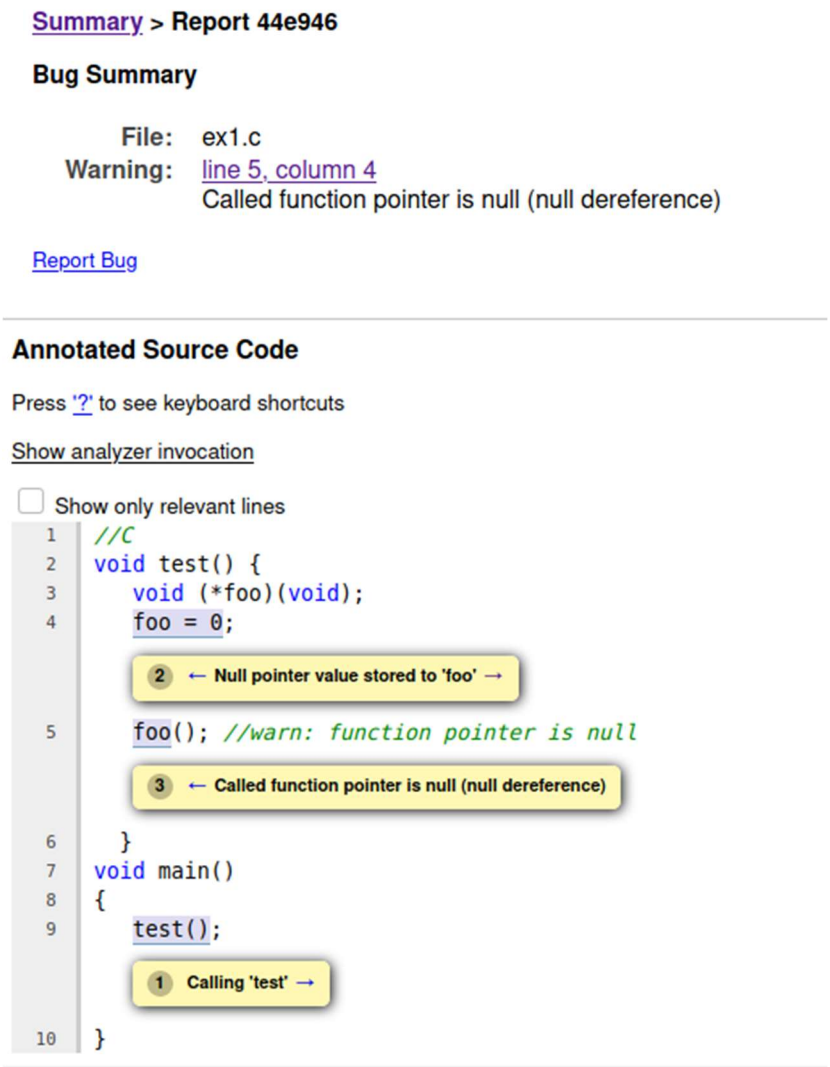
LAB 5: LLVM

1. DEFAULT CHECKERS:

a. Core: based on language features.

