Testing and Error Detection and Correction

# Elementary

The purpose of program testing is to test the code and to find any errors in the code, which cause unexpected results in the program.

Unit testing is usually done by the programmer first, but may also be tested later by a software tester (someone with specific skills in program testing). To do this, the programmer will run the program using a set of test data. The programmer should already know what the program will do with each of the sets of data. The programmer compares the actual program output with the expected output. If the two sets of data don’t match, the programmer then attempts to find the cause for the difference, which may be due to program errors.

# Basic

1. Syntax Errors: Like grammatical errors in code, such as forgetting a curly bracket.
   1. Objective-C

NSString \*string = “This is a string”;

*Error: Missing @ symbol before “This is a string”. Should be @”This is a string”.*

* 1. Objective-C

for(int i = 0; i < 10; i++{

NSLog(@”%i”, i);

}

*Error: Missing Right Parenthesis*

1. Logical Errors: A miss understanding of what has to be done by the programmer, such as using an incorrect algorithm.
   1. Multiplying when trying to divide (Objective-C)

int value1 = 50;

int value2 = 2;

NSLog(@”%i”, value1\*value2);

*Expecting the output of 25 but the actual output is 100.*

* 1. Forgetting a statement (Objective-C)

-(void)withdrawAmount:(float)amount {

float currentBalance = [self getCurrentBalance];

currentBalance -= amount;

[self spitOutMoney];

}

*Forgot to update the balance of the account. The user can withdraw all the money he wants without losing any.*

1. Run-time Errors: Errors that only show up when the program is executing (running).
   1. Division by 0 (Objective-C)

float value = 1998/0;

NSLog(@”%f”, value);

*Error: Illegal operation, division by zero.*

* 1. Expecting to receive data of a specific type but receiving data of a type, which does not match. (Objective-C)

-(NSArray \*)statesInAustralia {

return @[@"ACT", @"NSW", @"NT", @"QLD", @"SA", @"TAS", @"VIC", @"WA"];

}

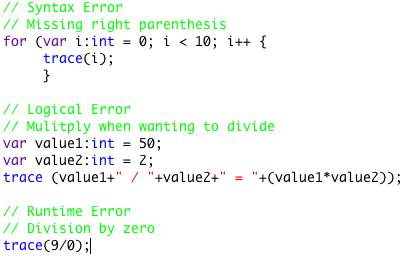
-(void)getStatesInAustralia {

NSString \*states = [self statesInAustralia];

}

*Error: Illegal conversion of NSArray to NSString.*

# Sound

Breakpoints: In software development, a breakpoint is an intentional stopping or pausing place in a program, put in place for debugging purposes. It is also sometimes simply referred to as a pause. When a breakpoint is hit, various tools are used to inspect the state of the program or alter it. Stack trace of each thread may be used to see the chain of function calls that led to the paused instruction. A list of watches allows one to view the values of selected variables and expressions. There may also be tools to show the contents of registers, loaded program modules and other information. Many processors include hardware support for breakpoints (typically instruction and data breakpoints). As an example, the x86 instruction set architecture provides hardware support for breakpoints with its x86 debug registers. Such hardware may include limitations, for example not allowing breakpoints on instructions located in branch delay slots. This kind of limitation is imposed by the microarchitecture of the processor and varies from processor to processor. Without hardware support, debuggers have to implement breakpoints in software.

# Thorough

**Logical Error:** Accidentally multiplying instead of dividing. This is a good example because firstly it was in the textbook, and because it can happen due to pure carelessness and may be hard to find in the code.

**Syntax Error:** Missing right parenthesis to close the parameters for the *for* statement.

This is a good example because forgetting to place parenthesis is a very common mistake amongst starting programmers.

**Runtime Error:** Accidentally dividing by zero. This is a good example because it may happen at the end of a series of math statements or generating random numbers, which include zero as an output.

# Extensive

Input(s): Number(s) = 4.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Line Number | Number | Count | Temp | Conditions | Input/Output |
| 1 | 4 |  |  |  | Input: Number = 4 |
| 2 |  | 0 |  |  |  |
| 3 |  |  |  | 0 < 4 ? is T |  |
| 4 |  |  | 0+1 = 1 |  |  |
| 5 |  |  |  |  | Output: Temp = 1 |
| 6 |  | 1 |  |  |  |
| 3 |  |  |  | 1 < 4 ? is T |  |
| 4 |  |  | 1+1 = 2 |  |  |
| 5 |  |  |  |  | Output: Temp = 2 |
| 6 |  | 2 |  |  |  |
| 3 |  |  |  | 2 < 4 ? is T |  |
| 4 |  |  | 2+1 = 3 |  |  |
| 5 |  |  |  |  | Output: Temp = 3 |
| 6 |  | 3 |  |  |  |
| 3 |  |  |  | 3 < 4 ? is T |  |
| 4 |  |  | 3+1 = 4 |  |  |
| 5 |  |  |  |  | Output: Temp = 4 |
| 6 |  | 4 |  |  |  |
| 3 |  |  |  | 4 < 4 ? is F |  |
| 7 |  |  |  |  | Output: Count = 4 |

The output if the user enters 4 will be as follows:

Temp = 1

Temp = 2

Temp = 3

Temp = 4

Count = 4