Looping (III)

Outline

- Motivation

1. More Examples
2. Nested Loops – A loop inside another loop
3. Common Mistakes of Nested Loops
4. A Challenging Example (Selection Sort)
5. Break and continue
6. In-class Exercise

Motivation

- In our previous lectures and labs, we have learned how to use **while** loop and **for** loop.

- We now move to more advanced usage of loops.

- Loops are VERY IMPORTANT as they enable solutions and algorithms for many problems.

1. More Examples (Example #1)

- Read **100** integers between **1** and **10** (inclusive) from the user and output the frequency of each number.

```
1  // freq[x-1] will store the frequency of x, where x = 1,…,10
2  int freq[10] = { 0 };    // Initialize all elements to 0
3  int i, input;
4
5  for (i = 0; i < 100; i++) {
6      scanf("%d", &input);
7
8      // Assume all inputs are between 1 and 10 (inclusive)
9      freq[ input - 1 ]++; // corresponding to counters [0]-[9]
10 }
11
12 for (i = 0; i < 10; i++)
13     printf("Frequency of %d: %d\n", i+1, freq[i]);
```

```c
// freq[x-1] will store the frequency of x, where x = 1,…,10
int freq[10] = { 0 };    // Initialize all elements to 0
int i, input;

for (i = 0; i < 100; i++) {
    scanf("%d", &input);

    // Assume all inputs are between 1 and 10 (inclusive)
    freq[ input - 1 ]++; // corresponding to counters [0]-[9]
}

for (i = 0; i < 10; i++)
    printf("Frequency of %d: %d\n", i+1, freq[i]);
```

```
6
8
4
6
5
...
```

100 inputs

```
Frequency of 1: 0
Frequency of 2: 20
...
Frequency of 9: 45
Frequency of 10: 1
```

1. More Examples (Example #2)

- Read **100** integers between **1** and **100** (inclusive) from a user and output the frequency of the numbers falling into these ranges: 1-10, 11-20, 21-30, …, 91-100

- In this example, we can map the input to the array index as

  f(x) = 0, *if* $1 \le x \le 10$
  1, *if* $11 \le x \le 20$
  …
  9, *if* $91 \le x \le 100$

  or

  f(x) = (x - 1) / 10 (note: integer division)

  - *In loops, it is always important to determine the trend!*

1. More Examples (Example #2)

```
1  int freq[10] = { 0 };    // Initialize all elements to 0
2  int i, input, index;
3
4  for (i = 0; i < 100; i++) {
5      scanf("%d", &input);
6
7      // Assume all inputs are between 1 and 100 (inclusive)
8      index = (input - 1) / 10;
9      freq[ index ]++;
10 }
11
12 for (i = 0; i < 10; i++)
13     printf("Frequency of %d-%d: %d\n",
14        i*10 + 1, i*10 + 10, freq[i]);
15
```

(Ask yourself: How do the input and output look like?)

1. More Examples (Example #2)

```
11
12 for (i = 0; i < 10; i++)
13     printf("Frequency of %d-%d: %d\n",
14         i*10 + 1, i*10 + 10, freq[i]);
15
```

Dry-running this loop…

i	i*10+1	i*10+10	freq[i]
0	1	10	the count for 1-10
1	11	20	the count for 11-20
...			
9	91	100	the count for 91-100

Observe how these numbers are generated from i

2. Nested Loops – A Loop Inside Another Loop

```
1  int i, j;
2  for (i = 1; i <= 3; i++) {
3      for (j = 1; j <= 4; j++) {
4          printf("%d %d\n", i, j);
5      }
6  }
```

```
1 1
1 2
1 3
1 4      } i = 1
2 1
2 2
2 3
2 4      } i = 2
3 1
3 2
3 3
3 4      } i = 3
```

↑ i ↑ j

- For each **outer loop** iteration, the **inner loop** iterates for 4 times.
  - Total # of printf() = 3 * 4 = 12 times
- The whole loop is considered as only ONE statement, i.e. no semi-colon (;) required

2.1. Outer Loop and Inner Loop(s)

- A nested loop always has an **outer loop** and one or more **inner loops**

- Assuming a simple nested loop of 2 levels
  - **Outer loop** represents repetition at the topmost level
  - The inner loop **represents a repetition that repeats in every outer loop**

2.1. Outer Loop and Inner Loop(s)

- A good example would be your university life, which involves nested repetition:
  - You will likely study for 4 years
  - Each year you will have 2 semesters (simplified)
  - Each semester you will have 13 weeks of study (simplified)

- **Question:** Which one of them is the outer loop? Inner loop(s)?

