (x, glob1) 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
#include <stdio.h>

typedef enum { TYPE1, TYPE2, TYPE3, TYPE4, TYPE5 } ynm;

void decide(ynm y)
{
   switch (y)
   {
   /*@fallthrough@*/
   case TYPE1:
       break;
   case TYPE2:
       printf("Type2!");
   case TYPE3:
       printf("Type3");
       break;
   case TYPE4:
   case TYPE5:
       printf("Type4");
       break;
   }
}

int main()
{
   printf("Checking fall through switch cases");
   return 0;
}
```

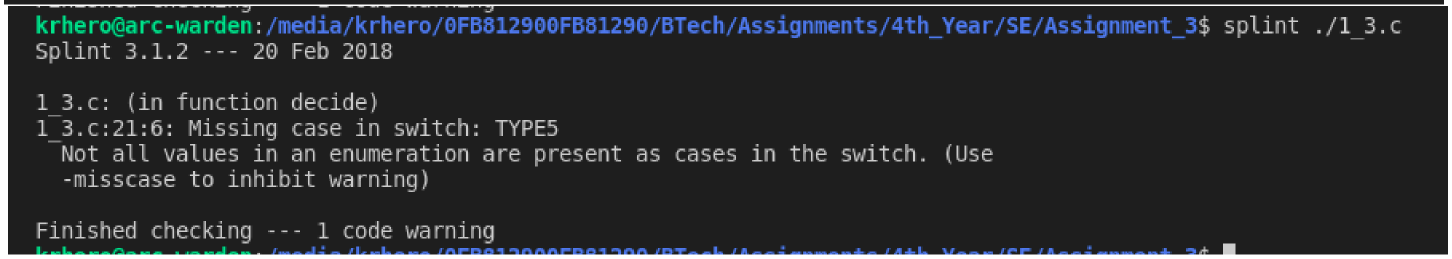


```
krhero@arc-warden:/media/krhero/0FB812900FB81290/BTech/Assignments/4th_Year/SE/Assignment_3$ splint ./1_2.c
Splint 3.1.2 --- 20 Feb 2018

1_2.c: (in function decide)
1_2.c:14:10: Fall through case (no preceding break)
  Execution falls through from the previous case (use /*@fallthrough@*/ to mark
  fallthrough cases). (Use -casebreak to inhibit warning)

Finished checking --- 1 code warning
```

Missing switch cases

```c
#include <stdio.h>

typedef enum { TYPE1, TYPE2, TYPE3, TYPE4, TYPE5 } ynm;
```

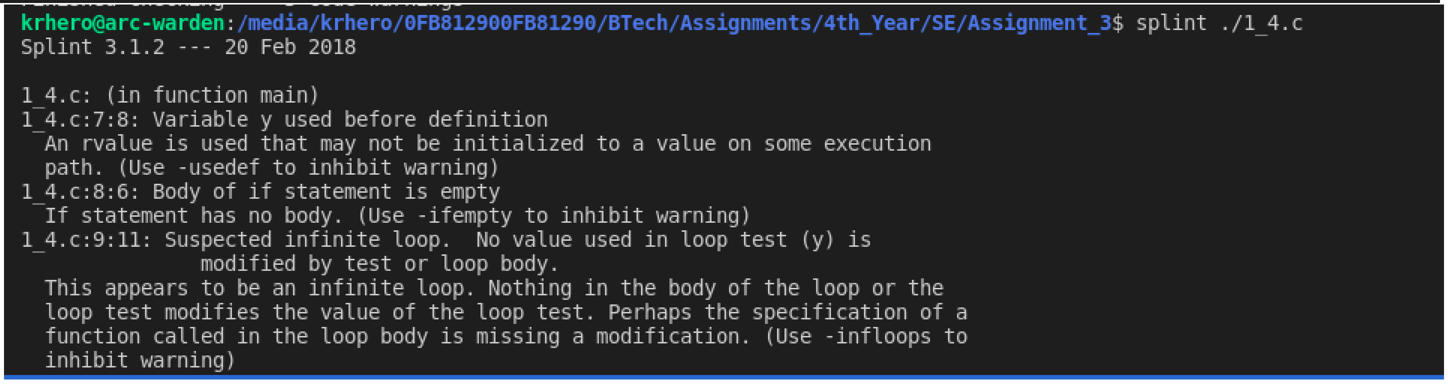
```
void decide(ynm y)
{
   switch (y)
   {
   /*@fallthrough@*/
   case TYPE1:
       break;
   case TYPE2:
       printf("Type2!");
       break;
   case TYPE3:
       printf("Type3");
       break;
   case TYPE4:
       printf("Type4");
       break;
   }
}