1.1.1.1

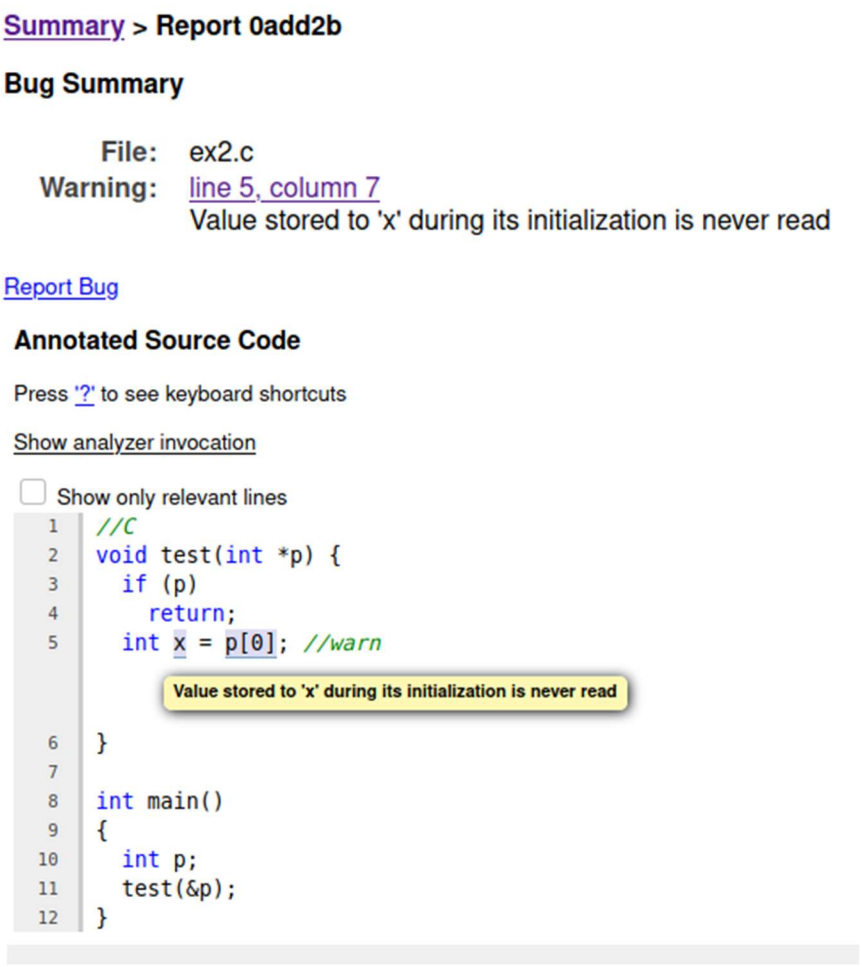
Bug found was that function pointer was assigned to null. Null dereference in C makes the program run in an undefined behavior hence, not allowed. This can be rectified by making the function pointer foo point to a valid function pointer i.e test.



```c
//C
void test() {
   void (*foo)(void)=test; //assigning function pointer foo to test()
   foo();
  }
void main()
{
   test();
}
```

1.1.1.4

The bug found was that the variable x initialized when the function test was never used before the function is returned making it a dead assignment. This is fixed by adding the print statement which reads and prints the value of x.



```c
// C
#include<stdio.h>
void test(int *p) {
  if (p) //if returns true
    return;
  int x = p[0];
  printf("%d",x);//variable x is printed
}
int main()
{
  int p={1};
  test(&p); // pass as the argument
}
```

1.1.1.5

Here the variable x is declared as static i.e the value it holds remains same throughout the program execution. Since x holds the reference of a local variable y, it becomes a dangling pointer when the function test() returns. This is fixed by making the local variable y as static too.

Summary > Report c49e23

Bug Summary

File: ex3.c
Warning: line 5, column 5
Address of stack memory associated with local variable 'y' is still referred to by the static variable 'x' upon returning to the caller. This will be a dangling reference

Report Bug

Annotated Source Code

Press '?' to see keyboard shortcuts

Show analyzer invocation

Show only relevant lines

```c
1
2  void test() {
3    static int *x;
4    int y;
5    x = &y; // warn
       (2) Address of stack memory associated with local variable 'y' is still referred to by the static variable 'x' upon returning to the caller. This will be a dangling reference
6  }
7
8  void main() {
9    test();
       (1) Calling 'test' →
10 }
```

```
void test() {
  static int *x;
  static int y; //making y as a static variable instead of local so the
value it holds persist throughout the program execution.
  x = &y; // Here x will still be referencing y beyond the scope of test().

}

void main() {
   test();
}
```

1.1.1.6

This example generated 2 bugs. One being a dead initialization where the initialized value of y is never read, and another is the value stored in y is a result of addition operation using an uninitialized variable x. Since x is uninitialized, it holds a garbage value by default. This is corrected by initializing x with an integer value and printing the resultant y value.