2.1. Outer Loop and Inner Loop(s)

```
1  int year, sem, week;
2  for (year = 1; year <= 4; year++) {
3      printf("Year %d\n", year);
4  }
5
6
7
8
9
10
```

The outer loop is obviously the 4 years you will spend in the university. In the example above, we model your 4 years of university life in C.

2.1. Outer Loop and Inner Loop(s)

```
1  int year, sem, week;
2  for (year = 1; year <= 4; year++) {
3      for (sem = 1; sem <= 2; sem++) {
4          printf("Year %d sem %d\n", year, sem);
5      }
6  }
7
8
9
10
```

In each year, you will have 2 semesters. So the repetitions of semester is modelled by the inner loop here.

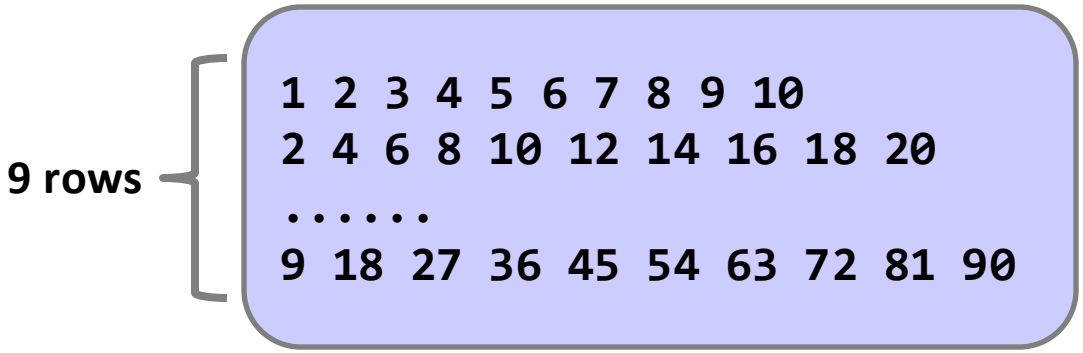
2.1. Outer Loop and Inner Loop(s)

```
1  int year, sem, week;
2  for (year = 1; year <= 4; year++) {
3     for (sem = 1; sem <= 2; sem++) {
4        for (week = 1; week <= 13; week++) {
5           printf("Year %d sem %d week %d\n",year,sem,week);
6        }
7     }
8  }
9
10
```

If we go further, we have roughly 13 weeks in each semester. So we can have a further inner loop to the semester loop. This makes our nested loop a 3-layer one.

2.2. Nested Loops (Example #1)

- Objective: To output a *multiplication table* in the following format:



- What are the things being repeated?
  - There are 9 rows (**outer loop**)
  - Each row contains 10 numbers (**inner loop**)

2.2. Nested Loops (Example #1)

```
1  int i, j;
2  for (i = 1; i <= 9; i++) {          // 9 rows
3
4      for (j = 1; j <= 10; j++) {     // 10 numbers per row
5          printf("%d ", _____________);
6      }
7
8      printf("\n");     // Newline appears only once per row
9  }
```

- What expression, in terms of `i` and `j`, will yield the numbers we need?

2.2. Nested Loops (Example #1)

```
1  int i, j;
2  for (i = 1; i <= 9; i++) {          // 9 rows
3
4      for (j = 1; j <= 10; j++) {     // 10 numbers per row
5          printf("%d ", i * j);
6      }
7
8      printf("\n");     // Newline appears only once per row
9  }
```

- Often, the numbers we want to generate inside a loop (or nested loops) can be expressed in terms the loop variables.

