Assignment 1

Yashica Patodia

19CS10067

1) Bug : The variable a is an irrational number and leads to loss of floating point precision when the sqrt(a) is squared again.

Correction: The if statement should be if (abs(a - sqr(b)) < ε), where ε can be defined as a very small floating point constant ie. 10^{-9}

Guidelines : To compare floating point numbers we should not use the "==" equality operator rather compare their absolute difference to a small positive constant.

2) The code leads to a segmentation fault.

No, the developed did not intend this. The developed must have intended to check if p is a null pointer or not to check if the memory allocation to p was successful or not.

Bug -: There is an assignment operation in the if stakment instead of an equality comparator. As a result in the if stakment the code makes p as a null pointer. This throws the exception

Fix- Instead of the assignment operator there should be equality comparator used.

```
if (p==NULL)
    cout << "No value" << endl;
else
    cout << (*p) << endl;
```

19CS10067

Yashica Patodia

Guidlines : While working with pointers, we should first check if the pointer is NULL or not and then proceed to perform operations on it (like printing its value). This will ensure that the memory allocation is successful.

.

Q3

Case 1:

Debug Build (Non-optimized)

if (r==0 || rem(n,r))

This first condition of if statement is check that and it is true, and since we are taking OR, the rest of statement is not checked and the compiler returns true.

if (rem(n,r) || r==0)

The call to function rem(n,r) runs the program to floating point exception because of doing modulus by zero

Release Build (Optimised)

Here the value of n and r are available during the compile time and during the release built it can optimise the code by evaluating the constant expression. Both the if statements have (r==0), since r is a constant it is evaluated and due to 'or' operator both the if statements return true and hence no error.

19CS10057
Yashica Patodia

Case 2

Here the behaviour of the compiler does not change depending on the build because the values of n and r are provided at run time, and hence for the first if statement it returns true and the second one it throws an exception similar to the debug built of Case 1.

Guideline : To avoid such error, we should not take modulus or division by 0. To do that we should always check if the divisor is 0 or not

```
if (divisor == 0)
    {
        // task 1
    }
else
    {
        // task 2 .
    }
```

19CS10064

Yashica Potodia

Q4 The statement char* str = "Anything" creates a string literal. The string literal is stored in the read-only part of the memory. According to the C++ standards, the string literals have static storage duration and any attempt at modifying them gives undefined behaviour

Function Name	Behaviour	Justification & comments
f1()	Compilation Error	str[0] = 'C', this expression throws error because the value of string literal cannot be changed. No error in str = "rat" and a warning is seen. "deprecated conversion from string constant to 'char*'" can be avoided by making it const
f2()	Compilation Error	Similar to f1() but now the warning is not shown because we have used to 'const' operator,
f3()	Compilation Error	Error is thrown due to both the following reasons char *const str = "Bat" 1) str[0] = 'C' // cannot edit the string literal 2) str = "Rat" // cannot change the string literal

19CS10067
Yashica Patodia

function name	Behaviour	Justification/Comments
f4()	Correct output	When strdup("Bat") is used it makes a pointer duplicate for bat and hence is equal to str. Since it is not const, it can be edited as well
f5()	Compilation error	str[0] = 'c' this expression throughs error as it cannot be edited but no error in changing, ie: str = strdup("Rat");
f6()	Compilation error	str[0] = 'c' this expression works fine, but in this statement ie str = strdup("Rat"); it throws error bcs the pointer itself is const hence cannot point anywhere else

Q5)

Line No	Behaviour	Justification / Comments
1.	Compilation Error	It tries a bind a non const reference (int &v) to an integer literal (e(a) returns) which throws compilation error. We can correct it by passing the address of int in the function e(x).
2.	Compilation Error	Similar to first line, the function f() is returning a constant and b cannot bind it to a non const reference.
3.	Unpredictable behaviour / wrong output	g() is returning reference to a local variable (since it calls by value). Output x=10 &x= (no address of local variable)
4.	Correct	It is correct and .h() returns the address of x (since it calls by reference). Output :- x=10 &x= (address of x)
5	wrong output / ~~compilation~~ Unpredictable behaviour	The variable is left unassigned due to previous error
6.	wrong output / unpredictable behaviour	The variable is left unassigned due to previous error.

19CS10064

Yashica Pahodia

Line Number	Behaviour	Justification/ Comments
7.	Unpredicable Output	It has reference to a variable that was local to g()
8.	correct output ***r=10 & &r= (address)	h() behaviour as expected
9.	Correct.	const refernce binds to return by value
10.	Correct	const refernce binds to return by value
11	Unpredictable behaviour	g() returns refernce to a local variable
12.	Correct	h() behaviour as expected
13.	Correct output	Value same as 'a' but different address
14.	Correct output	Value same as 'c' but diff address
15.	Unpredictable output	It has refernce to a local variable of g()
16.	Correct output	value and address same as 'a'

FACS10067
Yashica Patodia

Line	Behaviour	Justification/comment
17	Compilation error	e() returns by value and it is not a valid lvalue
18	Wrong output	Value of a remains 10 and not 1
19	Compilation Error	f() returns by value and it is not valid
20	Wrong Output	Similar to line 18, value of a remains 10 and not 1
21	Unpredictable behaviour	g() returns refrence to a local variable and not a
22	Wrong Output	a remains 10 and not 3
23	Correct	h() returns refernce to 'a' which is assign 4
24	Correct output	Value of 'a' changes to 4