Bug Group	Bug Type ▾	File	Function/Method	Line	Path Length		
Dead store	Dead initialization	ex4.c	test	3	1	View Report	Re Bu
Logic error	Result of operation is garbage or undefined	ex4.c	test	3	3	View Report	Re Bu

Bug Summary

File: ex4.c
Warning: line 3, column 7
Value stored to 'y' during its initialization is never read

Report Bug

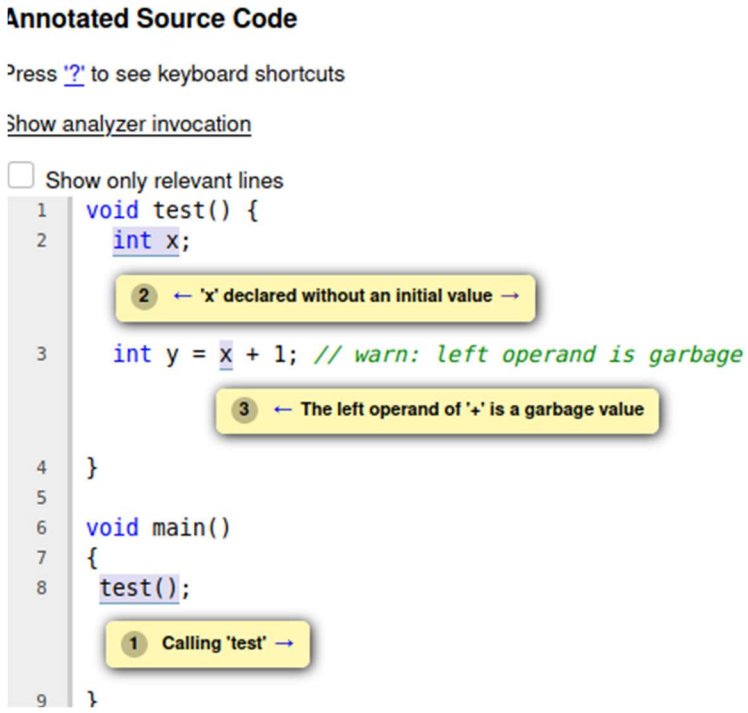
Annotated Source Code

Press '?' to see keyboard shortcuts

Show analyzer invocation

Show only relevant lines

```
1  void test() {
2    int x;
3    int y = x + 1; // warn: left operand is garbage
         Value stored to 'y' during its initialization is never read
4  }
5
6  void main()
7  {
8   test();
9  }
```

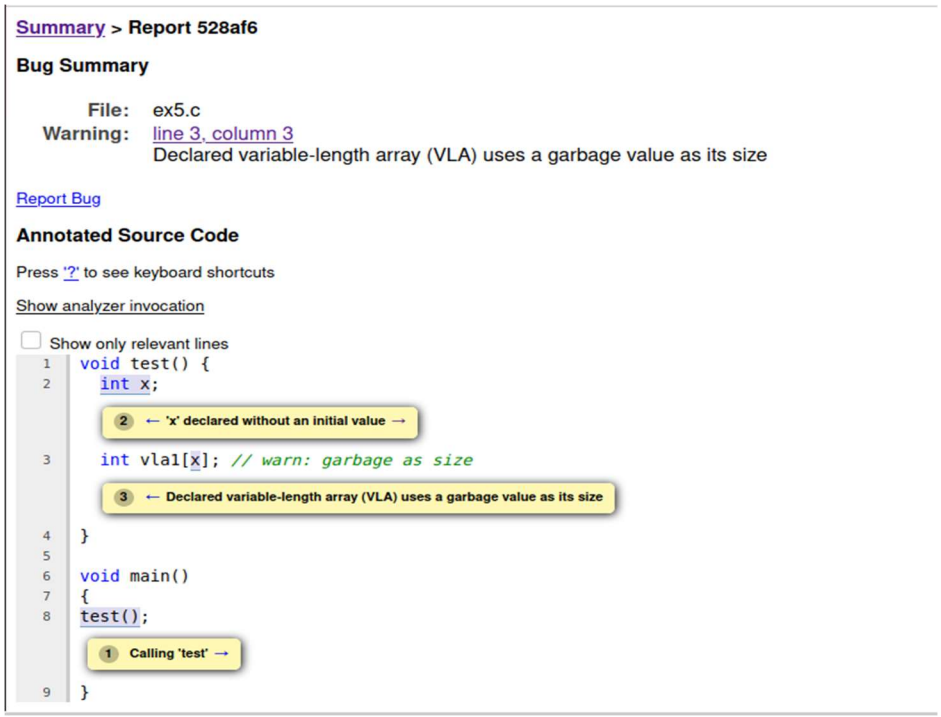



```c
#include<stdio.h>
void test() {
  int x = 3; // initialize x with an integer value
  int y = x + 1; // result of add operation is stored in y
  printf("%d",y); //printing the stored y value.
}

void main()
{
  test();
}
```

