Lab Session VII: Program Inspection and Debugging

Prerequisites: Before the lab you should check that the Eclipse/NetBeans IDE is being installed in the lab system.

Preparation: Review ten code fragments given as the text files using two ways, (1) the code is reviewed using the inspection (checklist-based approach) technique (2) the debugger in the IDE is used to identify the error and review the code fragment

Activities: Having considered the code fragments you should carry out the following activities that will be facilitated by your TA/Course Instructor.

You can also choose your own code fragment from GitHub (more than 1000 LOC) in any programming language sand do the inspection and debugging.

I. PROGRAM INSPECTION:

An Error Checklist for Inspections

An important part of the inspection process is the use of a checklist to examine the program for common errors. Unfortunately, some checklists concentrate more on issues of style than on errors (for example, "Are comments accurate and meaningful?" and "Are if- else, code blocks, and do..while groups aligned?"), and the error checks are too nebulous to be useful (such as "Does the code meet the design requirements?"). The checklist in this section was compiled after many years of study of software errors. The checklist is largely language independent, meaning that most of the errors can occur with any programming language. You may wish to supplement this list with errors peculiar to your programming language and with errors detected after using the inspection process.

Category A: Data Reference Errors

1. Does a referenced variable have a value that is unset or uninitialized? This probably is the most frequent programming error; it occurs in a wide variety of circumstances. For each reference to a data item (variable, array element, field in a structure), attempt to "prove" informally that the item has a value at that point.
2. For all array references, is each subscript value within the defined bounds of the corresponding dimension?
3. For all array references, does each subscript have an integer value? This is not necessarily an error in all languages, but it is a dangerous practice.
4. For all references through pointer or reference variables, is the referenced memory currently allocated? This is known as the "dangling reference" problem. It occurs in situations where the lifetime of a pointer is greater than the lifetime of the referenced memory. One situation occurs where a pointer references a local variable within a procedure, the pointer value is assigned to an output parameter or a global variable, the procedure returns (freeing the referenced location), and later the program attempts to use the pointer value. In a manner similar to checking for the prior errors, try to prove informally that, in each reference using a pointer variable, the reference memory exists.
5. When a memory area has alias names with differing attributes, does the data value in this area have the correct attributes when referenced via one of these names? Situations to look for are the use of the EQUIVALENCE statement in FORTRAN, and the REDEFINES clause in COBOL. As an

example, a FORTRAN program contains a real variable A and an integer variable B; both are made aliases for the same memory area by using an EQUIVALENCE statement. If the program stores a value into A and then references variable B, an error is likely present since the machine would use the floating-point bit representation in the memory area as an integer.

6. Does a variable's value have a type or attribute other than what the compiler expects? This situation might occur where a C, C++, or COBOL program reads a record into memory and references it by using a structure, but the physical representation of the record differs from the structure definition.
7. Are there any explicit or implicit addressing problems if, on the machine being used, the units of memory allocation are smaller than the units of memory addressability? For instance, in some environments, fixed-length bit strings do not necessarily begin on byte boundaries, but addresses only point to byte boundaries. If a program computes the address of a bit string and later refers to the string through this address, the wrong memory location may be referenced. This situation also could occur when passing a bit-string argument to a subroutine.
8. If pointer or reference variables are used, does the referenced memory location have the attributes the compiler expects? An example of such an error is where a C++pointer upon which a data structure is based is assigned the address of a different data structure.
9. If a data structure is referenced in multiple procedures or subroutines, is the structure defined identically in each procedure?
10. When indexing into a string, are the limits of the string off by-one errors in indexing operations or in subscript references to arrays?
11. For object-oriented languages, are all inheritance requirements met in the implementing Class?

Category B: Data-Declaration Errors

1. Have all variables been explicitly declared? A failure to do so is not necessarily an error, but it is a common source of trouble. For instance, if a program subroutine receives an array parameter, and fails to define the parameter as an array (as in a DIMENSION statement, for example), a reference to the array (such as C=A (I)) is interpreted as a function call, leading to the machine's attempting to execute the array as a program. Also, if a variable is not explicitly declared in an inner procedure or block, is it understood that the variable is shared with the enclosing block?
2. If all attributes of a variable are not explicitly stated in the declaration, are the defaults well understood? For instance, the default attributes received in Java are often a source of surprise.
3. Where a variable is initialized in a declarative statement, is it properly initialized? In many languages, initialization of arrays and strings is somewhat complicated and, hence, error prone.
4. Is each variable assigned the correct length and data type?
5. Is the initialization of a variable consistent with its memory type?
6. Are there any variables with similar names (VOLT and VOLTS, for example)? This is not necessarily an error, but it should be seen as a warning that the names may have been confused somewhere within the program.