int main()
{
   printf("Checking fall through missing switch cases");
   return 0;
}
```



```
krhero@arc-warden:/media/krhero/0FB812900FB81290/BTech/Assignments/4th_Year/SE/Assignment_3$ splint ./1_3.c
Splint 3.1.2 --- 20 Feb 2018

1_3.c: (in function decide)
1_3.c:21:6: Missing case in switch: TYPE5
  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

```
#include <stdio.h>


int main()
{
   int y;
   if(y>0)
   ;
   while(y>0);
   printf("Checking empty if, while or do while");
   return 0;
}
```

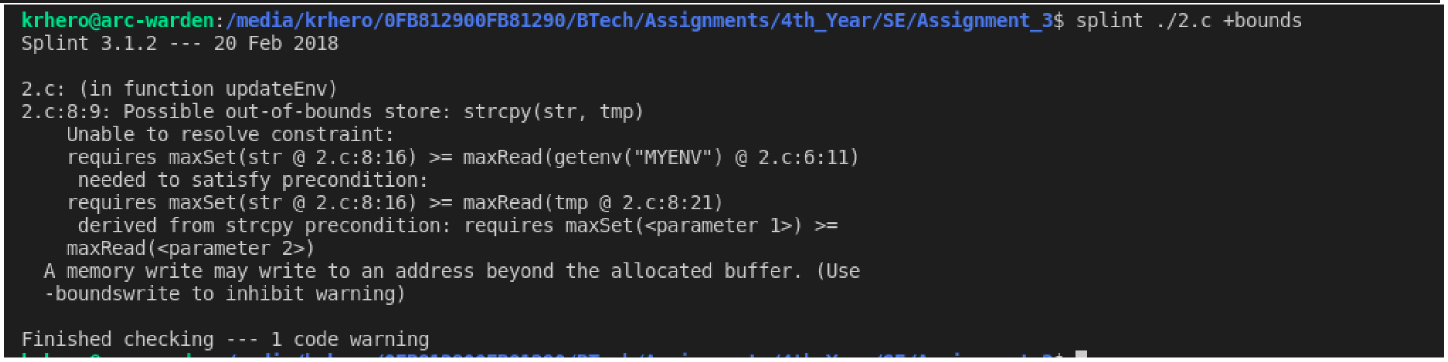


```
krhero@arc-warden:/media/krhero/0FB812900FB81290/BTech/Assignments/4th_Year/SE/Assignment_3$ splint ./1_4.c
Splint 3.1.2 --- 20 Feb 2018

1_4.c: (in function main)
1_4.c:7:8: Variable y used before definition
  An rvalue is used that may not be initialized to a value on some execution
  path. (Use -usedef to inhibit warning)
1_4.c:8:6: Body of if statement is empty
  If statement has no body. (Use -ifempty to inhibit warning)
1_4.c:9:11: Suspected infinite loop.  No value used in loop test (y) 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)
```

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

```
#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) >= strSize @*/
{
   char *tmp;
   tmp = getenv("MYENV");
   if (tmp != NULL)
   {
       strncpy(str, tmp, strSize - 1);
       str[strSize - 1] = '/0';
   }
}

int main()
```