1.1.1.7

The bug found in this example is that the array vla uses an uninitialized variable x to define its size, hence, making the array size to be a garbage value. This can be fixed by initializing the variable x with a non-zero integer value and using it to define the size of the array.

Bug Group	Bug Type ▾	File	Function/Method	Line	Path Length			
Logic error	Dangerous variable-length array (VLA) declaration	ex5.c	test	3	3	View Report	Report Bug	Open File




```
1 void test() {
2   int x=10; //initializing x with a non-zero integer so the size of array vla1 will be valid.
3   int vla1[x];
4 }
5
6 void main()
7 {
8 test();
9 }
```

1.1.1.8

In this example, the given array ‘a’ has not been initialized and, we are trying to initialize variable x with a value within the array ‘a’ using an uninitialized variable ‘i’ as the index. Lastly, this initialized variable x is not used making it a dead initialization.

To fix these bugs, we first initialize the array with some values and assign a valid integer to the variable ‘i’ so it can be used for array indexing and lastly, printing this newly assigned value ‘x’.

Bug Group	Bug Type ▾	File	Function/Method	Line	Path Length			
Logic error	Array subscript is undefined	ex6.c	test	3	3	View Report	Report Bug	Open File
Dead store	Dead initialization	ex6.c	test	3	1	View Report	Report Bug	Open File

Summary > Report cfad8d

Bug Summary

File: ex6.c
Warning: line 3, column 11
Array subscript is undefined

Report Bug

Annotated Source Code

Press '?' to see keyboard shortcuts

Show analyzer invocation

Show only relevant lines

```
1  void test() {
2    int i, a[10];
       (2) ← 'i' declared without an initial value →
3    int x = a[i]; // warn: array subscript is undefined
       (3) ← Array subscript is undefined
4  }
5
6  void main(){
7  test();
     (1) Calling 'test' →
8  }
```

Summary > Report 7f792c

Bug Summary

File: ex6.c
Warning: line 3, column 7
Value stored to 'x' during its initialization is never read

Report Bug

Annotated Source Code

Press '?' to see keyboard shortcuts

Show analyzer invocation

Show only relevant lines

```
1  void test() {
2    int i, a[10];
3    int x = a[i]; // warn: array subscript is undefined
       Value stored to 'x' during its initialization is never read
4  }
5
6  void main(){
7  test();
8  }
```

```c
#include <stdio.h>
void test() {
  int i=0, a[]={1,2}; // Initialize x and array a
  int x = a[i];
  printf("%d", x); //Initialized variables needs to be read, hence print x.
}

void main(){
test();
}
```

1.1.1.9

In this example, the variable x is used to perform a bitwise OR operation while it is uninitialized. The variable x is not utilized in the program thus resulting in a dead store. To fix this bug , we simply initialize the variable x with a value of integer datatype and print this variable x.