Category C: Computation Errors

1. Are there any computations using variables having inconsistent (such as non-arithmetic) data types?
2. Are there any mixed-mode computations? An example is the addition of a floating-point variable to an integer variable. Such occurrences are not necessarily errors, but they should be explored carefully to ensure that the language's conversion rules are understood. Consider the following Java snippet showing the rounding error that can occur when working with integers:

```
int x = 1;
int y = 2;
int z = 0;
        z = x/y;
                System.out.println ("z = " + z);
```

OUTPUT:

z = 0

3. Are there any computations using variables having the same data type but different lengths?
4. Is the data type of the target variable of an assignment smaller than the data type or result of the right-hand expression?
5. Is an overflow or underflow expression possible during the computation of an expression? That is, the end result may appear to have valid value, but an intermediate result might be too big or too small for the programming language's data types.
6. Is it possible for the divisor in a division operation to be zero?
7. If the underlying machine represents variables in base-2 form, are there any sequences of the resulting inaccuracy? That is, 10 x 0.1 is rarely equal to 1.0 on a binary machine.
8. Where applicable, can the value of a variable go outside the meaningful range? For example, statements assigning a value to the variable PROBABILITY might be checked to ensure that the assigned value will always be positive and not greater than
9. For expressions containing more than one operator, are the assumptions about the order of evaluation and precedence of operators correct?
10. Are there any invalid uses of integer arithmetic, particularly divisions? For instance, if i is an integer variable, whether the expression 2*i/2 == i depends on whether i has an odd or an even value and whether the multiplication or division is performed first.

Category D: Comparison Errors

1. Are there any comparisons between variables having different data types, such as comparing a character string to an address, date, or number?
2. Are there any mixed-mode comparisons or comparisons between variables of different lengths? If so, ensure that the conversion rules are well understood.
3. Are the comparison operators correct? Programmers frequently confuse such relations as at most, at least, greater than, not less than, less than or equal.
4. Does each Boolean expression state what it is supposed to state? Programmers often make mistakes when writing logical expressions involving and, or, and not.
5. Are the operands of a Boolean operator Boolean? Have comparison and Boolean operators been erroneously mixed together? This represents another frequent class of mistakes. Examples of a few typical mistakes are illustrated here. If you want to determine whether i is between 2 and 10, the expression 2<i<10 is incorrect; instead, it should be (2<i) && (i<10). If you want to determine whether i is greater than x or y, i>x||y is incorrect; instead, it should be (i>x)||(i>y). If you want to compare three numbers for equality, if(a==b==c) does something quite different. If you want to test the mathematical relation x>y>z, the correct expression is(x>y)&&(y>z).
6. Are there any comparisons between fractional or floating- point numbers that are represented in base-2 by the underlying machine? This is an occasional source of errors because of truncation and base-2 approximations of base-10 numbers.
7. For expressions containing more than one Boolean operator, are the assumptions about the order of evaluation and the precedence of operators correct? That is, if you see an expression such as (if((a==2) && (b==2) || (c==3)), is it well understood whether the *and* or the *or* is

performed first?

8. Does the way in which the compiler evaluates Boolean expressions affect the program? For instance, the statement if((x==0 && (x/y)>z) may be acceptable for compilers that end the test as soon as one side of an and is false, but may cause a division-by-zero error with other compilers.

Category E: Control-Flow Errors

1. If the program contains a multiway branch such as a computed GO TO, can the index variable ever exceed the number of branch possibilities? For example, in the statement

   GO TO (200, 300, 400), i

   will i always have the value of 1, 2, or 3?
2. Will every loop eventually terminate? Devise an informal proof or argument showing that each loop will terminate.
3. Will the program, module, or subroutine eventually terminate?
4. Is it possible that, because of the conditions upon entry, a loop will never execute? If so, does this represent an over- sight? For instance, if you had the following loops headed by the following statements:

```
for (i==x ; i<=z; i++) {
...
}
while (NOTFOUND) {
...
}
```

what happens if NOTFOUND is initially false or if x is greater than z?

5. For a loop controlled by both iteration and a Boolean condition (a searching loop, for example) what are the consequences of loop fall-through? For example, for the psuedo-code loop headed by

   DO I=1 to TABLESIZE WHILE (NOTFOUND)

   what happens if NOTFOUND never becomes false?
6. Are there any off-by-one errors, such as one too many or too few iterations? This is a common error in zero-based loops. You will often forget to count "0" as a number. For example, if you want to create Java code for a loop that counted to 10, the following would be wrong, as it counts to 11:

```
for (int i=0; i<=10;i++) {
System.out.println(i);
}
Correct, the loop is iterated 10 times:
for (int i=0; i <=9;i++) {
System.out.println(i);
```

7. If the language contains a concept of statement groups or code blocks (e.g., do-while or {...}), is there an explicit while for each group and do the do's correspond to their appropriate groups? Or is there a closing bracket for each open bracket? Most modern compilers will complain of such mismatches.
8. Are there any non-exhaustive decisions? For instance, if an input parameter's expected values are 1, 2, or 3, does the logic assume that it must be 3 if it is not 1 or 2? If so, is the assumption valid?

