Computer Science & Engineering Department
I. I. T. Kharagpur

Software Engineering: CS20006
SOLUTIONS

Assignment – 1: Better C & Guidelines *Marks: 30*

Assign Date: *19th January, 2021* Submit Date: *23:55, 23rd January, 2021*

Instructions: Please solve the questions using pen and paper and scan the images. Every image should contain your roll number and name.

1. Consider the following program which should obviously print `Match`.

```
#include <iostream>
#include <cmath>
using namespace std;

#define sqr(x) ((x) * (x))

int main() {
    double a = 4.0*atan(1.0); // pi
    double b = sqrt(a); // square-root of pi

    if (a == sqr(b)) // pi is equal to
                     // square of square-root of pi?
        cout << "Match" << endl;
    else
        cout << "Mis-Match" << endl;

    return 0;
}
```

However, on Visual C++ 64-bit compiler, it prints `Mis-Match`. Identify the bug and fix it. **[1 + 1 = 2]**

Write an appropriate guideline to avoid such bugs and improve the quality of the code. **[1]**

BEGIN SOLUTION

Bug: In `if (a == sqr(b))` we compare two floating point (`double`) values which theoretically should be same. But in floating point representation such values often differ by a small quantity.

Fix: To fix this bug, we need to compare their absolute difference and check if it is smaller than a tiny preset quantity. For example, we may replace

```
if (a == sqr(b))
```

by

```
const double prec = 1e-7;
if (fabs(a - b * b) < prec)
```

Guideline:
Do not compare two floating point quantities for equality – check their absolute difference to be small enough

END SOLUTION

2. What is the output of the following program? **[1]**

```
#include <iostream>
using namespace std;

int main() {
    int *p = new int(5);

    if (p = 0)
        cout << "No Value" << endl;
    else
        cout << *p << endl;

    return 0;
}
```

Is it what the developer intended? If yes, justify the thoughts of the developer. If no, find the bug in the program and fix it. **[1]**

Write an appropriate guideline to avoid such bugs and improve the quality of the code. **[1]**

BEGIN SOLUTION

Bug: In `if (p = 0)` the developer did a typo by writing the assignment operator (=) in place of the intended the equality operator (==). Hence, even though `p` has a proper allocated pointer value, it is reset to `NULL` and `No Value` is printed instead of `5`.

Fix: To fix this bug, we need to fix the typo. That is, we replace

```
if (p = 0)
```

by

```
if (p == 0)
```

Better still would be to replace by:

```
if (0 == p)
```

With that even if there is a typo like

```
if (0 = p)
```

it will be a Compilation Error.

Guideline:
While comparing a quantity for equality with `0`, write `0` as the left operand

END SOLUTION

3. Consider the following program:

```
#include <iostream>
using namespace std;

int rem(int n, int r) {
    return n % r;
}

int main() {
    int n = 15, r = 0; // Line 1

//  int n, r; // Line 2
//  cin >> n >> r; // Line 3

    if (r == 0 || rem(n, r))
        cout << "True" << endl;
    else
        cout << "False" << endl;

    if (rem(n, r) || r == 0)
        cout << "True" << endl;
    else
        cout << "False" << endl;

    return 0;
}
```

While using Visual C++ 64-bit compiler, the output is as follows:

Build Type	Output
Debug (Un-optimized)	`True` `Un-handled Floating Point Exception`
Release (Optimized)	`True` `True`

BEGIN SOLUTION

The behavior is due to optimization in Release build (as `r` is known to be `0` at compile time and due to short-cut computation of the boolean expressions:

Build Type	Output	Reason
Debug (Un-optimized)	`True` `FP Exception`	`r == 0` is `True` ⇒ condition is `True`. `rem(n, r)` is not evaluated `rem(n, r)` is evaluated. Exception on division by zero
Release (Optimized)	`True` `True`	`r == 0` *is* `0` & `r == 0` is statically `True`. Condition removed `r == 0` *is* `0` & `r == 0` is statically `True`. Condition removed

END SOLUTION

Now let us comment `Line 1` and un-comment `Line 2` & `Line 3`. The output changes to the following while we input `15` for `n` and `0` for `r` (as was initialized in `Line 1`):

Build Type	Output
Debug (Un-optimized)	`True` `Un-handled Floating Point Exception`
Release (Optimized)	`True` `Un-handled Floating Point Exception`

BEGIN SOLUTION

Now `r` is an input and not known at compile time. So Release build cannot optimize and behaves like the Debug build with short-cut computation of the boolean expressions:

Build Type	Output	Reason
Debug (Un-optimized)	`True` `FP Exception`	`r == 0` is `True` ⇒ condition is `True`. `rem(n, r)` is not evaluated `rem(n, r)` is evaluated. Exception on division by zero
Release (Optimized)	`True` `FP Exception`	`r == 0` is `True` ⇒ condition is `True`. `rem(n, r)` is not evaluated `rem(n, r)` is evaluated. Exception on division by zero

END SOLUTION

Explain the behavior in both cases, especially justifying the difference due to changing `Line 1` to `Line 2` & `Line 3` and providing the same input.