2.3. Nested Loops (Example #2)

```
1   // A and B represent two 3x3 matrices
2   int A[3][3] = { { 1, 2, 3 }, { 0, -1, 2 }, { 0, 0, 1 } };
3   int B[3][3];   // To store the transpose of matrix A
4   int i, j;
5
6   // Store transpose of A in B
7   for (i = 0; i < 3; i++)
8       for (j = 0; j < 3; j++)
9           B[j][i] = A[i][j];
10
11  // Print matrix A
12  for (i = 0; i < 3; i++) {
13      for (j = 0; j < 3; j++)
14          printf("%4d", A[i][j]);
15      printf("\n");
16  }
17  printf("\n");
18
19  // Repeat the above for-loop for matrix B
20  ...
```

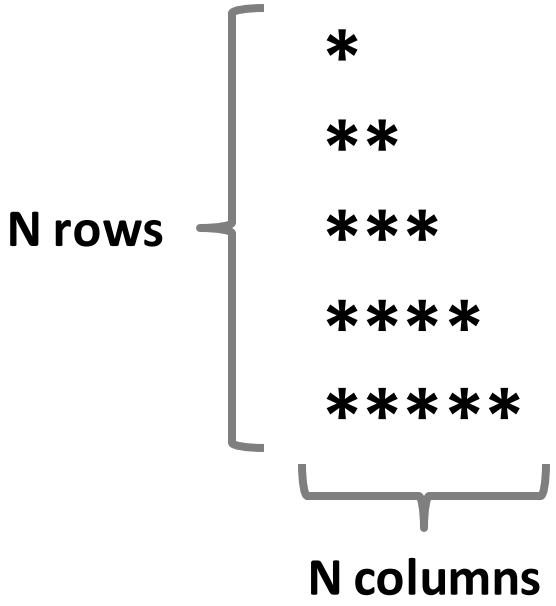
Example: Using a nested loop to handle a 2D array

```
   1   2   3
   0  -1   2
   0   0   1

   1   0   0
   2  -1   0
   3   2   1
```

2.4. Nested Loops (Example #3)

- Objective: Given a positive integer *N*, output a triangle in the following format (e.g., when *N* = 5):



2.4. Nested Loops (Example #3)

```
1   int i, j, N;
2
3   printf("N = ? ");
4   scanf("%d", &N);
5
6   for (i = 1; i <= N; i++) {          // N rows
7
8
9
10
11  }
```

To write a nested loop, we first need to consider what constitutes our **outer loop**.

In this example problem, we need to print a total of *N* lines, so an outer loop to count for *N* times would be good.

2.4. Nested Loops (Example #3)

```
1   int i, j, N;
2
3   printf("N = ? ");
4   scanf("%d", &N);
5
6   for (i = 1; i <= N; i++) {          // N rows
7       // Step 1: print i number of stars
8       // i.e. 1 star in line 1, 2 stars in line 2... etc.
9       // Step 2: print a new line character
10
11  }
```

Then we think about what to do in our **inner loop**. In this case, we need an inner loop to print the number of stars according to the line number.
We also need to print a ***new line character*** after finishing each row.

2.4. Nested Loops (Example #3)

```
1   int i, j, N;
2
3   printf("N = ? ");
4   scanf("%d", &N);
5
6   for (i = 1; i <= N; i++) {          // N rows
7       for (j = 1; j <= i; j++) {      // row #i has i stars
8           printf("*");
9       }
10      printf("\n");
11  }
```

```
N = ? 6
*
**
***
****
*****
******
```

This is already the solution.

For beginners, <u>always make sure you plan for the **outer loop** and **inner loop**</u> before you start writing your code or pseudocode.

2.5. Nested Loops (Example #4)

- **Objective:** To find all sets of integers that satisfy the following equality:

$$x^2 + y^2 + z^2 = 1000000, \qquad 0 \le x, y, z \le 1000$$

- One possible (quick and dirty) solution is to try all possible values for x, y, and z.

