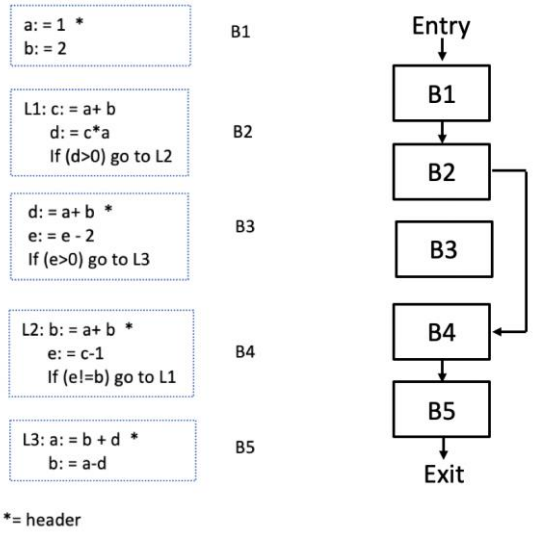
ECE/CS 4434/6434
Homework 6
Due Date: Friday, October 25, 4:59 PM

Reading: Lectures 12 – 14

Problem 1 (20 pts) **–** Below, you see a set of basic blocks B1-B5 in a program's code on the left. An example sequence of the execution of the blocks in *one run of the program* is on the right.



Part A (10 pts) Is there an error? If yes, find it in the execution sequence.

Yes, there is an error, the expected execution sequence would jump (go to) L1 after reaching the B4 block as e!=b but in the suggested sequence of execution the instruction pointer continues through B5 and exits which is not correct for the input a =1 and b=2.

Part B (5 pts) Show the correct execution sequence for the given code.

The correct execution is as follows: $B1 \rightarrow B2 \rightarrow B4 \rightarrow B2 \rightarrow B4 \rightarrow B2 \rightarrow B4$ ... and repeating forever as e is never equal to b at the final if condition in B4 always jumping back to B2. The variable e is always equal to b – 1 in this input configuration.

Part C (5 pts) What type of error is this? How can it be detected?

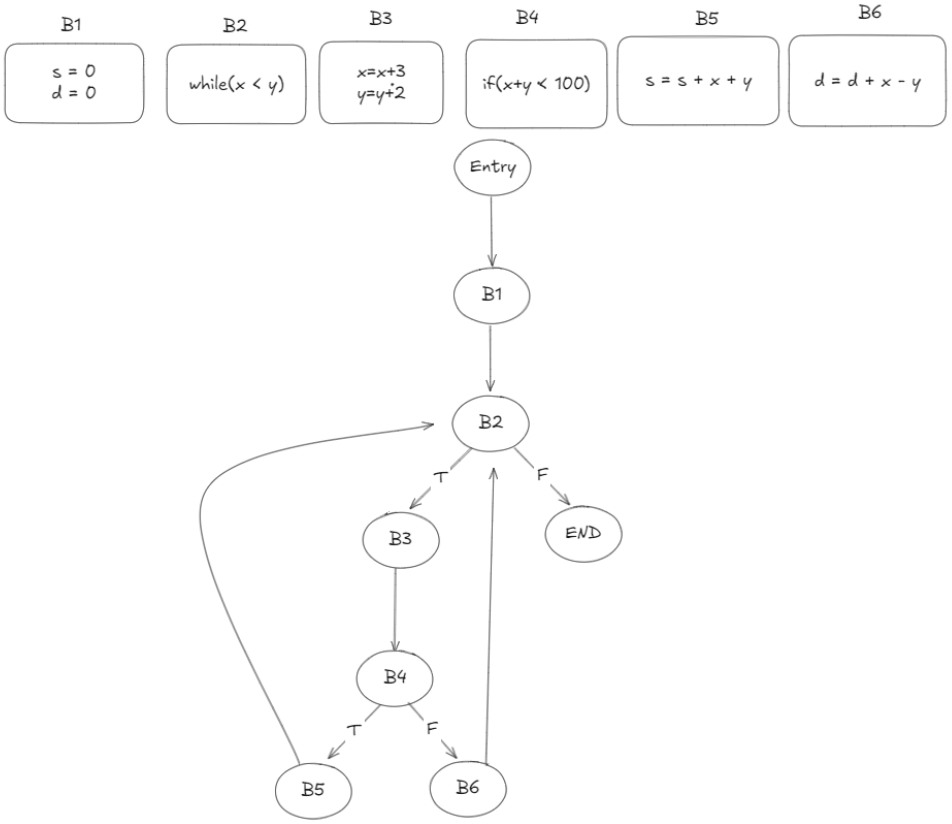
This is an infinite loop error. These types of errors are usually detected using a watchdog timer that the program must intermittently restart to ensure continued execution. Because this program is clearly looping, the timer would never get reset and the error would be discovered.

Problem 2 (50 pts) **–** You are given the following code and are asked to perform control flow checks to detect its normal vs. erroneous executions.