[2 + 2 = 4]

Write an appropriate guideline to avoid such bugs and improve the quality of the code. **[1]**

BEGIN SOLUTION

Guidelines:
Don't trust the compiler - Release and Debug builds may behave differently.
Remember that you may or may not have shortcut computation of a boolean expression while writing code

END SOLUTION

4. Consider the following program having 6 functions - each being a slight variant of the other. State the behavior (like *Compilation Error*, *wrong output*, *run-time exception*, *Correct Output - showing the output*, *unpredictable behavior*, etc.) of each function with proper justification (refer to specific lines in a function as you may need) of the behaviour as stated. You may compare the functions also from the perspectives of the quality of the code. Make a table in the following format in your submission sheet and fill up accordingly. **[0.3 * 6 = 3]**

Function Name	Behaviour	Justification & Comments
`f1()`		
`f2()`		
`...`		
`f6()`		

Finally, based on the observations above, formulate guidelines to maintain a good quality of code. **[2]**

```
#include <iostream>
#include <cstring>
using namespace std;

void f1() {
    char * str = "Bat";
    cout << str << endl;
    str[0] = 'C';
    cout << str << endl;
    str = "Rat";
    cout << str << endl;
    cout << endl;
}

void f2() {
    const char * str = "Bat";
    cout << str << endl;
    str[0] = 'C';
    cout << str << endl;
    str = "Rat";
    cout << str << endl;
    cout << endl;
}
```

```
void f3() {
    char * const str = "Bat";
    cout << str << endl;
    str[0] = 'C';
    cout << str << endl;
    str = "Rat";
    cout << str << endl;
    cout << endl;
}

void f4() {
    char * str = strdup("Bat");
    cout << str << endl;
    str[0] = 'C';
    cout << str << endl;
    str = strdup("Rat");
    cout << str << endl;
    cout << endl;
}

void f5() {
    const char * str = strdup("Bat");
    cout << str << endl;
    str[0] = 'C';
    cout << str << endl;
    str = strdup("Rat");
    cout << str << endl;
    cout << endl;
}

void f6() {
    char * const str = strdup("Bat");
    cout << str << endl;
    str[0] = 'C';
    cout << str << endl;
    str = strdup("Rat");
    cout << str << endl;
    cout << endl;
}

int main() {
    f1();
    f2();
    f3();
    f4();
    f5();
    f6();

    return 0;
}
```

BEGIN SOLUTION

Function Name	Behaviour	Justification and Comments
`f1()`	Warning & Run-time Exception	Warning: In `char * str = "Bat";` and `str = "Rat";` a constant string is being assigned to pointer to non-const char Segmentation fault: `str[0] = 'C';` is a violation as constant strings are stored in const segment and cannot be changed
`f2()`	Compilation Error	Error: `str[0] = 'C';` violates const-ness of declaration
`f3()`	Warning & Compilation Error	Warning: In `char * const str = "Bat";` as a constant string is being assigned to a pointer to non-const char Error: In `str = "Rat";` a constant pointer is being assigned
`f4()`	Correct Output	`strdup` function dynamically allocates memory, hence can be changed. However, there will be memory leak as none of the duplicate allocated strings have been released
`f5()`	Compilation Error	Error: `str[0] = 'C';` violates const-ness of declaration
`f6()`	Compilation Error	Error: In `str = strdup("Rat");` a constant pointer is being assigned

Guideline:
Always honour the const-ness of the pointer and the pointee

END SOLUTION

5. Consider the following program where 24 lines have been marked. State the behavior (like *Compilation Error*, *wrong output*, *run-time exception*, *Correct Output - showing the output*, *unpredictable behavior*, etc.) of each line with proper justification (refer to specific lines in a function as you may need) of the behaviour as stated. Make a table in the following format in your submission sheet and fill up accordingly. **[0.5 * 24 = 12]**

Line #	Behaviour	Justification & Comments
`Line 01`		
`Line 02`		
`...`		
`Line 24`		