2.5. Nested Loops (Example #4)

```
1  int x, y, z;
2
3  for (x = 0; x <= 1000; x++) {
4      for (y = 0; y <= 1000; y++) {
5          for (z = 0; z <= 1000; z++) {
6              if (x*x + y*y + z*z == 1000000)
7                  printf("%d, %d %d\n", x, y, z);
8          }
9      }
10 }
```

3. Nested Loop Common Mistakes

- Always use a different index in each layer of nested loops
  - By convention we often use `i`, `j` and k
  - What happens if you mistakenly used the same index for different layers of a nested loop? Check out a modified Example #1 below:

```
int i, j;
for (i = 1; i <= 3; i++) {
    for (i = 1; i <= 4; i++) {
        printf("%d\n", i);
    }
}
```

Can you dry run the program to see the expected output?

3. Nested Loop Common Mistakes

- Be aware of the placement of new lines

```
int i, j;
for (i = 1; i <= 3; i++) {
    for (j = 1; j <= i; j++) {
        printf("*", i);
    }
}
printf("\n");
```

When should the output finish printing a line?

- Indentation and relationship between loops

```
for (i = 1; i <= 3; i++)
    for (j = 1; j <= i; j++) {
        ...
}
} // where does this } come from??
```

4. Challenging Example (Selection Sort)

- **Selection Sort** is a sorting method that involves a simple 2-layer nested loop
- Selection Sort is among the first algorithm you learn in this course

- An algorithm is a set of instructions that one can follow to solve a general set of problems

4. Challenging Example (Selection Sort)

```
int A[6] = { 4, 5, 7, 2, 9, 1 };
```

Iteration #	*Step 1:* **Subarray to be processed:** N=6 **A[*i*] … A[*N-1*]**	*Step 2:* **Locate the smallest # in the sub-array**	*Step 3:* **Swap the smallest # with A[*i*]**
i = 0	4 5 7 2 9 1 (i)	4 5 7 2 9 1	**1** 5 7 2 9 4
i = 1	**1** 5 7 2 9 4 (i)	**1** 5 7 2 9 4	**1 2** 7 5 9 4
i = 2	**1 2** 7 5 9 4 (i)	**1 2** 7 5 9 4	**1 2 4** 5 9 7
i = 3	**1 2 4** 5 9 7 (i)	**1 2 4** 5 9 7	**1 2 4 5** 9 7
i = 4	**1 2 4 5** 9 7 (i)	**1 2 4 5** 9 7	**1 2 4 5 7** 9

This part is sorted!

4. Challenging Example (Selection Sort)

```
// Pseudocode:
// Given an array A[] of size N

for i = 0 to N-2 { //step1
    minPos = position of the smallest element
             among A[i] … A[N-1]; //step2

    Swap A[i] with A[minPos]; //step3
}

// Note: When there is only one element left, we do not
// need to perform the steps in the loop.
// Therefore, the loop only needs to iterate
// N-1 times (i.e., from 0 to N-2).
```

```
1   // Suppose A[] contains N numbers. The following segment of
2   // code sorts the data in the array so that
3   //      A[0] <= A[1] <= ... <= A[N-2] <= A[N-1]
4
5   int i, j, minPos, tmp;
6   for ( i = 0; i < N-1; i++ ) { //step1
7
8       // Locate the smallest element among A[i] … A[N-1],
9       // and store the location of it in minPos. //step2
10      minPos = i;
11      for ( j = i+1; j < N; j++ )
12          if ( A[ j ] < A[ minPos ] )
13              minPos = j;
14
15      // Swap A[i] with A[minPos] only if necessary //step3
16      if (minPos != i) {
17          tmp = A[ i ];
18          A[ i ] = A[ minPos ];
19          A[ minPos ] = tmp;
20      }
21  }
22
```