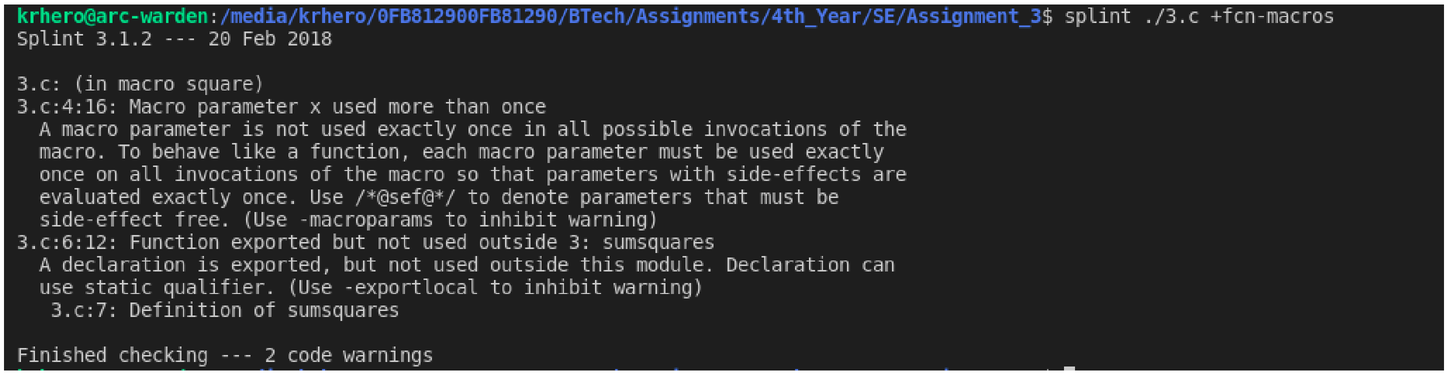
```c
{
   printf("Buffer Owerflow attack\n");
   return 0;
}
```



```
krhero@arc-warden:/media/krhero/0FB812900FB81290/BTech/Assignments/4th_Year/SE/Assignment_3$ splint ./2.c +bounds
Splint 3.1.2 --- 20 Feb 2018

2.c: (in function updateEnv)
2.c:8:9: Possible out-of-bounds store: strcpy(str, tmp)
    Unable to resolve constraint:
    requires maxSet(str @ 2.c:8:16) >= maxRead(getenv("MYENV") @ 2.c:6:11)
     needed to satisfy precondition:
    requires maxSet(str @ 2.c:8:16) >= maxRead(tmp @ 2.c:8:21)
     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
```

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

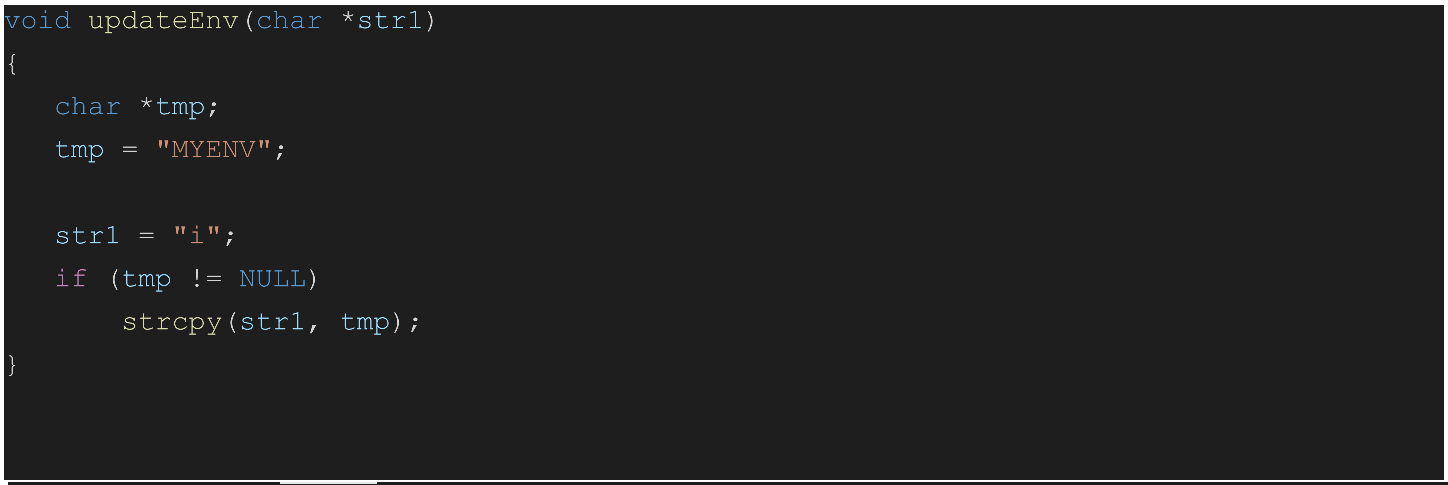
```c
#include <stdio.h>

extern int square( int x);
#define square(x) ((x) * (x))

extern int sumsquares(int x, int y);
#define sumsquares(x, y) (square(x) + square(y))

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

   printf("Checking Macro implementation and invocations\n");
   return 0;
}
```