Bug Group	Bug Type ▾	File	Function/Method	Line	Path Length			
Logic error	Assigned value is garbage or undefined	ex7.c	test	3	3	View Report	Report Bug	Open File
Dead store	Dead increment	ex7.c	test	3	1	View Report	Report Bug	Open File

Summary > **Report 87119d**

Bug Summary

File: ex7.c
Warning: line 3, column 5
The left expression of the compound assignment is an uninitialized value. The computed value will also be garbage

Report Bug

Annotated Source Code

Press '?' to see keyboard shortcuts

Show analyzer invocation

Show only relevant lines

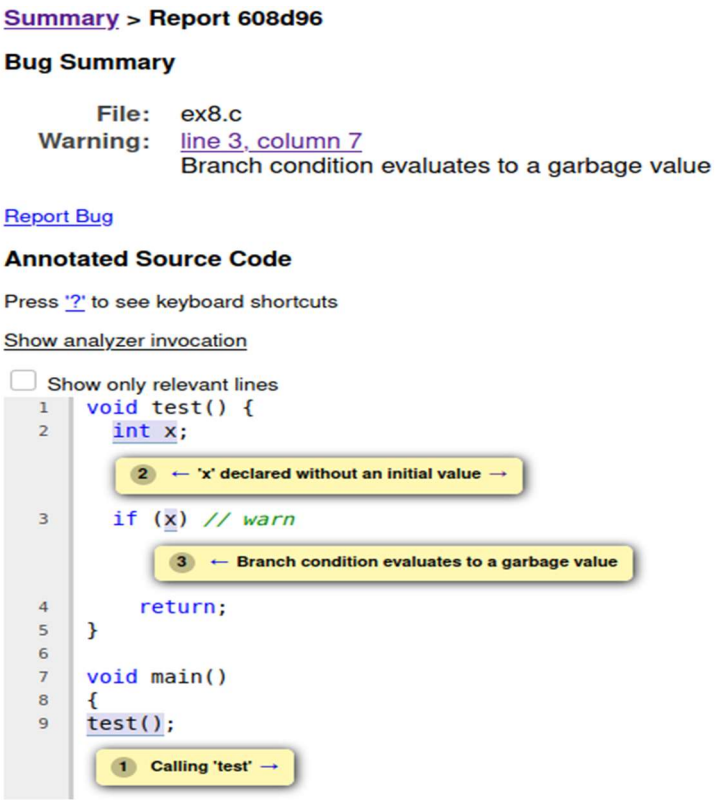
```c
1  void test() {
2    int x;
       2 ← 'x' declared without an initial value →
3    x |= 1; // warn: left expression is uninitialized
       3 ← The left expression of the compound assignment is an uninitialized value. The computed value will also be garbage
4  }
5
6  void main() {
7  test();
     1 Calling 'test' →
8  }
```


```c
#include<stdio.h>
void test() {
  int x=0; //initialize x to perform bitwise OR operation in the next line.
  x |= 1;
  printf("%d", x);
}

void main() {
test();
}
```

1.1.1.10

Here, the branching condition for if statement is based off the value of x which holds a garbage value. This is fixed by initializing x so that the if statement evaluates to be valid (true in this case).

```c
void test() {
  int x=1;
  if (x) //Since x is initialized, if statement will evaluate to be true
    return;
}

void main()
{
test();
}
```



1.1.1.11

I used clang instead of gcc to support the block type extension and build the program using the following command:

```
sowmyashree@sowmyashree-VirtualBox:~$ scan-build -o . clang ex9.c -fblocks -lBlocksRuntime
scan-build: Using '/usr/lib/llvm-10/bin/clang' for static analysis
```

Two bugs were found, one was that the block was capturing an uninitialized variable x. And the value of y was unused after assignment. This was fixed by initializing x before it is captured by the block and later printing the value of y to fix dead assignment.

File: ex9.c
Warning: line 6, column 3
Variable 'x' is uninitialized when captured by block

Report Bug

Annotated Source Code

Press '?' to see keyboard shortcuts

Show analyzer invocation

Show only relevant lines

```
1  #include<stdio.h>
2  #include <Block.h>
3
4  void test() {
5    int x;
       (2) ← 'x' declared without an initial value →
6    ^{ int y = x; }();
       (3) ← Variable 'x' is uninitialized when captured by block
7  }
8  int main()
9  {
10  test();
       (1) Calling 'test' →
11  return 0;
```

Bug Summary

File: ex9.c
Warning: line 6, column 10
Value stored to 'y' during its initialization is never read