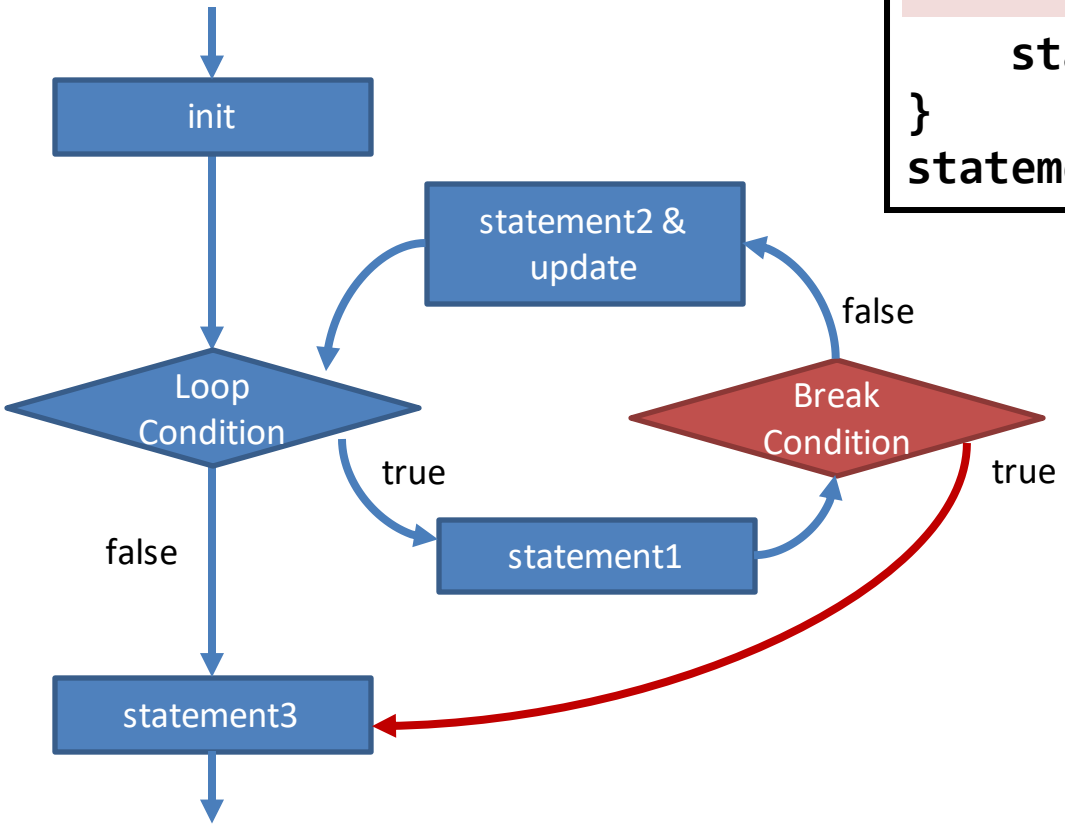
Challenging Example (Selection Sort)

5. Break and continue

- **`break`** statement

- **`continue`** statement

5.1. What is **break**?




```
for (init; LoopCondition; update) {
    statement1;
    if (BreakCondition)
        break;
    statement2;
}
statement3;
```

Early Exit in a Loop

The **break** statement, when executed, causes the program to *leave* the **closest enclosing loop** immediately.

5.2. Examples of **break** statement

```
1   int input, sum = 0; // To store input value and their sum
2
3   while (1) {         // Use "break" to stop the loop instead
4
5       printf("Input: ");
6       scanf("%d", &input);
7
8       if (input == 0)
9           break;      // Loop will stop when input value is 0
10
11      sum = sum + input;
12  }
13
14  printf("Sum = %d\n", sum);
15
```

Rewriting the example from **Looping (I)**.

5.2. Examples of **break** statement

- Using **break** to stop a for loop

```
1   int i;
2
3   for (i = 0; i < 10; i++) {
4       if (i == 3)
5           break;
6
7       printf("%d\n", i);
8   }
9
10  printf("Bye!\n");
11
```