Category F: Interface Errors

1. Does the number of parameters received by this module equal the number of arguments sent by each of the calling modules? Also, is the order correct?
2. Do the attributes (e.g., data type and size) of each parameter match the attributes of each corresponding argument?
3. Does the units system of each parameter match the units system of each corresponding argument? For example, is the parameter expressed in degrees but the argument expressed in radians?
4. Does the number of arguments transmitted by this module to another module equal the number of parameters expected by that module?
5. Do the attributes of each argument transmitted to another module match the attributes of the corresponding parameter in that module?
6. Does the units system of each argument transmitted to another module match the units system of the corresponding parameter in that module?
7. If built-in functions are invoked, are the number, attributes, and order of the arguments correct?
8. Does a subroutine alter a parameter that is intended to be only an input value?
9. If global variables are present, do they have the same definition and attributes in all modules that reference them?

Category G: Input / Output Errors

1. If files are explicitly declared, are their attributes correct?
2. Are the attributes on the file's OPEN statement correct?
3. Is there sufficient memory available to hold the file your program will read?
4. Have all files been opened before use?
5. Have all files been closed after use?
6. Are end-of-file conditions detected and handled correctly?
7. Are I/O error conditions handled correctly?
8. Are there spelling or grammatical errors in any text that is printed or displayed by the program?

Category H: Other Checks

1. If the compiler produces a cross-reference listing of identifiers, examine it for variables that are never referenced or are referenced only once.
2. If the compiler produces an attribute listing, check the attributes of each variable to ensure that no unexpected default attributes have been assigned.
3. If the program compiled successfully, but the computer produced one or more "warning" or "informational" messages, check each one carefully. Warning messages are indications that the compiler suspects that you are doing something of questionable validity; all of these suspicions should be reviewed. Informational messages may list undeclared variables or language uses that impede code optimization.
4. Is the program or module sufficiently robust? That is, does it check its input for validity?
5. Is there a function missing from the program?

Program Inspection: (Submit the answers of following questions for each code fragment)

1. How many errors are there in the program? Mention the errors you have identified.
2. Which category of program inspection would you find more effective?
3. Which type of error you are not able to identified using the program inspection?
4. Is the program inspection technique is worth applicable?

II. CODE DEBUGGING: Debugging is the process of localizing, analyzing, and removing suspected errors in the code.

Instructions (Use Eclipse/Netbeans IDE, GDB Debugger)

- Open a NEW PROJECT. Select Java/C++ application. Give suitable name to the file.
- Click on the source file in the left panel. Click on NEW in the pull down menu.
- Select main Java/C++ file.
- Build and Run the project.
- Set a toggle breakpoint to halt execution at a certain line or function
- Display values of variables and expressions
- Step through the code one instruction at a time
- Run the program from the start or continue after a break in the execution
- Do a backtrace to see who has called whom to get to where you are
- Quit debugging.

Debugging: (Submit the answers of following questions for each code fragment)

1. How many errors are there in the program? Mention the errors you have identified.
2. How many breakpoints you need to fix those errors?
    a. What are the steps you have taken to fix the error you identified in the code fragment?
3. Submit your complete executable code?

Testing a Program – An Example

Question: Write a set of test cases – specific set of data – to properly test a relatively Simple program. Create a set of test data for the program - data the program must handle correctly to be considered a successful program. Here's a description of the program:

"The program reads three integer values from an input dialog. The three values represent the lengths of the sides of a triangle. The program displays a message that states whether the triangle is scalene, isosceles, or equilateral".

Code: The function *triangle* takes three integer parameters that are interpreted as the lengths of the sides of a triangle. It returns whether the triangle is equilateral (three lengths equal), isosceles (two lengths equal), scalene (no lengths equal), or invalid (impossible lengths).

```
final int EQUILATERAL = 0;
final int ISOSCELES = 1;
final int SCALENE = 2;
final int INVALID = 3;
int triangle(int a, int b, int c)
{
   if (a >= b+c || b >= a+c || c >= a+b)
         return(INVALID);
   if (a == b && b == c)
         return(EQUILATERAL);
   if (a == b || a == c || b == c)
         return(ISOSCELES);
   return(SCALENE);
}
```