Report Bug

Annotated Source Code

Press '?' to see keyboard shortcuts

Show analyzer invocation

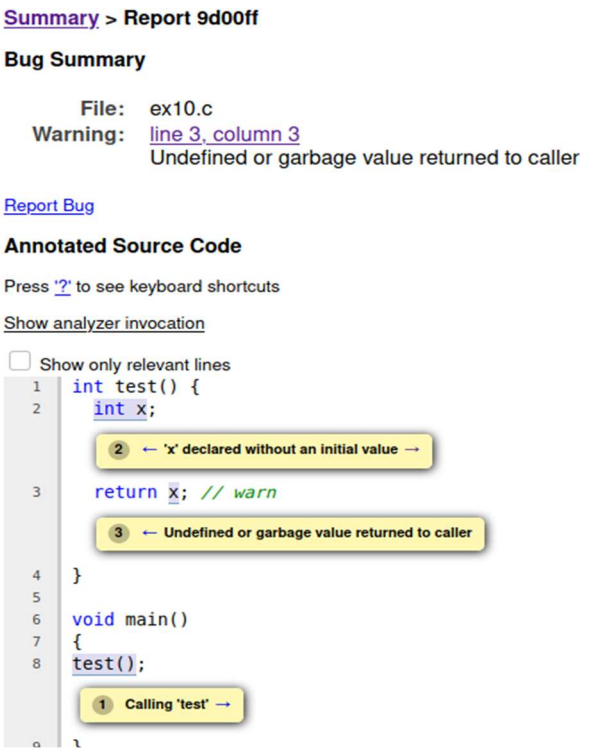
Show only relevant lines

```
1  #include<stdio.h>
2  #include <Block.h>
3
4  void test() {
5    int x;
6    ^{ int y = x; }();
           Value stored to 'y' during its initialization is never read
7  }
8  int main()
9  {
10  test();
11  return 0;
12  }
```

```c
#include<stdio.h>
#include <Block.h>
//typedef void (^b1)();
void test() {
  int x=3; //initialize value of x
  ^{ int y = x;
   printf("%d",y);}(); //print the value of y
}
int main()
{
 test();
 return 0;
 }
```

1.1.1.12

The bug found is that the function test() returns an uninitialized value. Hence, it is fixed by assigned an integer value to x before returning it. Also since the test() returns an integer value, we assign that to a variable y in main() and print that value to avoid dead assignment.

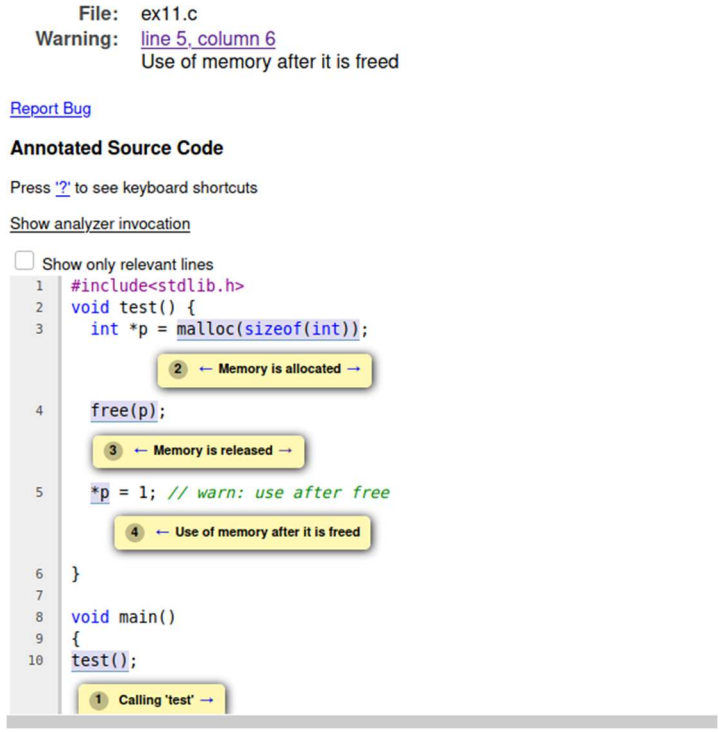


```c
#include<stdio.h>
int test() {
  int x=5;
  return x; //initialize x before returning the value to main
}

void main()
{
 int y=test();
 printf("%d",y);
}
```

b. Unix: checkers based on Posix/unix

1.1.7.2

This example throwed a memory error where it was trying to assign a value to the dynamically allocated integer pointer p after it was freed. This can be fixed can interchanging the lines 4 and 5 i.e freeing the pointer after the assignment.




```c
#include<stdlib.h>
void test() {
  int *p = malloc(sizeof(int));
 *p = 1;
 free(p); //free the pointer after its usage.
}

void main()
{
test();
}
```