Finally, based on the observations above, formulate guidelines to maintain a good quality of code. **[2]**

```
#include <iostream>
#include <cmath>
using namespace std;

int e(int x) {
    cout << "x = " << x << " &x = " << &x << endl;
    return (x);
}
int f(int &x) {
    cout << "x = " << x << " &x = " << &x << endl;
    return (x);
}
int& g(int x) {
    cout << "x = " << x << " &x = " << &x << endl;
    return (x);
}
int& h(int &x) {
    cout << "x = " << x << " &x = " << &x << endl;
    return (x);
}
```

```
int main() {
    int a = 10;
    cout << "a = " << a << " &a = " << &a << endl;

    int& rvv = e(a); // Line 01
    int& rrv = f(a); // Line 02
    int& rvr = g(a); // Line 03
    int& rrr = h(a); // Line 04

    cout << "rvv = " << rvv << " &rvv = " << &rvv << endl; // Line 05
    cout << "rrv = " << rrv << " &rrv = " << &rrv << endl; // Line 06
    cout << "rvr = " << rvr << " &rvr = " << &rvr << endl; // Line 07
    cout << "rrr = " << rrr << " &rrr = " << &rrr << endl; // Line 08

    const int& rvvc = e(a); // Line 09
    const int& rrvc = f(a); // Line 10
    const int& rvrc = g(a); // Line 11
    const int& rrrc = h(a); // Line 12

    cout << "rvvc = " << rvvc << " &rvvc = " << &rvvc << endl; // Line 13
    cout << "rrvc = " << rrvc << " &rrvc = " << &rrvc << endl; // Line 14
    cout << "rvrc = " << rvrc << " &rvrc = " << &rvrc << endl; // Line 15
    cout << "rrrc = " << rrrc << " &rrrc = " << &rrrc << endl; // Line 16

    e(a) = 1; // Line 17
    cout << "a = " << a << " &a = " << &a << endl; // Line 18
    f(a) = 2; // Line 19
    cout << "a = " << a << " &a = " << &a << endl; // Line 20
    g(a) = 3; // Line 21
    cout << "a = " << a << " &a = " << &a << endl; // Line 22
    h(a) = 4; // Line 23
    cout << "a = " << a << " &a = " << &a << endl; // Line 24

    return 0;
}
```

BEGIN SOLUTION

Line #	Behaviour	Justification and Comments
Line 01	Compilation Error	"cannot bind non-const lvalue reference of type 'int&' to an rvalue of type 'int'." - `e(a)`, the return by value, is a temporary object that cannot be referred by a non-const reference `rvv`.
Line 02	Compilation Error	Same error as Line 01. - `f(a)`, the return by value, is a temporary object that cannot be referred by a non-const reference `rrv`.
Line 03	Correct O/p	Works fine printing the values of `a` and `x` though they have difference addresses. However, this is risky as the reference `rvr` is to a local variable (`x` in `f` - copy of `a`) is being made. Note the consequence in Line 07
Line 04	Correct O/p	Works fine printing the values of `a` and `x` having same addresses. This is safe as `rrr` becomes a reference to `a`. Check Line 08 too.
Line 05	Compilation Error	Line 05 is dependent on Line 01.
Line 06	Compilation Error	Line 06 is dependent on Line 02.
Line 07	Run-time Exception	In Line 03, `rvr` was set to local variable that went out of scope. So it become a null pointer.
Line 08	Correct O/p	`rrr` has a valid reference to `a` (Line 04). It will print correct values.
Line 09	Correct O/p	Unlike Line 01, this works fine as we are use a const reference `rvvc` to (a copy of) `e(a)` - the return by value temporary object. Check Line 13.
Line 10	Correct O/p	Like Line 09 - had failed for Line 02 - works fine with const reference
Line 11	Correct O/p	Like Line 03, this works fine. But is it safe? Check Line 15
Line 12	Correct O/p	Same as Line 04 - safer with const reference
Line 13	Correct O/p	Good value set in Line 09. Works fine here.
Line 14	Correct O/p	Good value set in Line 10. Works fine here.
Line 15	Run-time Exception	In Line 11, `rvrc` was set to local variable that went out of scope. So it become a null pointer.
Line 16	Correct O/p	Works fine Line 12 is correct and safe
Line 17	Compilation Error	Function `e` returns by value. So `e(a)` is an r-value, not a modifiable l-value as needed.
Line 18	Correct O/p	No change to `a` as Line 17 does not work.
Line 19	Compilation Error	Function `f` returns by value. So `f(a)` is an r-value, not a modifiable l-value as needed.
Line 20	Correct O/p	No change to `a` as Line 19 does not work.
Line 21	Run-time Exception	`g(a)` is a reference to a local variable. So in `main` it turns out to be a null pointer for the reference.
Line 22	Correct O/p	No change to `a` as Line 21 does not work.
Line 23	Correct O/p	`h(a)` is a reference to `a`. So 4 gets rightly assigned to `a`.
Line 24	Correct O/p	'a' value will be 4 and address will not change.

Guideline:
Do not return reference to local variables of a function.
Do not set reference to a value returned by value from a function - it is a temporary object. Use const reference for variables that really exist

END SOLUTION