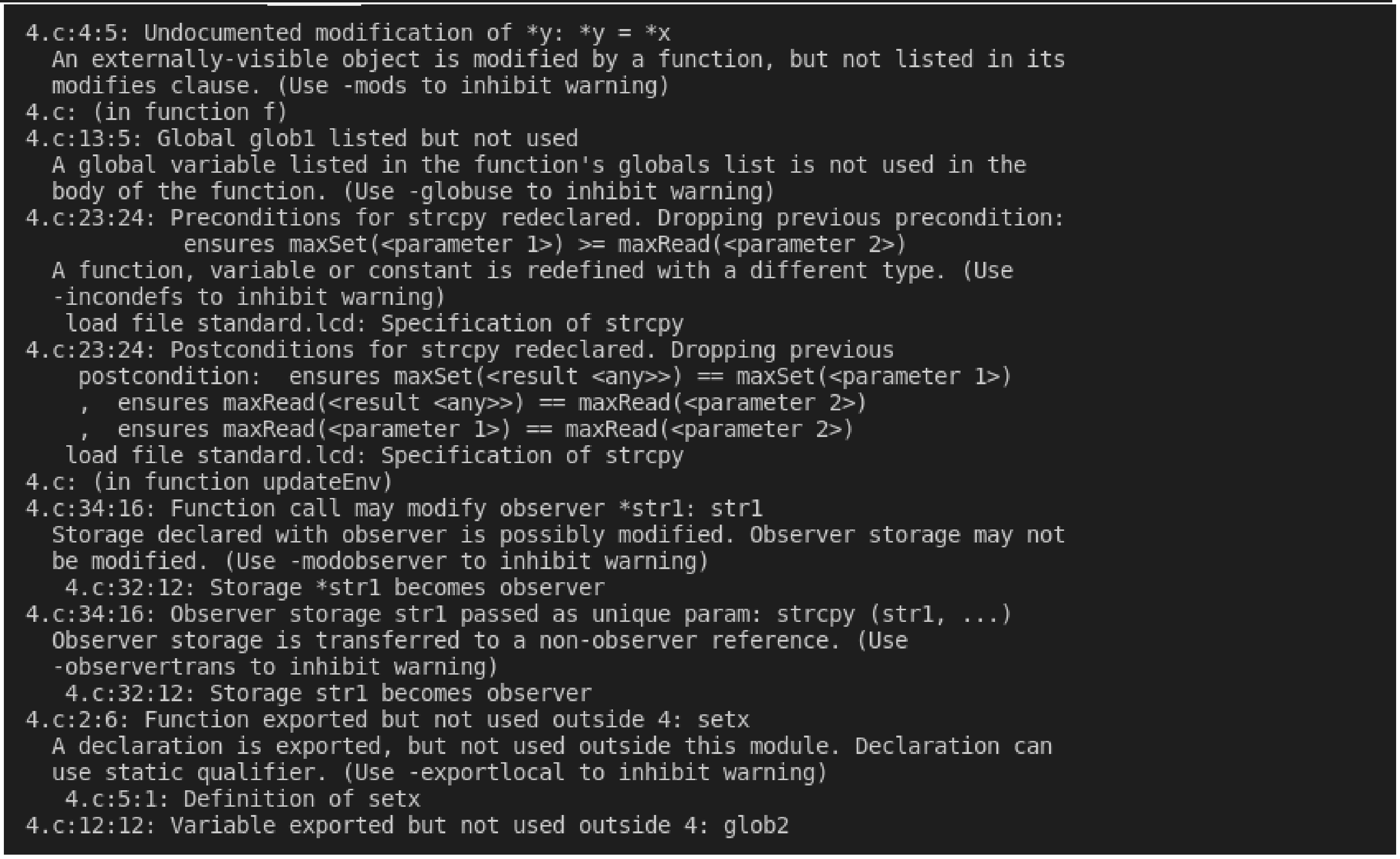
```
krhero@arc-warden:/media/krhero/0FB812900FB81290/BTech/Assignments/4th_Year/SE/Assignment_3$ splint ./3.c +fcn-macros
Splint 3.1.2 --- 20 Feb 2018

3.c: (in macro square)
3.c:4:16: Macro parameter x used more than once
  A macro parameter is not used exactly once in all possible invocations of the
  macro. To behave like a function, each macro parameter must be used exactly
  once on all invocations of the macro so that parameters with side-effects are
  evaluated exactly once. Use /*@sef@*/ to denote parameters that must be
  side-effect free. (Use -macroparams to inhibit warning)
3.c:6:12: Function exported but not used outside 3: sumsquares
  A declaration is exported, but not used outside this module. Declaration can
  use static qualifier. (Use -exportlocal to inhibit warning)
   3.c:7: Definition of sumsquares

Finished checking --- 2 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 the Splint tool. Check whether they are detected by the standard C compiler or not.

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

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




//Require and Ensure Clauses
#include <stdio.h>
#include <string.h>
void /*@alt char * @*/ strcpy(/*@unique@*/ /*@out@*/ /*@returned@*/ char *s1, char
*s2)
   /*@modifies *s1@*/
   /*@requires maxSet(s1) >=3 @*/
   /*@ensures maxRead(s1) >= maxRead (s2) @*/;
```



```c
void updateEnv(char *str1)
{
    char *tmp;
    tmp = "MYENV";

    str1 = "i";
    if (tmp != NULL)
        strcpy(str1, tmp);
}
```