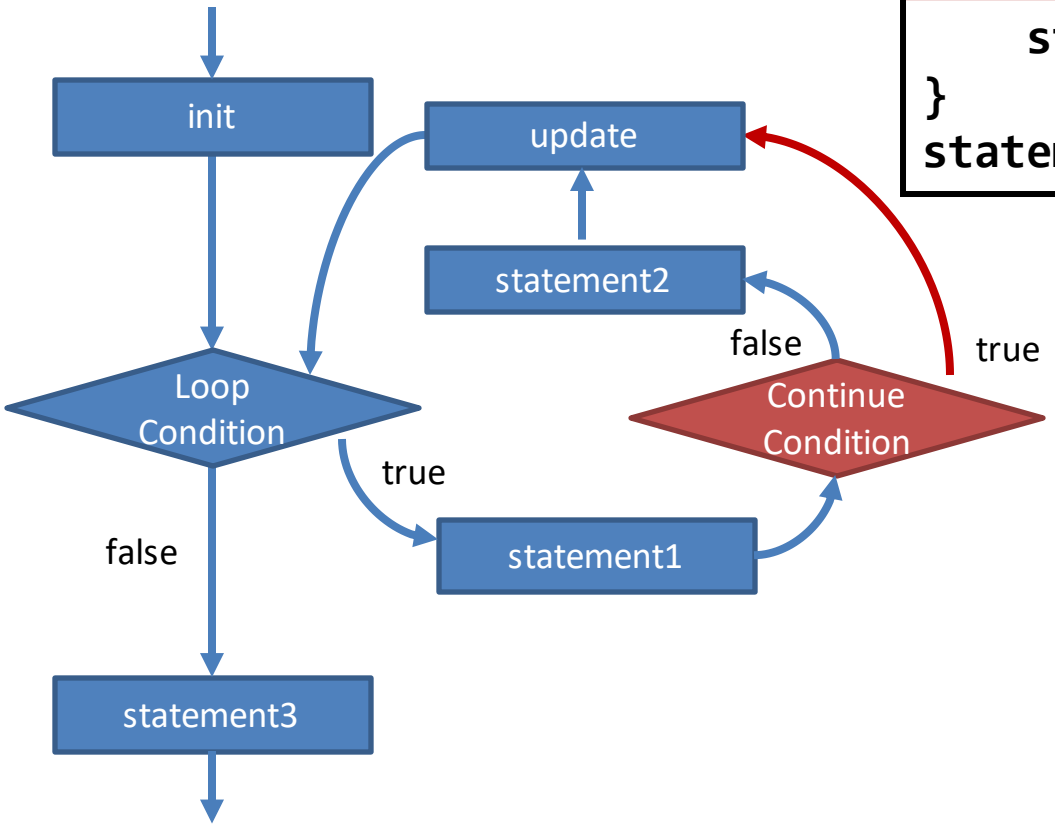
```
0
1
2
Bye!
```

5.2. Another Example

To compare if two arrays have the same content, we need to compare their elements one by one. *As soon as* we encounter the first difference between the two arrays, we can stop early using **`break`**!

```
1   int A[10], B[10], i;
2   int hasSameContent;
3   …           // Suppose A and B are assigned some values here
4
5   hasSameContent = 1;     // Assume they have the same content
6   for (i = 0; i < 10; i++) {
7       if (A[i] != B[i]) {
8           hasSameContent = 0; // a counter-example is found!
9           break;
10      }
11  }
12  // After the loop, if hasSameContent remains to be 1,
13  // then the arrays have the same content
```