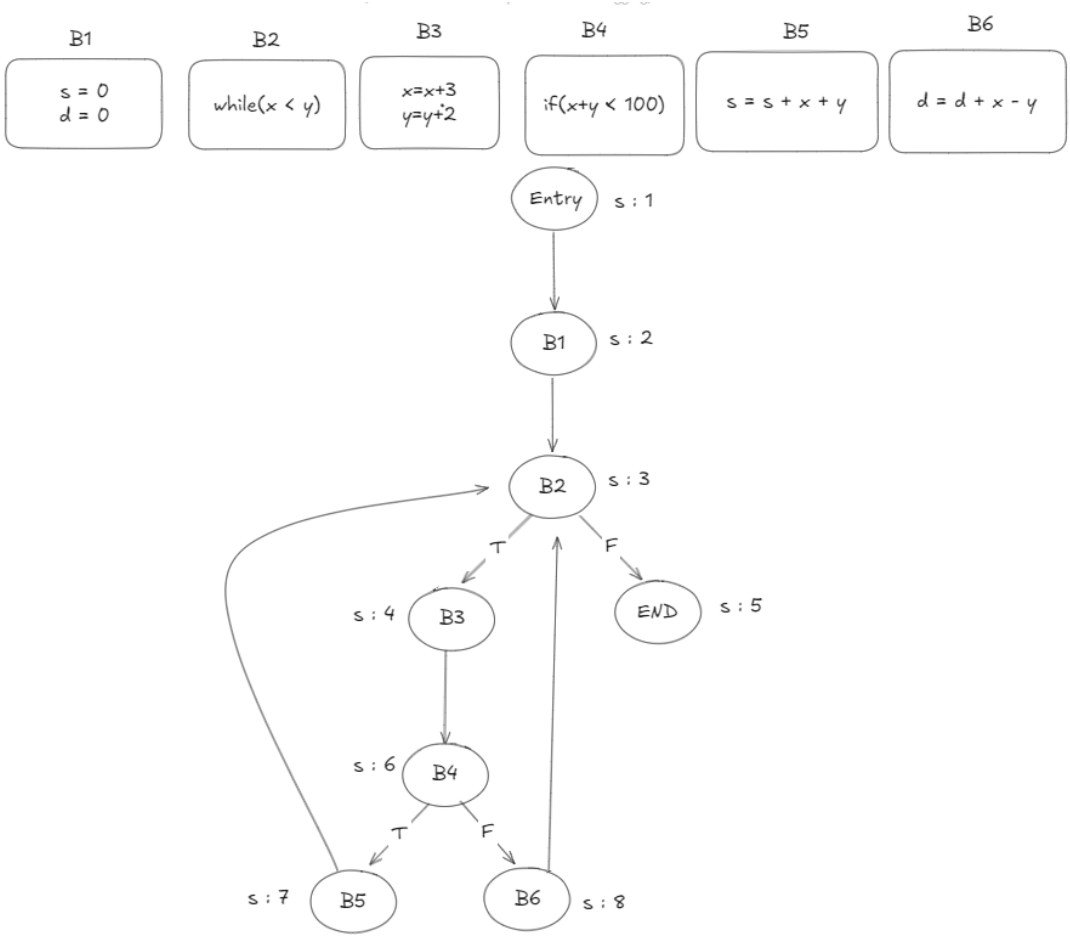
```
s: = 0;
d: = 0;
```

```
while (x<y) {
  x: = x+3;
  y: = y+2;
  if (x + y < 100)
    s: = s + x + y;
  else
    d: = d + x - y;
}
```

Part A (20 pts) Draw this program's Control Flow Graph (CFG) under normal execution.



Part B (10 pts) Label the nodes and edges in your graph and assign a golden (or reference) signature to each node. **Hint:** Signatures can be assigned as fixed random values or be calculated using a logic (e.g., checksum).



Part C (20 pts) Give an example of a case when the control flow is violated and show the signature differences and how they will help detect this violation.

Let's suppose a hypothetical situation where a malicious attack occurs, and the control flow of the program follows this path: $Entry \rightarrow B1 \rightarrow B2 \rightarrow B4 \rightarrow B6 \rightarrow B2 \rightarrow END$. In this scenario, the conditional while loop check was erroneously skipped. The valid execution path would be $Entry \rightarrow B1 \rightarrow B2 \rightarrow B3 \rightarrow B4 \rightarrow B6 \rightarrow B2 \rightarrow END$.

This erroneous path would be caught at B4 as the expected signature for B4 can only be 1 + 2 + 3 + 4 = 10. The erroneous path would calculate as 1 + 2 + 3 = 6. 6 is not equal to 10 and thus we would catch a control flow error at B4 in the erroneous path. This would likely be implemented as a lookup table for each node where the lookup returns possible sequence values (in the cases where there are multiple paths to that node).

Problem 3 (30 pts) – You are provided with a code snippet that implements a certain algorithm for sorting the elements in an array. You do not have to write the sorting algorithm yourself but only verify its correct execution.

Part A (20 pts) – Show an example of using runtime assertions to verify whether: (i) the input data is uncorrupted and (ii) the sorting algorithm output is correct. Assume you have access to a library function **`assert().`**

```
void Sort(int b[], int numVariables){

     /* your code for implementing runtime assertions*/

     for (int i = 0; i < numVariables; i++)
     {
          assert(sizeOf(b)/sizeOf(b[0]) = numVariables);
     }

     /* sorting algorithm */

     /* your code for implementing runtime assertions*/

     for (int i = 1; i < numVariables; i++)
     {
          assert(b[i - 1] <= b[i]);
     }

     return 0;
}
a = {2, 1, 9, 0, 0, 1, 3, 6};
Sort (a, 8);
```

Part B (10 pts) – How would you use the same checks done in Part A to implement a recovery block mechanism? Assume you have access to a function implementing an alternative sorting algorithm **`backup_sort(int b[], int numVariables).`**

I would modify the check for the list being sorted as follows:

```
     bool is_sorted = true;
     for (int i = 1; i < numVariables; i++)
     {
          if(b[i - 1] > b[i]){
               is_sorted=false;
          };
     }

     if(!is_sorted){
          backup_sort(b, num_variables);
```

```
      }

      for (int i = 1; i < numVariables; i++)
      {
            assert(b[i - 1] <= b[i]);
      }

      return 0;
}
a = {2, 1, 9, 0, 0, 1, 3, 6};
Sort (a, 8);
```