```
4.c:4:5: 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.c: (in function f)
4.c:13:5: 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.c:23:24: Preconditions for strcpy redeclared. Dropping previous precondition:
            ensures maxSet(<parameter 1>) >= maxRead(<parameter 2>)
  A function, variable or constant is redefined with a different type. (Use
  -incondefs to inhibit warning)
   load file standard.lcd: Specification of strcpy
4.c:23:24: Postconditions for strcpy redeclared. Dropping previous
    postcondition:  ensures maxSet(<result <any>>) == maxSet(<parameter 1>)
    ,  ensures maxRead(<result <any>>) == maxRead(<parameter 2>)
    ,  ensures maxRead(<parameter 1>) == maxRead(<parameter 2>)
   load file standard.lcd: Specification of strcpy
4.c: (in function updateEnv)
4.c:34:16: Function call may modify observer *str1: str1
  Storage declared with observer is possibly modified. Observer storage may not
  be modified. (Use -modobserver to inhibit warning)
   4.c:32:12: Storage *str1 becomes observer
4.c:34:16: Observer storage str1 passed as unique param: strcpy (str1, ...)
  Observer storage is transferred to a non-observer reference. (Use
  -observertrans to inhibit warning)
   4.c:32:12: Storage str1 becomes observer
4.c:2:6: Function exported but not used outside 4: setx
  A declaration is exported, but not used outside this module. Declaration can
  use static qualifier. (Use -exportlocal to inhibit warning)
   4.c:5:1: Definition of setx
4.c:12:12: Variable exported but not used outside 4: glob2
```