1.1.7.3

Here, the bug was type mismatch while creating a pointer p dynamically using malloc. Conversion of short to long is incompatible. Hence, this can be fixed by changing datatype from short to long as the operand for sizeof().

Summary > Report 11c625

Bug Summary

File: ex12.c
Warning: line 3, column 13
Result of 'malloc' is converted to a pointer of type 'long', which is incompatible with sizeof operand type 'short'

Report Bug

Annotated Source Code

Press '?' to see keyboard shortcuts

Show analyzer invocation

Show only relevant lines

```
1  #include<stdlib.h>
2  void test() {
3    long *p = malloc(sizeof(short));
                Result of 'malloc' is converted to a pointer of type 'long', which is incompatible with sizeof operand type 'short'
4      // warn: result is converted to 'long *', which is
5      // incompatible with operand type 'short'
6    free(p);
7  }
8
9  void main()
10 {
11 test();
12 }
```

```
1 #include<stdlib.h>
2 void test() {
3   long *p = malloc(sizeof(long)); //Change short to long datatype to make it compatible.
4   free(p);
5 }
6
7 void main()
8 {
9 test();
10 }
```

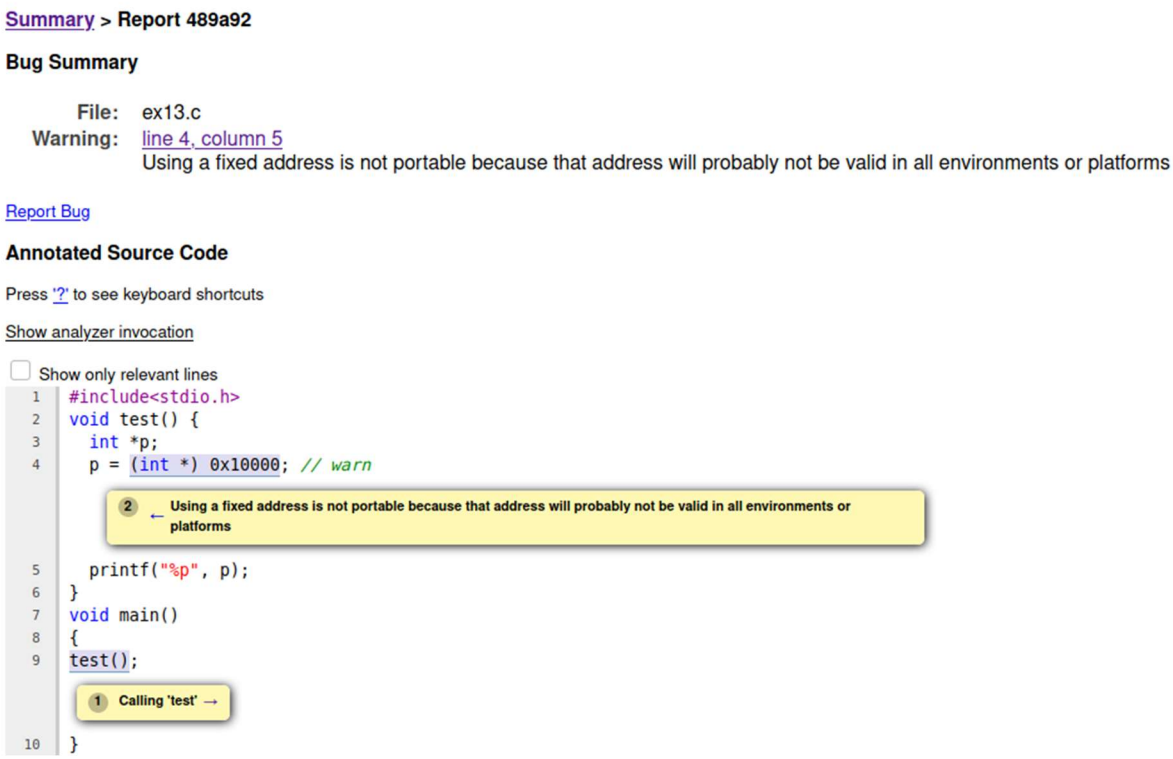
2. Experimental Checkers: To test this checkers we need to use additional options: **--use-analyzer=/usr/bin/clang -enable-checker <checkername>** while building.