5.3. What is continue?

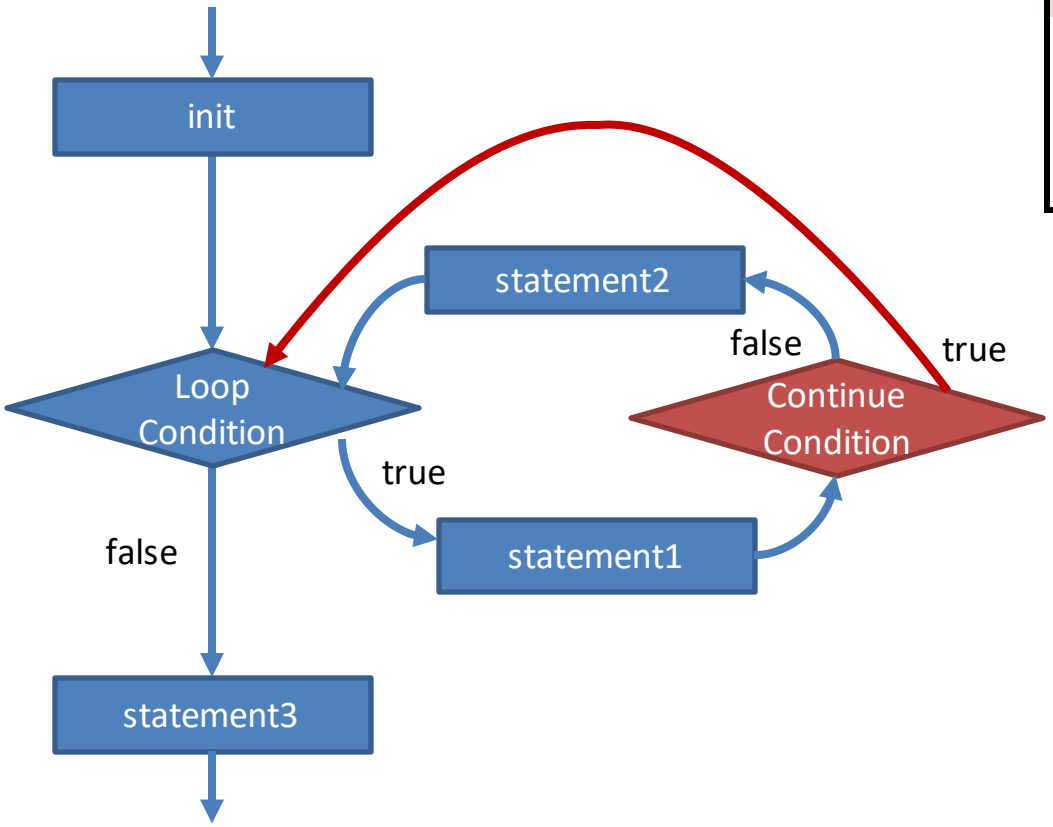



```
for (init; LoopCondition; update) {
    statement1;
    if (continueCondition)
        continue;
    statement2;
}
statement3;
```

Continuation in a Loop

The **continue** statement, when executed, causes the program to *skip the remaining statements* in the **closest enclosing loop** for the current iteration.

5.3. What is continue?




```
while ( LoopCondition ) {
    statement1;
    if (continueCondition)
        continue;
    statement2;
}
statement3;
```

The while-loop version

5.4. Examples of `continue` statement

- for-loop version

```
1   int i;
2
3   for (i = 0; i < 10; i++) {
4       if (i == 3)
5           continue;
6
7       printf("%d\n", i);
8   }
9
10  printf("Bye!\n");
11
```

```
0
1
2
4
5
6
7
8
9
Bye!
```

5.4. Examples of `continue` statement

- while-loop version

```
1   int i;
2
3   i = 0;
4   while (i < 10) {
5       if (i == 3) {
6           i++;
7           continue;
8       }
9       printf("%d\n", i);
10      i++;
11  }
12  printf("Bye!\n");
```

```
0
1
2
4
5
6
7
8
9
Bye!
```

The **i++** at *line 10* will be skipped when **continue** is executed. Without the **i++** at *line 6*, the loop will iterate forever.

6. In-Class Exercise

- Suppose now I want to write a C program to compute and **find out all prime numbers** in the range of **2 to 1000**
- How are you going to tackle the problem?
  - What will be our **outer loop**?
  - How about our **inner loop**?
- *Hint: In this case, you can first write your inner loop first and make sure it is correct before you proceed with the outer loop.*

Summary

- More examples on looping
- Nested loops
  - Syntax and examples
  - Common mistakes
  - A challenging example: selection sort
- `break` and `continue`

Reading Assignment

- C: How to Program, 8th ed, Deitel and Deitel
- Chapter 3 Structured Program Development in C
  - Sections 3.7 – 3.10
- Chapter 4 C Program Control
  - Sections 4.1 – 4.6, 4.9