1.2.2.8

We build using the following command:

```
sowmyashree@sowmyashree-VirtualBox:~$ scan-build --use-analyzer=/usr/bin/clang -enable-checker alpha.core.FixedAddr  -o . gcc ex13.c
scan-build: Using '/usr/bin/clang' for static analysis
```

In this example, we are assigning a fixed address to the pointer p. This is not recommended because the memory address during execution varies every-time. Hence, it can cause the program to behave in an unpredictable manner. To fix this, we can make the pointer as the reference of another variable/intended variable. This can be done using the '&' operator. Then print the value of p to fix dead assignment.



Summary > Report 3e6580

Bug Summary

File: ex13.c
Warning: line 3, column 3
Value stored to 'p' is never read

Report Bug

Annotated Source Code

Press '?' to see keyboard shortcuts

Show analyzer invocation

Show only relevant lines

```
1  void test() {
2    int *p;
3    p = (int *) 0x10000; // warn
     Value stored to 'p' is never read
4  }
5  void main()
6  {
7  test();
8  }
```

```
1 #include<stdio.h>
2 void test() {
3   int a;
4   int *p=&a; //Fixed addresses can't be assigned. Instead assign the address of the variable
5   *p = 100;
6   printf("%d", *p);
7 }
8 void main()
9 {
10 test();
11 }
```

1.2.2.10

We build using the following command:

```
owmyashree@sowmyashree-VirtualBox:~$ scan-build --use-analyzer=/usr/bin/clang -enable-checker alpha.core.PointerArithm  -o . gcc ex14.c
can-build: Using '/usr/bin/clang' for static analysis
14.c:4:3: warning: Value stored to 'p' is never read
```

Two bugs were found, one is where we are performing the arithmetic operations on the address and another being a dead assignment. Here performing such operations on the address of a variable involves manipulation of memory addresses which again can make the program unpredictable. This can be fixed by assigning a pointer p to reference the required variable, x and performing the addition operation using this pointer. Also, we need to initialize x with an integer to avoid it from holding garbage value.

Later print this variable to fix dead assignment.

Summary > Report 271326

Bug Summary

File: ex14.c
Warning: line 4, column 10
Pointer arithmetic on non-array variables relies on memory layout, which is dangerous

Report Bug

Annotated Source Code

Press '?' to see keyboard shortcuts

Show analyzer invocation

Show only relevant lines

```
1  void test() {
2    int x;
3    int *p;
4    p = &x + 1; // warn
       (2) ← Pointer arithmetic on non-array variables relies on memory layout, which is dangerous
5  }
6  void main(){
7  test();
     (1) Calling 'test' →
8  }
```

Summary > Report 3934bd

Bug Summary

File: ex14.c
Warning: line 4, column 3
Value stored to 'p' is never read

Report Bug

Annotated Source Code

Press '?' to see keyboard shortcuts

Show analyzer invocation

Show only relevant lines

```
1  void test() {
2    int x;
3    int *p;
4    p = &x + 1; // warn
       Value stored to 'p' is never read
5  }
6  void main(){
7  test();
8  }
```

```
1 void test() {
2   int x=0;
3   int *p = &x; // Create a reference/pointer to the variable x
4   *p = *p + 1; // Now, we can perform addition using that pointer.
5 }
6 void main(){
7 test();
8 }
```